ISSN 0105-8517

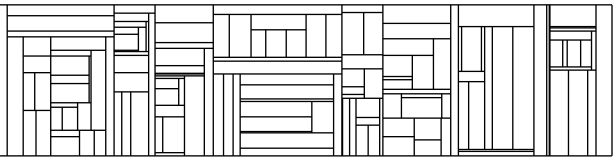
Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools

Aarhus, Denmark, October 8-11, 2004

Kurt Jensen (Ed.)

DAIMI PB - 570
October 2004

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF AARHUS
IT-Parken, Aabogade 34
DK-8200 Aarhus N, Denmark



Preface

This booklet contains the proceedings of the Fifth Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, October 8-11, 2004. The workshop is organised by the CPN group at Department of Computer Science, University of Aarhus, Denmark. The papers are also available in electronic form via the web pages: http://www.daimi.au.dk/CPnets/workshop04/

Coloured Petri Nets and the CPN tools are now used by 1400 users in 85 countries all over the world. The aim of the workshop is to bring together some of the users and in this way provide a forum for those who are interested in the practical use of Coloured Petri Nets and their tools.

The submitted papers were evaluated by a programme committee with the following members:

Wil van der Aalst, Netherlands
Jonathan Billington, Australia
Jörg Desel, Germany
Joao M. Fernandes, Portugal
Jorge de Figueiredo, Brazil
Nisse Husberg, Finland
Kurt Jensen, Denmark (chair)
Ekkart Kindler, Germany
Lars M. Kristensen, Denmark
Charles Lakos, Australia
Tadao Murata, USA
Daniel Moldt, Germany
Laure Petrucci, France
Karsten Schmidt, Germany
Rüdiger Valk, Germany
Lee Wagenhals, USA
Jianli Xu, Finland
Wlodek Zuberek, Canada

The programme committee has accepted 13 papers for presentation. Most of these deal with different projects in which Coloured Petri Nets and their tools have been put to practical use – often in an industrial setting. The remaining papers deal with different extensions of tools and methodology.

The papers from the first four CPN Workshops can be found via the web pages: http://www.daimi.au.dk/CPnets/. After an additional round of reviewing and revision, some of the papers have also been published as a special section in the International Journal on Software Tools for Technology Transfer (STTT). For more information see: http://sttt.cs.uni-dortmund.de/

Kurt Jensen

Table of Contents

Formal Specification and State Space Analysis of an Operational Planning Process

Brice Mitchell[1], Lars M. Kristensen[2], Lin Zhang[1]

[1] Command and Control Division, Defence Science and Technology Organisation, Edinburgh, SA 5111, Australia
{Brice.Mitchell,Lin.Zhang}@dsto.defence.gov.au
[2] Department of Computer Science, University of Aarhus, IT-Parken, Aabogade 34, DK-8200 Aarhus N, Denmark
kris@daimi.au.dk

Abstract. Formal models of business processes support performance and behavioural analysis of the processes for continuous improvement. Formal models are also useful in guiding the development of software tools to support the processes. This paper presents a formal model of the operational planning process used in the Deployable Joint Force Headquarters of the Australian Defence Force. The formal process model was developed using Coloured Petri Nets (CPN or CP-nets) and the supporting Design/CPN computer tool. The constructed CPN model has allowed the planning process to be validated and analysed using simulation and state spaces. State space analysis was conducted using full state spaces and the sweep-line state space reduction method.

Topics: Business process modelling, Design and analysis of business processes, Coloured Petri Nets

1. Introduction

Workflow modelling [18] based on formal methods such as Petri Nets [16] for rigorous specification and analysis of business processes is becoming applied more and more in practice [19]. Business processes in a military organisation take the form of Standard Operating Procedures (SOP), guided by principles expressed in a *doctrine*. The representation and analysis of military business processes for continuous improvement is of great importance as the military activities allow little inefficiency or ambiguity.

This paper presents the formal specification and analysis of the business process for planning at the *Deployable Joint Force Headquarters* (DJFHQ) of the *Australian Defence Force* (ADF). The DJFHQ is a Joint Headquarters (HQ) for the Army, Navy and Air Force of the ADF. It can be deployed for offshore military operations and has been deployed for operations such as East Timor in 1999. The doctrine that the DJFHQ uses for planning is the Joint Military Appreciation Process (JMAP) [1]. The HQ has a set of SOPs [4] that describe the DJFHQ implementation of the JMAP principles in detail. The JMAP and associated SOPs are described in several natural lan-

guage documents, but these documents do not describe the planning process formally nor completely. As military require efficiency and clarity in operations, it is beneficial that the process is formalised especially for the purposes of training new staff officers, analysing the process for improvement, and guiding the development of software tools to support the process.

The project reported in this paper is aimed at contributing to the development of a robust operational planning process at the HQ based on the doctrine and current SOP. The project consisted of three steps. The first step was to specify the DJFHQ planning process using Coloured Petri Nets (CPNs or CP-nets) [10,11,12,13] and the supporting Design/CPN computer tool [5]. This step involved liaising with staff officers from DJFHQ to ensure that the CPN model properly reflected the planning process. The next step was to validate the constructed CPN model and conduct initial analysis of the planning process using simulation. The third step was to conduct state space analysis of the CPN model. The basic idea behind state spaces [11] (also called reachability trees/graphs or occurrence graphs) is to compute a directed graph (called the state space), which represents all possible executions of the CPN model. These states can then be traversed to find qualitative and quantitative properties of the process. This type of analysis led to a better understanding of the planning process, and enabled identification of areas for improvement. In the analysis step, we also investigated the use of the sweep-line method [3] in the domain of workflow modelling. The sweep-line method exploits the progress present in systems to reclaim memory during state space exploration and thereby alleviate the state explosion problem [17].

The choice of CPNs as the modeling language in the project was based on the authors experience with CP-nets from earlier projects [14,15] in the area of operational planning. In [14], a CPN model of the DJFHQ planning process based on the observation of a training exercise was reported. The process used in the planning exercise can be seen as one of many possible implementations of the doctrine and SOP based process that we consider in this paper. The work in [15] reported a formal specification of the planning process at another HQ of the ADF, Headquarters Australian Theatre (HQAST), using CPNs. The findings from these earlier projects was that: 1) CPNs enabled complex processes to be decomposed by the use of hierarchical constructs, something which is important for presentation purposes and to manage complexity, and 2) the state space tool of Design/CPN provided the required flexibility to implement the algorithms to analyse the planning process as per DJFHQ requirements.

This paper is organised as follows. Section 2 briefly describes the JMAP as well as the approach used in the development of the CPN process model. Section 3 provides an overview of the CPN model. Section 4 explains how the CPN model was analysed using simulation. Section 5 presents the full state space analysis of the process, while Section 6 discusses the sweep-line analysis. Finally, Section 7 gives the conclusions and discusses future work. The reader is assumed to be familiar with the basic ideas of high-level Petri Nets.

2 Model Development

The JMAP is a logical decision-making process that guides military staff in producing an operational plan. It comprises four consecutive and iterative *steps* as illustrated in Figure 1: *Mission Analysis*, *Course of Action (COA) Development*, *COA Analysis*, and *Decision and Execution*. Prior to these four steps, *Preliminary Scoping* is normally conducted to analyse the superior HQ's intent and guidance to gain an idea of the "bigger picture".

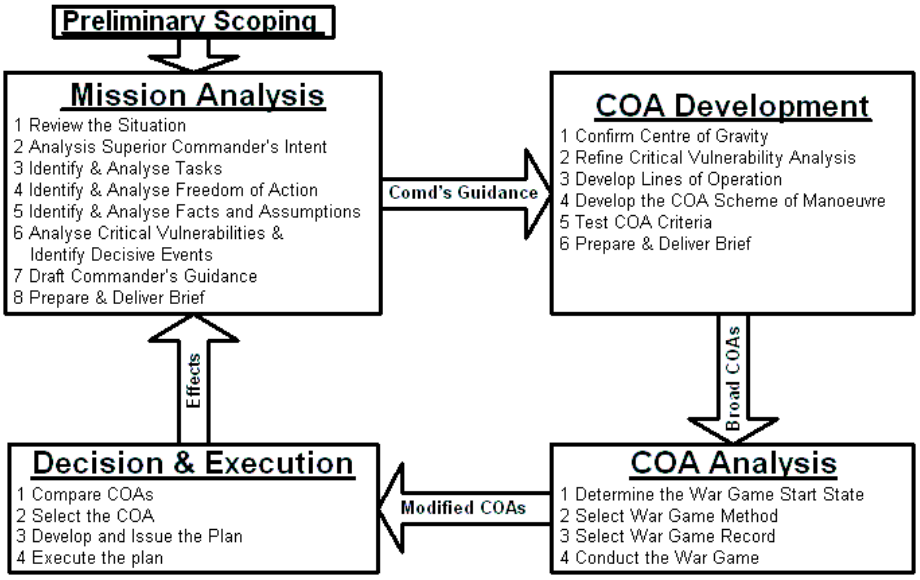


Figure 1: Joint Military Appreciation Process (JMAP).

In the *Mission Analysis* step, staff officers provide input to the process, which leads to an awareness of the situation. Staff officers analyse all mission aspects and compile a *brief* (presentation) to the Commander. The Commander, upon receipt of the brief, provides a guidance to staff for subsequent steps. The *COA Development* step consists of identifying a broad range of potential COAs that achieve the mission in accordance with the Commander's guidance (*Comd's Guidance*). The broad COAs are presented to the Commander in the COA Development Brief. The *COA Analysis* step consists of a war game where, typically, two sides are formed. One side acts as the enemy and the other side as the friendly force. The purpose of a war game is to investigate each COA by determining the risks, feasibility, strengths, and weaknesses. This provides *Modified COAs* to be used in the next step. The final step of the process is *Decision and Execution* when the Commander compares the strengths and weaknesses of each COA as revealed by the war game with assistance from staff. The result is the selection of a COA to be developed into a plan and executed. The *Effects* of this execution enables the process to start again leading to an iterative process. All JMAP steps are supported with intelligence update activities.

The highly structured nature of the JMAP suggested that the CPN model to be developed should reflect this hierarchical representation. From Fig.1, the top level CPN model would comprise *substitution transitions* to represent *Preliminary Scoping* and the JMAP steps (i.e. *Mission Analysis, COA Development, COA Analysis, and Decision & Execution*). The second level would comprise transitions to model sub-steps of each of the JMAP steps. The sub-steps are listed inside the boxes in Fig.1, and can be broken down into lower level *activities* in the process. It is important to note that the numbering of sub-steps does not impose an ordering of their occurrences. Rather, the timing of an activity is determined by the availability of required information, staff officers, and completion of other activities. Activities can occur concurrently and out of the JMAP step and sub-step order if the above conditions are satisfied. Strictly speaking, the grouping of JMAP steps and sub-steps is for the purpose of representation, and should not constrain the ordering of activities. For this reason, we consider it important to model the individual behaviour of each activity in order to study the overall and complete behaviour of the operational planning process. The execution of individual activities in the process model would then generate the overall behaviour for analysis. One of the objectives was to investigate possible execution sequences of activities in the JMAP in order to determine the most efficient allocation of staff resources.

We consider that each activity in the JMAP can be characterised with six attributes (see Fig.2): Input Information, Output Information, Prior Activities, Required Staff, Desired Staff, and Duration. If information is obtained on all six attributes for each activity in the JMAP, a CPN model can then be constructed and populated. The rest of this section briefly describes these attributes.

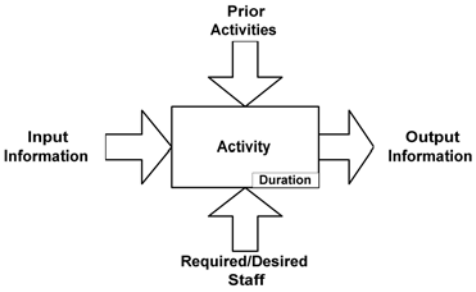


Figure 2: An activity in the JMAP.

Input Information must be available before the activity can occur, while *Output Information* is produced at the conclusion of the activity. Information is usually in the form of an electronic document (i.e., Word, Excel, or PowerPoint files) or handwritten notes, but may also be passed verbally. The output information of one activity may become input information of another activity, introducing a dependency between activities in the JMAP. *Prior Activities* are activities that must be completed before the activity can occur. Prior activities are another mechanism for modelling dependency between activities, as it is sometimes difficult to represent these dependencies through the input/output mechanism.

The *Required Staff* must be available before the activity can occur. This is defined by a set of conditions $\{c_1,\ldots,c_N\}$, where condition $c_i = (n_i , \{s_{i1},\ldots,s_{im}\})$ is satisfied if

n_i or more staff officers are available from the set $\{s_{i1},\ldots,s_{im}\}$. It is a precondition for an activity to start that all such conditions $\{c_1,\ldots,c_N\}$ for the activity are satisfied. The *Desired Staff* set can provide assistance in the activity or benefit from attending the activity. Desired staff officers attend the activity if they are available, but do not prevent the activity from occurring.

Duration is the expected length of the activity in minutes. This deterministic time is based on the available documentation and estimates from DJFHQ staff officers based on domain knowledge and experience.

3 Overview of the CPN Model

This section describes the hierarchical CPN model that has been constructed using the approach described in Section 2. Figure 3 shows the *hierarchy page* of the CPN model. Each node in Figure 3 represents a page (module) in the CPN model. An arc going from a higher-level page to a lower-level page indicates that the higher-level page contains a *substitution transition* that has the lower-level page as its associated *subpage*. The immediate subpages of the *JMAP* page represent the five steps of the JMAP as shown in Figure 1. Subpages representing the JMAP steps are divided into subpages representing the activities constituting the steps. These activities are grouped according to the logical structure of the JMAP as described in [1]. Representative pages of the CPN model will be described in the following sections, and are highlighted with a thick border in Figure 3.

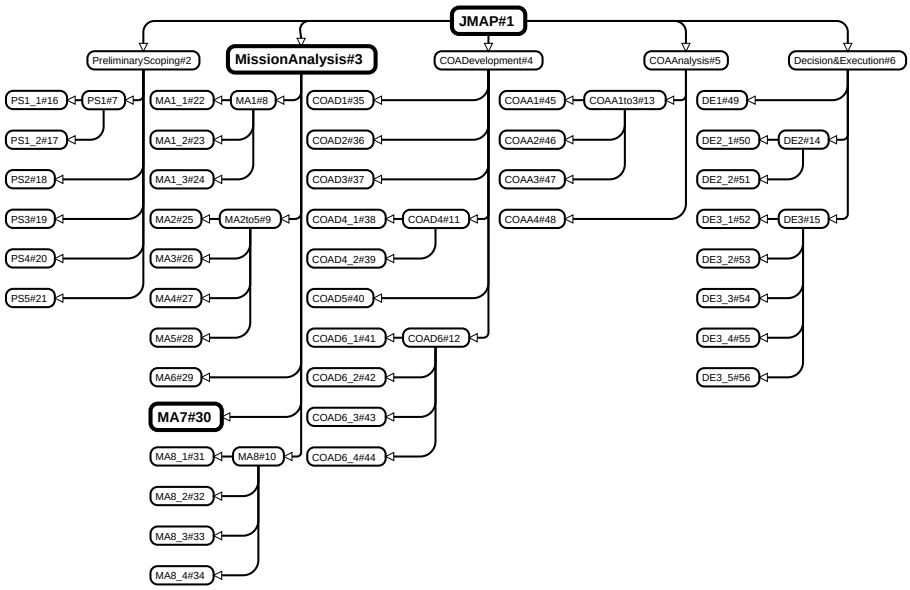


Figure 3: The hierarchy page.

3.1 The JMAP page

The *JMAP* page is the highest-level page in the CPN model and is shown in Figure 4. On this page there are five substitution transitions (indicated by the HS tag in the lower right corner of the transition) corresponding to Preliminary Scoping and the four JMAP steps. The subpage of each substitution transition in Fig.4 is the accordingly named page in Figure 3. Note that for the Decision and Execution step, the CPN model captures activities up to the production of plans. The process completes when a token of colour *Plan* is produced in the *Planning Completed* place.

There are six places on the JMAP page named *External Information*, *Input Information*, *Output Information*, *Planning Completed*, *Completed Activities*, and *Staff*. These places hold information about the process, including what information has been produced (*Output Information* place), what information is available (*Input Information* and *External Information* places), what activities have been completed (*Completed Activities* place), what staff officers are currently available (*Staff* place), and whether the process has been completed (*Planning Completed* place).

Since we require some of the produced output information to be used by other activities as input information, we define the *Input Information* and *Output Information* as *fusion places* belonging to the same *fusion set.* Fusion places are indicated by the FG tag next to the place. This implies that the places *Input Information* and *Output Information* always have the same marking.

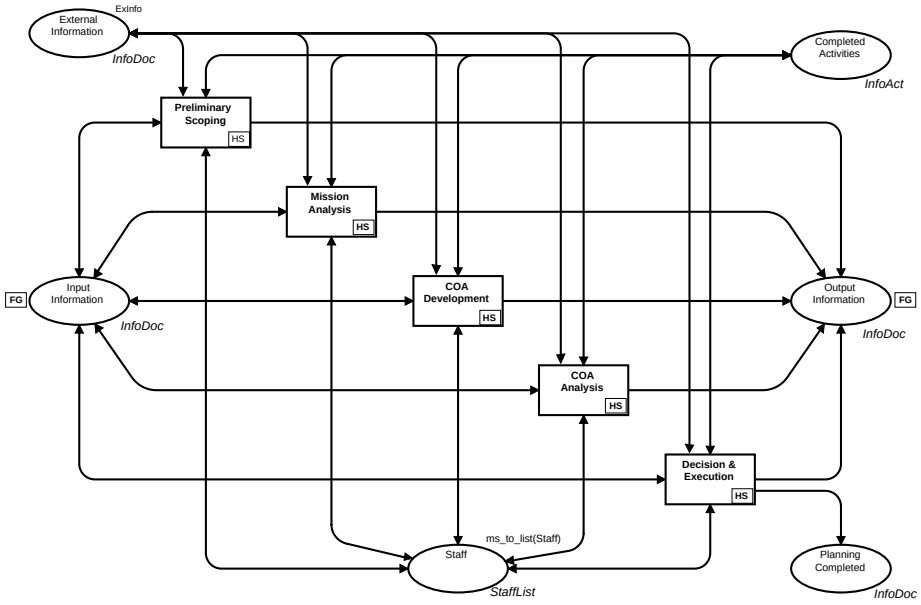


Figure 4: The JMAP page.

The *InfoDoc* colour set is an enumerated type that represents all information that can be produced in the process. The *Input Information*, *Output Information, External Information*, and *Planning Completed* places have this colour set to model what in-

formation is currently available. The *InfoAct* colour is an enumerated type that represents all possible activity names in the JMAP planning process. The *Completed Activities* place has this colour set to store all activities that have been completed in the process. The *StaffList* colour set is a list of the *Staff* colour set that is an enumerated type representing all staff officers involved in the process. The *Staff* place has the *StaffList* colour set to model staff officers that are available to activities that require them. A list type is used on the *Staff* place to make it efficient to determine which of the desired staff officers available will participate in a given activity. We will return to this issue when we present the lower level pages of the model.

Only two places on the *JMAP* page have non-empty initial markings: *External Information* and *Staff* places. The initial marking of the *External Information* place is a set of information units provided by external sources (i.e., external HQ or processes). The initial marking of the *Staff* place (*ms_to_list(Staff)*) is a list of all staff officers.

3.2 The Mission Analysis page

Fig.5 shows the Mission Analysis step. It is the subpage of the *Mission Analysis* substitution transition in Fig.4. The five *port places* (places indicated by a P tag positioned next to the them) are connected to the accordingly named *socket places* in Fig.4. Port and socket places are the mechanism by which a subpage interfaces with the superpage containing the substitution transition. The marking of a port place is always the same as it corresponding socket place. See [10] for details.

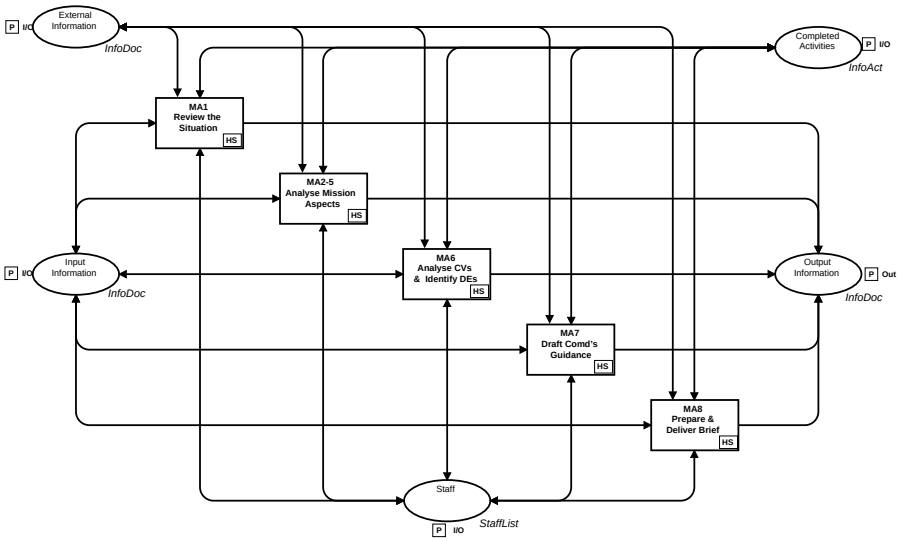


Figure 5: Example of a JMAP Step page - Mission Analysis.

The five transitions on the *Mission Analysis* page represent the group of activities that are involved in the Mission Analysis step. These transitions are also substitution transitions. The *MA6 Analyse CVs & Identify DEs* and *MA7 Draft Comd's Guidance* transitions are represented by activity pages, while the *MA1 Review Situation*, *MA2-5*

Analyse Mission Aspects, and *MA8 Prepare & Deliver Brief* transitions are represented by intermediate pages.

3.3 The Draft Commander's Guidance Page

An example of an activity page is given in Figure 6. It is the subpage of the *MA7 Draft Comd's Guidance* substitution transition on the *Mission Analysis* page from Fig.5. All activity pages contain 9 places (5 port places and 4 ordinary places), and 2 transitions (*Start Activity* and *Stop Activity*). The 5 port places relate to the accordingly named socket places on higher-level pages, and the 4 ordinary places (*Duration*, *Activity Occurring*, *Required Staff* and *Desired Staff*) represent the detailed information needed for an activity to occur.

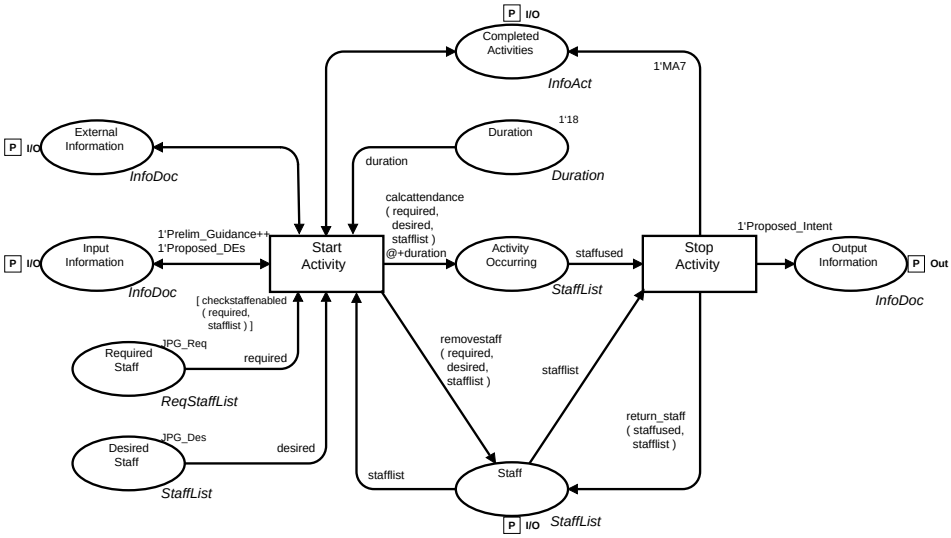


Figure 6: Example of an Activity page – MA7.

Every activity in the planning process is modelled to occur in two stages. The first stage represents the start of an activity (transition *Start Activity*), and the second stage the termination of the activity (transition *Stop Activity*). Enabling of the *Start Activity* transition requires the following conditions to be satisfied.

Firstly, necessary information from the *Input Information* and *External Information* places must be available as per inscriptions on the arcs between these two places and the *Start Activity* transition. For this example, no external information is required, and hence the arc has an empty inscription.

Secondly, necessary prior activities must have been completed. Normally, these activities are specified through the arc inscription between the *Start Activity* transition and the *Completed Activities* place. In this activity, no prior activities are specified other than implicit dependencies through input and output information. This arc therefore has an empty inscription in this case.

Thirdly, the required staff are available on the *Staff* place. The required staff are specified through the initial marking of the *Required Staff* place (*JPG_Req* in this activity) and the *required* inscription. The colour set of the *Required Staff* place is *ReqStaffList*. *ReqStaffList* is a list of the *StaffCondition* colour set, which is a product of two colour sets: *NumberReq* and *StaffList*. The *NumberReq* colour set is an integer type representing the number of staff officers that are at least required. The second part of the product *StaffList* represents candidates of the required staff. The desired staff are specified through the initial marking of the *Desired Staff* place (*JPG_Des* in this activity) and the *desired* inscription. Obviously there need to be tokens in the *Required Staff* and *Desired Staff* places for *Start Activity* to be enabled, although desired staff are optional. The condition on required staff in the *Staff* place is ensured through the transition guard: *checkstaffenabled(required, stafflist)*. This expression evaluates to true only if *required* staff is contained in the *stafflist*.

The duration of the activity is specified by the *duration* inscription and the initial marking of the *Duration* place. The *Duration* colour set is an integer type. The activity pages are the only pages that directly use the time concept of CP-nets.

When *Start Activity* occurs, tokens from the *Duration*, *Required Staff* and *Desired Staff* places are consumed. Input information, external information and completed activities are examined, and then reproduced in their respective places. The staff list is taken from the *Staff* place, and then returned to the *Staff* place after the removal of the required and desired staff through the function *removestaff(required, desired, stafflist)*. The staff officers participating in the activity are put on the output place *Activity Occurring*, and will stay there for the duration of the activity. When time has elapsed corresponding to the duration of the activity, the transition *Stop Activity* can occur. When this transition occurs, the officers that participated in the activity are returned to the *Staff* place, the information produced by the activity (*Proposed_Intent*) is added to the *Output Information* place, and a token corresponding to the activity (*MA7*) is produced on the *Completed Activities* place.

Note that each of the *Duration*, *Required Staff* and *Desired Staff* places contains exactly one token as an initial marking and tokens are not returned to these places when the *Start Activi*ty transition occurs. This implies that each activity will only occur once which is in accordance with the planning process.

4 Simulation

Using the Design/CPN simulator, simulations were performed to validate the CPN model and to conduct initial analysis. For validation, interactive (single-step) simulation was used to investigate if an execution of the model could reach the desired terminal state. A desired terminal state is characterised as follows. All 77 units of information are produced (77 tokens on *Output Information*/*Input Information* place), the 7 units of external information are still available at the end of the process (7 tokens on the *External Information* place), all staff are returned (ordered staff list was the same as the initial marking on the *Staff* place), all activities are completed (41 tokens on the *Completed Activities* place), and a plan was produced (a token with colour *Plan* on the *Planning Completed* place).

After the model was validated, behaviours of the process were investigated through the use of automatic simulation and simulation reports, where all the steps that occurred during a simulation were recorded. The simulation report can be used to produce a GANTT chart. Fig.7 depicts a GANTT chart created from the simulation report where the execution terminated in a desired terminal state. Activities in the JMAP are shown on the y-axis, and time (in minutes) is on the x-axis. The process took 2095 minutes to complete. It can be noted that the process is very sequential in nature, but some activities have occurred simultaneously and out of the order of the JMAP steps. For example, some Preliminary Scoping activities (*PS1_1*, *PS1_2* and *PS2*) occurred simultaneously, and some other Preliminary Scoping activities *PS4* and *PS5* occurred after certain Mission Analysis activities (labels starting with MA). These kinds of properties make the process more flexible and therefore a plan could be produced quicker than a strictly sequential process that would take 2151 minutes. The CPN model allows activities to occur "out of sequence" according to the activity attributes.

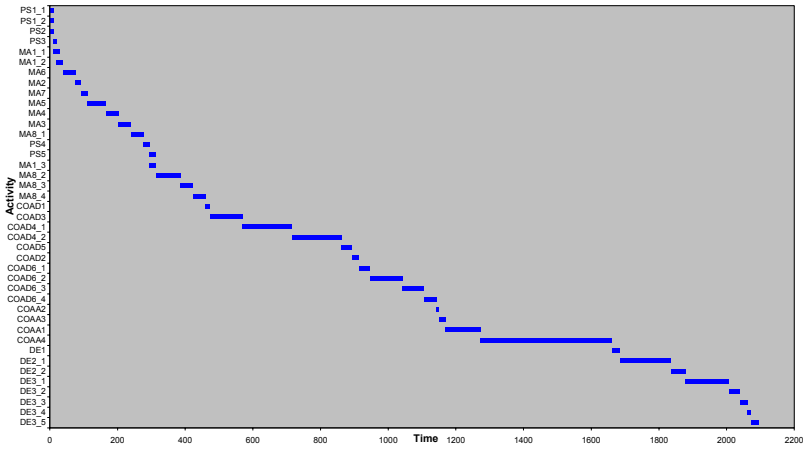


Figure 7: GANTT chart extracted from an automatic simulation.

We also investigated the completion time of the process when there were no resource requirements, i.e., the process was based solely on information flow with staff requirements ignored. A simulation report of this model was produced, and the corresponding GANTT chart is shown in Figure 8. From the GANTT chart, we found that the process took 1845 minutes to complete when there are no resource constraints. A larger number of activities were shown to have occurred concurrently. For example, certain *Mission Analysis* activities (*MA3*, *MA4*, *MA5*, and *MA6*) occurred in parallel. The simulation results suggest that one method of improving the process efficiency is to enable concurrent activities through de-conflicting staff requirements on activities.

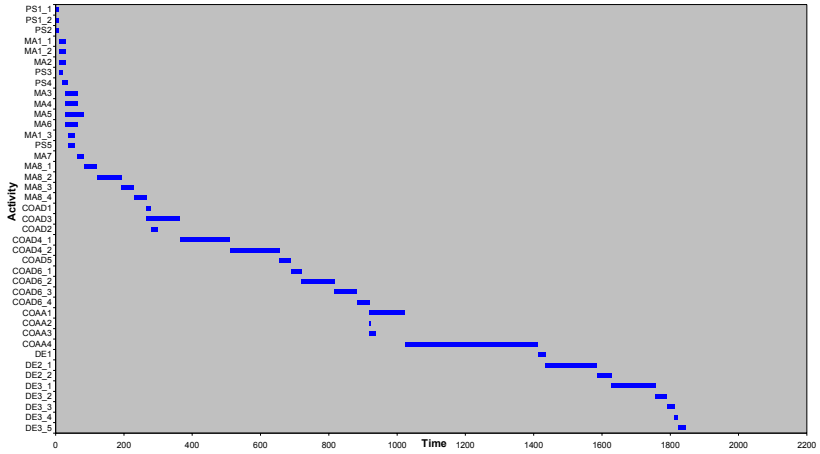


Figure 8: GANTT chart of a process with no resource constraints.

5 Full State Space Analysis

The interactive and automatic simulation reported in the previous section served as a first step to validate the CPN model and analyse the planning process. To obtain a rigorous analysis of the planning process and the CPN model, state space analysis was applied. The full state space of the CPN model has 14783 nodes, 21690 arcs, and could be generated in 2 minutes and 14 seconds on a PIII Linux PC.

The first part of the state space analysis was based on the state space report that can be produced fully automatically by the Design/CPN state space tool. The state space report contains answers to a number of standard dynamic properties of Petri nets such as *boundedness* properties, *home* and *liveness* properties, and *fairness* properties. In the following we interpret selected results from the state space report in the context of the DJFHQ planning process.

Boundness properties. The integer bounds specify the minimal and maximal number of tokens that can reside on a given place. The multi-set bounds give information about the minimal and maximal numbers of tokens with a certain colour that reside on the place in any reachable state. The state space report specifies the integer and multi-set bounds for each place in the CPN model. Table 1 specifies the lower and upper integer bounds for five places from the JMAP page, previously shown in Fig.4.

Place	Upper Bound	Lower Bound
External Information	7	7
Completed Activities	41	0
Input Information	77	0
Output Information	77	0
Planning Completed	1	0

Table 1: Selected upper and lower integer bound of places.

Both lower and upper integer bounds of the place *External Information* are 7, showing that there are always 7 tokens present on this place. Careful inspection of the upper and lower multi-set bounds (not shown) shows that the multi-set of tokens present on the *External Information* is always equal to the external information initially present when the planning process commences. This shows that the external information is not consumed by any activities, but only read. The upper integer bound of 41 for place *Completed Activities* shows that at most 41 activities can be completed. The 41 tokens correspond to the total number of activities present in the CPN model. Similarly, the upper integer bound on *Input Information* and *Output information* corresponds to the 77 information units that can maximally be produced in the planning process. The upper integer bound of 1 on the *Planning Completed* place shows that there exist states in which a plan has been produced. This confirms the observation made during the interactive and automatic simulations of the CPN model.

Liveness Properties. The CPN model has 14 reachable dead states (states without enabled transitions). These states correspond to states in which the planning process has terminated. To investigate whether these states represent desired terminal states of the planning process, a predicate on states was written expressing that a terminal state is a desired terminal state if the requirements stated in the beginning of Section 4 are all satisfied. Applying the predicate shows that all dead states represent desired terminal states of the planning process. This shows that if the planning process terminates, then it terminates in the desired state. Inspection of the dead states shows that the planning process may take 2141 minutes in worst case, and 2059 minutes in the best case. A path corresponding to an optimal schedule for the planning process can easily be obtained as a path in the state space from the initial state to a state where the planning process has terminated at time 2059.

Home Properties. A home space [11] is a set of states H with the property that from any reachable state, it is always possible to reach at least one of the states in H. Using the query function *HomeSpace* available in the Design/CPN state space tool, it was shown that the set of states constitute a home space. This means that the planning process has the property that it is always possible to terminate the process in a state where the plan has been produced. Generation of the strongly connected components graph showed that the state space is acyclic. Since the state space is also finite, this implies that when started, the process will eventually terminate in a state in which a plan has been produced. This establishes the soundness of the planning process.

Completion times. Another measure of interest in the analysis of the planning process is the earliest and latest time each activity can be completed. This information can also be obtained from the state space. Table 2 lists these results for the activities in the mission analysis step of the planning process. These results were obtained by traversing the state space using the functions available in the Design/CPN state space tool for writing non-standard queries. Similar results were obtained for the activities in the other steps of the process, and similar results can be obtained for the best and worst case start times of the activities.

Activity	Min	Max	Activity	Min	Max
MA2	28	157	MA3	65	286
MA4	65	286	MA5	83	286
MA6	65	268	MA7	85	286
MA11	28	28	MA12	28	286
MA13	56	414	MA81	277	323
MA82	350	432	MA83	387	469
MA84	424	506			

Table 2: Earliest and latest completion time for mission analysis activities.

6 Sweep-Line State Space Analysis

Full state space analysis of the DJFHQ CPN model was feasible with the available computing resources because the state space of the CPN model was of a moderate size. Since we eventually want to extend our work to cover even more complex and detailed business processes of the ADF, we are likely to encounter the state explosion problem, i.e., state space analysis will be prohibited because of the size of the state space. As part of the project we therefore experimented with the use of the sweep-line state space analysis method [3].

The basic idea behind the sweep-line method is to exploit a formal notion of progress present in many concurrent and distributed systems. Exploiting progress makes it possible to reclaim memory during state space exploration by deleting visited states on-the-fly. The deletion is done such that the state space exploration will eventually terminate and upon termination all reachable states will have been explored exactly once. Below we explain the basic ideas behind the sweep-line method and show how the method can be applied in on-the-fly state space analysis of the DJFHQ CPN model. The reader is referred to [3] for a complete presentation of the sweep-line method. For the experiments, we used the sweep-line library [7] available for Design/CPN.

The sweep-line method has until now only been used on communication protocols [8], exploiting progress originating from internal states of protocol entities, retransmission counters, and packet sequence numbers. There is however an intuitive presence of progress in many business processes from the start of the process toward the termination of the process when the desired outcome has been produced. The progress can, e.g., be measured in the number of completed activities, the number of documents produced, and the elapse of time. This kind of progress is also present in the DJFHQ planning process, and it is reflected in the state space of the CPN model. Figure 9 shows the initial fragment of the state space for the DJFHQ CPN model. The initial state is represented by node 1, and initially three different activities may start. The states have been organised into layers (separated by a horizontal line) based on how far the system has *progressed* according to the *creation time* of the marking (nodes). The creation time of a state in a timed CP-net represents the time at which

the system entered the corresponding state. For example, layer 0 contains the nodes representing states with creation time 0. The marking in layer 1 has creation time 10.

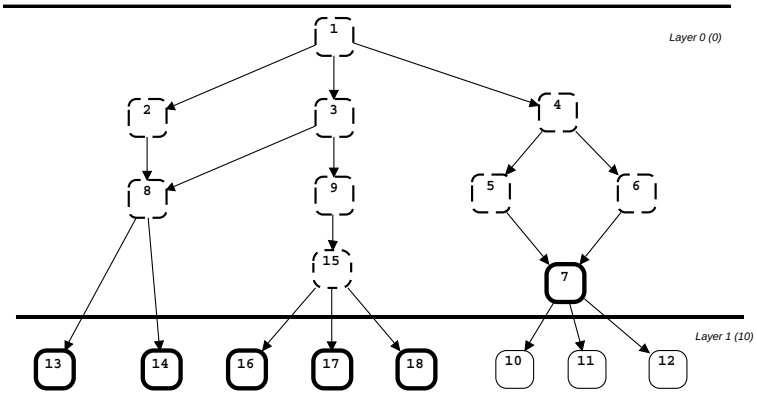


Figure 9: Initial fragment of the state space.

The key observation to make is that progress in the DJFHQ CPN model manifests itself by the property that a state in a given layer has successor states either in the same layer or in some lower layer, but never in an upper layer. This is a consequence of the fact that the creation time in a timed CP-nets increases along an occurrence sequence [11]. The idea underlying the sweep-line method is to exploit such progress by deleting states on-the-fly during state space exploration.

To illustrate how the sweep-line method operates, consider Fig.9 and assume that it represents a snapshot taken during conventional state space exploration. Dashed nodes are *fully processed* states (i.e. states that are stored in memory and all their successor states have been calculated). Nodes with a thick solid black border are *unprocessed* nodes (i.e. nodes that are stored in memory, but their successor states have not yet been calculated). Nodes with a thin solid black border have not yet been calculated.

If the state space exploration algorithm processes states according to their creation time, node 7 will be the state among the unprocessed states that will be selected for processing next. This will add nodes 10, 11, and 12 to the set of stored states and mark these as unprocessed. At this point it can be observed that it is not possible from any of the unprocessed states to reach one of the markings 1-9 or 15. The reason is these nodes represent states where the planning process has not progressed as far as in any of the unprocessed states. Hence, it is safe to delete these nodes, as they cannot possibly be needed for comparison with newly generated states when checking (during the state space exploration) whether a state has already been visited. In a similar way, once all the states in the second layer have been fully processed these nodes can be deleted from the set of nodes stored in memory. Intuitively, one can think of a *sweep-line* as being aligned with the highest layer (seen from the top) that contains unprocessed states. During state space exploration, unprocessed states are selected for processing in a least-progress first order causing the sweep-line to move downwards. States will thereby be added in front of the sweep-line and deleted behind the sweep-

line. We could have subdivided each layer further by taking into account, e.g., the number of started and completed activities or the number of produced documents.

To use the sweep-line method as implemented in the library [7], a progress measure must be provided to the tool. The progress measure specifies the progress to be exploited by the sweep-line method, and consists of a mapping from states into progress values. The progress value of a state quantifies the progress of the system in that state. The progress mapping is required to preserve the reachability relation of the CPN model, i.e., a successor state S' of a state S is required to have the same or a higher progress value than S. For the CPN model of the DJFHQ planning process, we used a function that maps a state into its creation time. As an inherited property of timed CP-nets, this mapping preserves the reachability relation.

The peak number of states stored with the sweep-line method using a progress measure based on creation time is 2149 nodes. Assuming that memory consumption is linear in the number of states stored, this corresponds to a reduction in peak memory consumption for analysis to 11.8 %. The total time used to conduct the sweep of the state space was 2 minutes and 33 seconds (compared to 2 minutes and 14 seconds for full state space generation). Using the sweep-line method, we can investigate the same dynamic properties as was considered in the previous section using the query functions available in the sweep-line library. The main difference is that analysis is now done on-the-fly during the state space exploration. This is necessary since state information is deleted by the sweep-line method. The sweep-line method can be used to reason about home and liveness properties because states in a strongly connected component of the state space will have the same progress value and hence will be present in memory simultaneously before being deleted.

7 Conclusions and Future Work

We have presented the development of a CPN model of the DJFHQ planning process. The model gives a formal and graphical representation of the process. It captures activities in the process, and how staff and information flow between these activities. An important feature of the CPN model is the uniform modelling of activities which eased the development of the CPN model based on the JMAP and SOP documents. The CPN model is useful for training new staff officers, assisting the HQ in modifying existing planning documentation based on the JMAP, and providing a framework to test variations of the JMAP and other business processes.

Another contribution of this paper is the analysis of the DJFHQ planning process using simulation and state spaces. The simulation results allowed recommendations to be given to the DJFHQ to facilitate concurrent activities in the process, and hence an earlier completed plan. The state space analysis allowed the soundness of the planning process to be established together with additional quantitative properties. To alleviate the state explosion problem, we have reported on initial experiments with the application of the sweep-line method in the workflow domain. These experimental results are very encouraging for the use of the sweep-line method in this domain, where models typically have an inherited presence of progress that can be exploited.

The planned direction of this work is to extend the CPN model to represent the external JMAP processes and other related processes at DJFHQ that interact with the JMAP. Also, it would be of interest to refine the CPN model by replacing the deterministic duration of activities with time intervals. The work on Interval Timed Coloured Petri nets and their state space analysis [2, 20] could serve as a starting point for this work. Recently, the JMAP process as presented in this paper has been modeled and analysed usingn stochastic Petri nets [6]. Finally, we are planning activities where the CPN model is applied at DJFHQ for training staff, and as a tool for monitoring the process during a planning exercise. In such a setting, the progress of the planning process can be monitored at the level of the CPN model, and state space analysis using the current state as initial state can be used to make predications, e.g., about worst and best case termination time given the current state of the planning process.

References

1. Australian Defence Force Publications (ADFP). *Joint Military Appreciation Process*. Operations Series 9, Joint Planning, Chapter 8, 1999.
2. G. Bertholot. Occurrence Graphs for Interval Timed Coloured Petri Nets. In Proc. Of ICATPN'94, volume 815 of Lecture Notes in Computer Science, pp. 79-98. Springer-Verlag, 1994.
3. S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In Proceedings of TACAS'2001, volume 2031 of Lecture Notes in Computer Science, pp. 450-464. Springer Verlag, 2001.
4. Deployable Joint Force Headquarters (DJFHQ). SOP 310 – The Operational Planning Process, 2001.
5. Design/CPN Online. http://www.daimi.au.dk/designCPN/.
6. J. Freiheit and Jonathan Billington. Using TimeNET to Evaluate Operational Planning Processes. In Proceedings of BPM 2004, volume 3080 of Lecture Notes in Computer Science, pp.17-32, Springer-Verlag, 2004.
7. G. E. Gallasch, L. M. Kristensen, and T. Mailund. The Sweep/CPN Library. Available via http://www.daimi.au.dk/designCPN/libs/sweepcpn/
8. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In Proceedings of Petri Nets 2002, volume 2360 of Lecture Notes in Computer Science, pp.182-202. Springer-Verlag, 2002.
9. R. Harper. *Introduction to Standard ML*. Technical Report ECS-LFCS-86-14, University of Edinburgh, Department of Computer Science, 1986.
10. K. Jensen. *Coloured Petri Nets: Volume 1: Basic Concepts* Monographs in Theoretical Computer Science, Spinger-Verlag, 1997.
11. K. Jensen. *Coloured Petri Nets: Volume 2: Analysis Methods* Monographs in Theoretical Computer Science, Spinger-Verlag, 1994.
12. K. Jensen. *Coloured Petri Nets: Volume 3: Practical Use.* Monographs in Theoretical Computer Science, Spinger-Verlag, 1997.
13. L. M. Kristensen, S. Christensen, and K. Jensen. *The Practitioner's Guide to Coloured Petri Nets*. International Journal on Software Tools for Technology Transfer, 2(2):98-132, 1998.

14. L. M. Kristensen, B. Mitchell, L. Zhang, and J. Billington. *Modelling and Initial Analysis of Operational Planning Processes using Coloured Petri nets*. In proceedings of Workshop on Formal Methods Applied to Defence Systems, volume 12 in Conferences in Research and Practice in Information Technology, pp. 105-114. Australian Computer Society, 2002.
15. S. Lumsden, R. Smallwood, B. Mitchell, and L. Zhang. *Modelling Operational Level Planning Processes with Coloured Petri Nets*. 7th International Command and Control Research and Technology Symposium. 2002.
16. T. Murata. Petri Nets: Properties, Analysis, and Application. In Proceedings of the IEEE, Vol. 77. No. 4, pp. 541-580. IEEE Computer Society, 1989.
17. A. Valmari. The State Explosion Problem. Lectures on Petri Nets I: Basic Models. Volume 1491 of Lecture Notes in Computer Science, pp. 429-528. Springer-Verlag, 1998.
18. W. van der Aalst and K. van Hee. *Workflow Management – Models, Methods and Systems*. The MIT Press, 2002.
19. W. van der Aalst. Advanced Tutorial on Workflow Management. 23rd International Conference on Application and Theory of Petri Nets, Adelaide, June 2002.
20. W. van der Aalst. Interval Timed Coloured Petri Nets and Their Analysis. In Proc. Of ICATPN'93, volume 691 of Lecture Notes in Computer Science, pp. 453-472. Springer-Verlag, 1993.

Experimenting with Progress Mappings for the Sweep-Line Analysis of the Internet Open Trading Protocol*

Guy Edward Gallasch[1], Chun Ouyang[1], Jonathan Billington[1], and Lars Michael Kristensen[2]**

[1] Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
Email: guy.gallasch@postgrads.unisa.edu.au, chun.ouyang@unisa.edu.au, jonathan.billington@unisa.edu.au
[2] Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, DENMARK
Email: kris@daimi.au.dk

Abstract. The sweep-line occurrence graph method exploits a behavioural notion of progress found in many systems. This allows states to be deleted that will not be revisited during occurrence graph generation, allowing fewer states to be stored in main memory for the necessary comparisons, thus providing savings in both memory and time. Properties of the system (such as deadlocks) can then be verified on-the-fly. This method is relatively new and needs to be evaluated on a range of examples. One class of protocols that seems to be suited to sweep-line analysis is transaction protocols. This is because transaction protocols often have an occurrence graph that starts with a request and finishes with the request being satisfied (or not). Thus there is a natural progression of states as the transaction proceeds. This paper provides insight into how to design a progress mapping, central to the use of the sweep-line method, for a transaction protocol known as the Internet Open Trading Protocol (IOTP). IOTP is quite complex and is modelled using hierarchical Coloured Petri Nets (CPNs). The sweep-line method is particularised for CPNs and three progress mappings are developed for IOTP. The results show that naive choices for the progress mapping leads to unnecessary regeneration of states, due to the mapping not being monotonic. Refinement of the mapping leads to a monotonic progress measure, which allows results to be obtained for IOTP that were not previously possible.

Keywords: State space methods, Occurrence graph methods, Sweep-line, State explosion problem, Internet Open Trading Protocol, Coloured Petri Nets, Verification.

1 Introduction

State space (occurrence graph) methods encompass the paradigm of analysis techniques that involve generation of all or part of the reachable state space (occurrence graph) of a system in order to answer verification questions. This paradigm is one of the main analysis methods for Coloured Petri nets (CPNs) [19,20] and has been used successfully to analyse and verify many systems (for examples see [1, 21]). These methods have an advantage over theorem proving techniques [35] in that the mathematics can be neatly contained in automated software tools such as Design/CPN [5, 8].

One disadvantage that has been the subject of much research is that of the *state explosion problem*. Even for relatively simple systems, the number of reachable states can be very large. The unfortunate result of state explosion is that in many cases, the entire occurrence graph is too large to fit into computer memory. This has led to a number of so-called *state space reduction* techniques to alleviate the problem.

A good survey of reduction techniques is provided in [35]. These techniques may be classified into three main categories. The first are those that represent the occurrence graph

* Supported by an Australian Research Council (ARC) Discovery Grant (DP0210524).
** Supported by the Danish Natural Science Research Council

in a condensed or compact form, such as symmetry reduction [7, 9, 18]. The second class explores only a subset of the reachable states. Partial order methods [31, 34, 38] such as stubborn sets fall into this category. The third class involves deleting or throwing away states or state information during exploration and include *bit-state hashing* [15, 16, 39], *state space caching* [11, 13, 14] and the *pseudo-root* technique [30].

The sweep-line exploration method belongs to the third category. It guarantees full coverage of the occurrence graph but differs from the state space caching and pseudo-root techniques in the way that states are selected for deletion. By exploiting *progress* in the model being analysed, a *progress mapping* can be defined which identifies states that are guaranteed not to be reached again [6] or are unlikely to be reached again [24].

The Internet Open Trading Protocol (IOTP) [3, 4] is an electronic commerce protocol developed by the Internet Engineering Task Force (IETF). The core of IOTP is a set of electronic transactions that reflect common trading activities, such as purchasing goods or depositing funds, over the Internet. The specification of IOTP, published as Request For Comments (RFC) 2801 [3], was the largest RFC developed by IETF to that time, spanning 290 pages. The RFC however contains an informal narrative description of IOTP, and so far no complete implementation of IOTP yet exists [17, 32].

A hierarchical CPN model of IOTP was created [29] and further improved [27] to cover most protocol features in RFC 2801. A set of desired properties of IOTP (e.g., correct termination) were investigated in [28] and revealed errors in the design of IOTP. Changes to RFC 2801 were suggested and a revised IOTP CPN model [26] developed. This paper has arisen from attempts to analyse the revised IOTP CPN, which presents a practical challenge due to the large number of reachable states for increasing parameter values.

The purpose of this paper is to apply the sweep-line method to the revised IOTP CPN in order to obtain results for larger parameter values than is possible using conventional analysis [26] and to provide more experience in applying the sweep-line method to practical examples. Another objective is to provide a comparison between the effectiveness of sweep-line analysis when using a progress mapping based on general protocol properties and one based on IOTP-specific properties when analysing the revised IOTP CPN.

The rest of this paper is organised as follows. Section 2 provides a description of the sweep-line method. Sections 3 and 4 introduce the Internet Open Trading Protocol and its CPN model, respectively. The derivation of three different progress mappings and some insights into the process for doing this are presented in Section 5 and the experimental results obtained by using them are presented in Section 6. Finally, some concluding remarks and future work are presented in Section 7. We assume that the reader is familiar with the basic concepts of CPNs and reachability analysis.

2 The Sweep-line Method

We present the sweep-line method in the context of Coloured Petri nets [19, 22] as we are using the sweep-line method to analyse a CPN model. The method is, however, not specific to CPNs, but applicable to a wide range of modelling languages and formalisms.

The sweep-line method is based on the notion of *progress* within the system being modelled. Systems exhibit progress in different ways. One example is found in transaction protocols such as IOTP, where interacting protocol entities move through a series of interactions (called *exchanges* in IOTP) towards a final completed state. IOTP is described in more detail in the next section. Communication protocols in general exhibit progress through sequence numbers and retransmission counters. The key concept behind the sweep-line method is that if we can

quantify the progress of a system in each state, then we can identify the states with a lower *progress value* that cannot be reached from states with a higher progress value. When states are no longer reachable we do not need to keep them in memory for comparison with each newly generated state.

The notion of progress is captured formally in a *progress measure* [6, 24]. Importantly a progress measure specifies a *progress mapping* ψ from states to progress values that are ordered. In this paper we shall use the natural numbers $\mathbb{N}$ as the set of progress values and their usual order relations (e.g. $\leq, <, >$). We firstly introduce the concept of a CPN instrumented with a progress mapping ψ.

Definition 1. *A CP-net with a progress mapping is a tuple $CPN_\psi = (CPN, \psi)$ where CPN is a Coloured Petri net (defined in [19]) and ψ is a progress mapping given by $\psi : \mathbb{M} \to \mathbb{N}$ where $\mathbb{M}$ is the set of possible markings for CPN.*

From [19], let $[M_0\rangle$ be the set of *reachable* markings of CPN and $be \in BE$ be a binding element enabled in marking M. If, for a given progress mapping $\psi, \forall M, M' \in [M_0\rangle, M[be\rangle M' \Rightarrow \psi(M) \leq \psi(M')$ then the mapping ψ is *monotonic* with respect to the reachability relation and implies that if $\psi(M) > \psi(M')$ for $M, M' \in [M_0\rangle$ then $M' \notin [M\rangle$.

We may consider that the mapping ψ induces an ordered partition on the set of reachable markings. Once all successors of all markings with a particular (minimum) progress value have been generated, then, for a monotonic progress mapping, we can delete the markings (that are not of interest) with this progress value, freeing up memory, and reducing the time spent comparing new markings with those already generated. The overhead is calculating the progress value for each state, and ensuring that markings are processed in a least-progress-first order.

The monotonicity of the progress mapping can be checked during occurrence graph (OG) generation as all arcs in the OG are traversed by the sweep-line method. If, however, $\psi(M') < \psi(M)$ for some $M[be\rangle M'$ then we have a *regress edge*:

Definition 2. *Let OG_{CPN} be the occurrence graph of CPN in CPN_ψ as given in Definition 1. An occurrence of binding element $be \in BE$ of CPN in marking $M \in [M_0\rangle$, leading to marking M' in which $\psi(M') < \psi(M)$ is called a* **regress edge** *with respect to ψ of OG_{CPN}.*

Regress edges may lead to new markings or to markings that have already been explored but subsequently deleted from memory. The sweep-line algorithm has no way of distinguishing between these two types of markings and so must treat all destinations of regress edges as if they have not yet been explored. Exploration does not continue along regress edges, however the destinations of regress edges are marked as roots (initial states) for a subsequent sweep of the OG. To guarantee termination of the algorithm these states are also marked as *persistent*. Persistent states cannot be deleted and so each state can be marked at most once as a root state for a subsequent sweep.

An algorithm for the sweep-line method that takes into account non-monotonic progress mappings was presented in [24]. We present a modified version in Fig. 1 for CPN_ψ. Let there be a set ROOTS which contains the starting markings of a sweep; a set PERSISTENT which contains markings that cannot be deleted; a set UNEXPLORED which contains all the markings generated so far within a sweep that have not had their successors explored; a set PROCESSED which stores markings which have had their successors generated in a sweep; a set SUCCESSORS which holds the successors of a given marking; and a set DM which stores all dead markings. The algorithm selects and removes a marking from UNEXPLORED that has the minimum progress value among all states in UNEXPLORED (lines 12 and 13), adds it to

```
1: Roots ← {M0}
2: Persistent ← ∅
3: Unexplored ← ∅
4: Processed ← ∅
5: Successors ← ∅
6: DM ← ∅
7: while Roots ≠ ∅ do
8:    Unexplored ← Roots
9:    Roots ← ∅
10:   while Unexplored ≠ ∅ do
11:      (* Generate the successors of a node in Unexplored that has the lowest progress value *)
12:      Select M ∈ Unexplored such that ∀M' ∈ Unexplored, ψ(M) ≤ ψ(M')
13:      Unexplored ← Unexplored \ {M}
14:      Processed ← Processed ∪ {M}
15:      Successors ← {M'|M[be⟩M'}
16:      if Successors = ∅ then
17:         DM ← DM ∪ {M}
18:      else
19:         Roots ← Roots ∪ {M' ∈ Successors | ψ(M') < ψ(M) and M' ∉ Persistent}
20:         Persistent ← Persistent ∪ {M' ∈ Successors | ψ(M') < ψ(M)}
21:         Unexplored ← Unexplored ∪ {M' ∈ Successors | ψ(M') ≥ ψ(M) and M' ∈ Processed}
22:      end if
23:      (* Delete states that have a progress value less than those in Unexplored *)
24:      Processed ← Processed \ {s ∈ Processed|∀M' ∈ Unexplored, ψ(s) < ψ(M')}
25:   end while
26: end while
```

Fig. 1. The Generalised Sweep-line Algorithm, based on the algorithm from [24].

the set of processed markings (line 14) and generates all successors of this marking (line 15). If there is no successor, the marking is added to the set of dead markings (line 17). If any regress edges are detected, their destination markings (if not already marked as persistent) are added to Roots as initial states for the next sweep (line 19) and marked as persistent (line 20). Destinations of non-regress edges that have not already been processed are added to Unexplored (line 21). Deletion of states occurs on line 24.

The example shown in Fig. 2 (from [25]) illustrates the behaviour of the sweep-line method. This figure shows three snapshots of the OG during OG exploration. Arc labels have been omitted to simplify the diagram. The states are arranged from left to right in ascending progress order. Nodes that have been explored and deleted are represented as empty circles. Nodes currently in memory (but that have not yet been explored) are solid black circles. Nodes yet to be discovered are grey circles. In Fig. 2 (a) the states M_0 and M_1 have been explored and

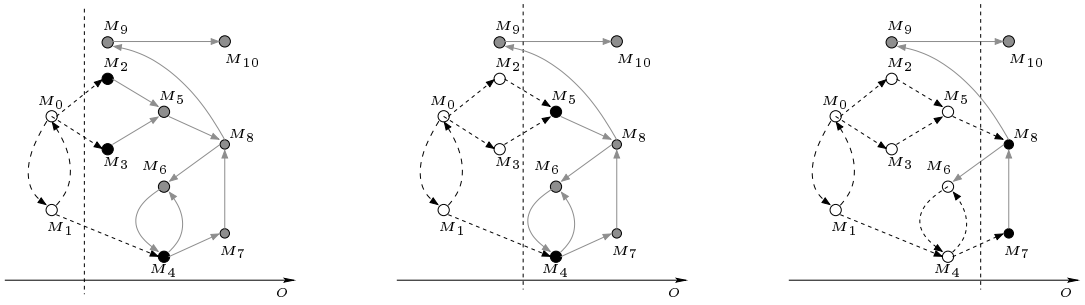


Fig. 2. Snapshots of sweep-line occurrence graph exploration.

subsequently deleted because they have a smaller progress value than the minimal progress value among the unprocessed states M_2, M_3 and M_4. The conceptual sweep-line is shown as a vertical dashed line, immediately to the left of the unprocessed states.

Exploring in least-progress-first order means that either M_2 or M_3 will be explored next. When both have been explored, the sweep-line moves to the right and M_2 and M_3 are deleted, giving the situation shown in Fig. 2 (b). M_4, M_5 and M_6 will be explored and eventually the situation shown in Fig. 2 (c) will be obtained. When M_8 is explored two regress edges are identified, one going to the previously explored state M_6 and the other to the unexplored state M_9. Note that the algorithm does not know that M_6 was previously explored, as it was deleted. The algorithm marks both M_6 and M_9 as persistent and flags them as roots for a subsequent sweep. In the subsequent sweep, M_{10} is discovered, along with the re-exploration of M_4, M_7 and M_8. Because M_6 and M_9 are persistent, the regress edges to M_6 and M_9 discovered in the re-exploration of M_8 do not induce a further sweep. The correctness of the sweep-line algorithm (both termination and full OG coverage) was proved in [24].

One drawback of the sweep-line method is that users need to define and supply their own progress mapping. Steps have been taken towards automatic generation of ψ for low-level Petri nets [33] and compositional systems [23].

3 The Internet Open Trading Protocol (IOTP)

IOTP [3] focuses on consumer-to-business e-commerce applications. It defines five *trading roles* to identify the different roles that organisations can assume while trading. These are *Consumer*, *Merchant*, *Payment Handler* (a bank), *Delivery Handler* (a courier firm) and *Merchant Customer Care Provider*. The core of IOTP is an *Authentication* transaction and five payment-related transactions named *Purchase*, *Deposit*, *Withdrawal*, *Refund* and *Value Exchange*. Each transaction comprises a sequence of IOTP message exchanges between trading roles, where each IOTP message comprises a set of pre-defined trading blocks. IOTP [3] currently uses HTTP [10] as its transport mechanism.

3.1 Document Exchanges and Transactions

IOTP defines a set of document exchanges as building blocks for creating transactions. These are: *Authentication*, *Brand Dependent Offer*, *Brand Independent Offer*, *Payment*, *Delivery*, and *Payment-and-Delivery*. An Authentication transaction consists of just an Authentication (document) exchange. A Purchase transaction comprises an optional Authentication, an Offer (either a Brand Dependent Offer or a Brand Independent Offer), and then, a Payment exchange, a Payment followed by a Delivery exchange, or a Payment-and-Delivery exchange. A Deposit, Withdrawal, or Refund transaction starts with an optional Authentication, an Offer, and a Payment exchange. Finally, a Value Exchange transaction begins with an optional Authentication followed by an Offer and two Payment exchanges in sequence.

Below, we consider an example of a Purchase transaction comprised of an Authentication, a Brand Dependent Offer, a Payment and a Delivery exchange. Figure 3 shows a possible sequence of messages exchanged between the four trading roles involved in the transaction.

In the beginning the Consumer decides to buy goods and so sends a Purchase Request (event 1) to the Merchant. This event initiates a Purchase transaction, however it is not part of Baseline IOTP [3] and is handled by HTTP [10].

Upon receiving the Purchase Request, the Merchant starts an Authentication document exchange (events 2-4) to verify the *bona fides* of the Consumer. In IOTP's terminology, the

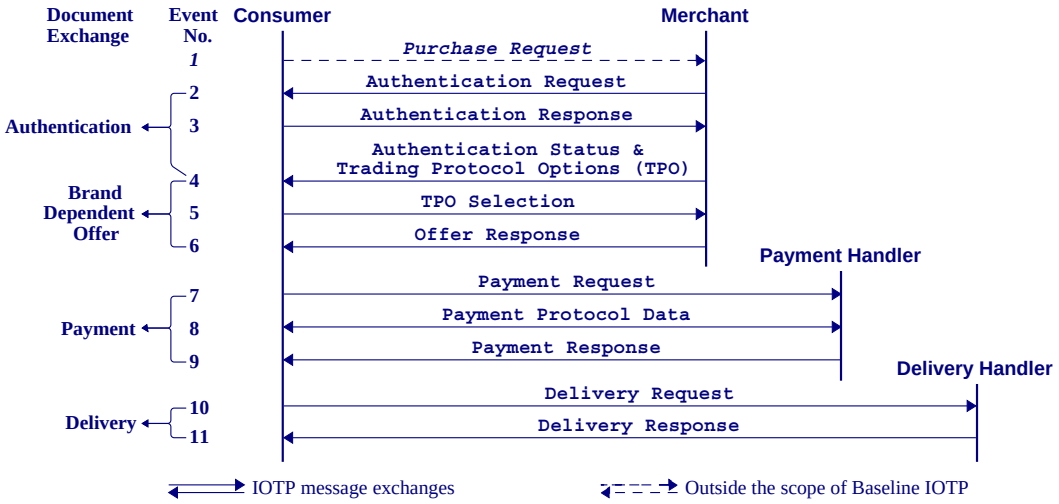


Fig. 3. A possible sequence of message exchanges in a Purchase transaction.

Merchant acts as the *Authenticator* and the Consumer the *Authenticatee*. At first, an Authentication Request is issued by the Merchant (event 2), specifying the authentication algorithm to be used. As a result, the Consumer replies with an Authentication Response containing the authentication data obtained using the above algorithm (event 3). After verifying the Consumer's response, the Merchant generates an Authentication Status indicating that the authentication is successful (part of event 4).

Once the authentication completes, the Merchant continues to a Brand Dependent Offer document exchange in our example (events 4-6) by providing the Consumer a list of Trading Protocol Options (TPO). This includes the available payment methods and associated payment protocols. The message combining the TPO and the above Authentication Status is then sent to the Consumer (event 4). The Consumer chooses one of the options, and sends it back as a TPO Selection (event 5). The Merchant uses the selection to create and send back an Offer Response (event 6), which contains details of the goods to be purchased together with payment and delivery instructions.

Next, a Payment document exchange starts between the Consumer and the Payment Handler (events 7-9). After checking the Offer Response for purchase details, the Consumer sends the Payment Handler a Payment Request (event 7). The Payment Handler checks the Payment Request, and if valid, the payment is conducted using Payment Protocol Data exchanges (event 8) as determined by the encapsulated payment protocol (e.g., Secure Electronic Transaction). After the payment protocol data exchange has finished, the Payment Handler sends a Payment Response (event 9) containing the payment result (e.g., receipt).

Finally, a Delivery document exchange is carried out between the Consumer and the Delivery Handler (events 10-11). After checking the Payment Response, the Consumer sends the Delivery Handler a Delivery Request (event 10). The Delivery Handler schedules the delivery and sends the Consumer a Delivery Response (event 11) containing details of the delivery, and possibly the actual delivery if the goods are electronic (e.g., an e-journal).

It should be mentioned that in a Brand Independent Offer the TPO Selection in event 5 does not occur. A Brand Dependent Offer occurs when the Merchant offers some additional benefit (e.g., price discount) in the Offer Response that depends on the specific *payment brand* (e.g., VISA or MasterCard) chosen in the Consumer's TPO Selection. In the Brand Independent Offer, the Offer Response is independent of the TPO and so the TPO Selection

(event 5) does not happen. Also, IOTP defines a combined TPO and Offer Response message (combining events 4 and 6) for a Brand Independent Offer.

3.2 Transaction Cancellation and Error Handling

A Cancel message is used for transaction cancellation and an Error message for reporting errors and instigating retransmissions. A transaction may be cancelled by any trading role engaged in that transaction. For example, in the Purchase transaction shown in Fig. 3, the Merchant would cancel the transaction if the Consumer's Authentication Response failed. Error handling is concerned with how trading roles handle technical errors and exceptions that occur during a transaction. For example, in Fig. 3, the Merchant may re-send the TPO upon reception of an Error message when expecting the Consumer's TPO Selection. Also, IOTP defines a *message identifier* to uniquely identify IOTP messages at each local trading role. Only duplicates have the same message identifier.

4 A Revised IOTP CPN Model

RFC 2801 [3] contains an informal narrative description of IOTP and suffers from ambiguities and incompleteness. We have created a CPN model of IOTP for six Authentication and Payment-related transactions [29] and further improved it to include procedures for error handling and arbitrary cancellation [27]. Analysis of the IOTP CPN model [28] revealed two main errors in the current design of IOTP, where the Payment-related transaction fails to terminate correctly. This motivated the development of a revised IOTP specification to eliminate the identified errors. To see if the revised protocol was correct, we revised the previous IOTP CPN [27]. In this section, we describe the revised IOTP CPN model only to the level of detail necessary to understand the derivation of the progress mappings in Sect. 5. A presentation of the complete CPN model can be found in [26].

4.1 Net Structure

Figure 4 shows the *hierarchy page* for the revised IOTP CPN. There are 31 pages organised into four hierarchical levels, giving a logical structure that can be validated against RFC 2801.

The first (top) level has one page named IOTP_TopLevel. The second level comprises four pages: Consumer, Merchant, PHandler and DHandler, corresponding to four trading roles[1]. We refer to these pages as *trading role pages*. Each trading role page has a set of subpages specifying the possible Authentication and payment-related transactions for that trading role. All these subpages, which we call *transaction pages*, constitute the third level of the model. The initial letter of a trading role is used as a suffix of the name of transaction pages modelled for that trading role. For example, the page Consumer has four subpages modelling six transactions for the Consumer. The three transactions Deposit, Withdrawal and Refund use the same procedure and therefore are modelled on one page named Deposit_C/Withdrawal_C/Refund_C. Each transaction page is further decomposed into a set of subpages modelling the document exchanges that are used to construct the transaction as well as error handling and cancellation procedures. All these subpages, which we call *exchange level pages*, constitute the fourth level of the model. For example, the page Purchase_C has six subpages modelling the six document exchanges used to implement the Purchase transaction for the Consumer and two others for

[1] The Merchant Customer Care Provider, currently not used in any transaction, is not modelled.

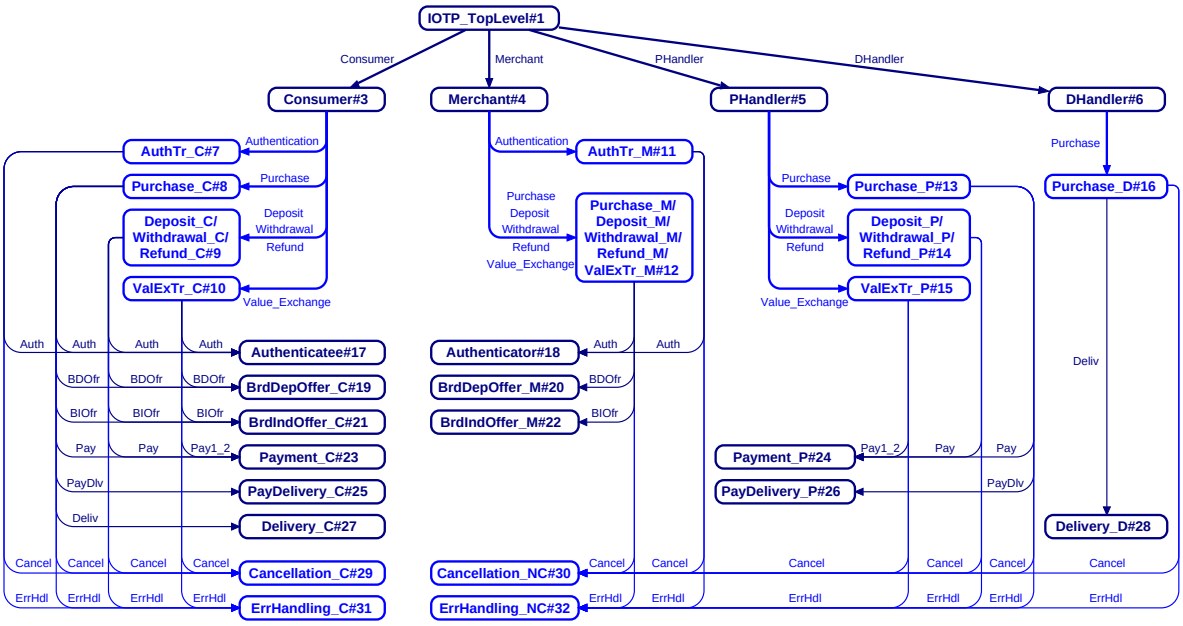


Fig. 4. The Hierarchy page.

error handling and cancellation. Since a document exchange involves two trading roles, we have modelled each exchange as a pair of pages - one for each of the trading roles involved in the document exchange. For example, the pages Authenticatee and Authenticator in Fig. 4 represent an Authentication exchange where the Consumer is authenticated by the Merchant.

4.2 Global Declarations

Figure 5 shows part of the global declarations that define the colour sets and variables for IOTP messages and trading role states. The declarations not shown in Fig. 5 define functions used in the model.

IOTP messages are modelled by the colour set IotpMsg (line 15) as a *list* of *trading blocks*, as derived from their XML definition in RFC 2801. The colour set TradingBlk (lines 8-14) is defined as the *union* of different kinds of trading blocks of IOTP messages. A trading block describes several attributes, some of which are specified by the first four colour sets (lines 1-5). The colour set TwoErrAttrsOfErrComp (line 6) is defined as a *product* of the two colour sets Severity (line 4) and ErrorCode (line 5), and models an *Error Block* containing these two attributes. The colour set MsgId (line 17), defined as *integers*, models the message identifier. The colour set Message (line 18), defined as the product of IotpMsg and MsgId, specifies that each IOTP message has a message identifier. Since IOTP currently operates over HTTP, we consider a reliable transport medium with no loss, duplication or re-ordering. Because of this, a list of Messages is defined by MsgQueue (line 20) for use when modelling a First-In-First-Out queue of messages between trading roles. The colour set TradingRole (line 21) enumerates the four trading roles. The colour set TRxTRxMQ (line 22) is defined as a product of a sender TradingRole, a receiver TradingRole and the MsgQueue between the two trading roles.

Trading role states are modelled by the colour set State (line 31), which is the product of the following five colour sets. InternalState (lines 25-26) specifies the eight internal states of a trading role in a document exchange. Exchange (line 27) represents the six document exchanges and a value NoExch indicating no document exchange occurs. R_Counter (line 29) models the message retransmission counter used in the error handling procedure, which increments its

```
   (* Selected Attributes *)
 1 color IotpTransType = with Authentication | Purchase | Deposit | Withdrawal | Refund |
 2                       ValueExchange;
 3 color DelivExch = with True | False;
 4 color Severity = with TransientError | HardError;
 5 color ErrorCode = with MsgErr | MsgBeingProc;
 6 color TwoErrAttrsOfErrComp = product Severity * ErrorCode;
 7 var trtype: IotpTransType;   var dlv: DelivExch;

   (* Trading Blocks and IOTP Messages *)
 8 color TradingBlk = union TransRefBlk:IotpTransType +
 9                          AuthReqBlk + AuthRespBlk + AuthStatusBlk +
10                          TpoBlk + TpoSelectionBlk + OfferRespBlk:DelivExch +
11                          PayReqBlk + PayExchBlk + PayRespBlk +
12                          DeliveryReqBlk + DeliveryRespBlk +
13                          ErrorBlk:TwoErrAttrsOfErrComp +
14                          CancelBlk;
15 color IotpMsg = list TradingBlk;
16 var m, rm, sm: IotpMsg;

   (* IOTP Message with Message Identifier *)
17 color MsgId = int;
18 color Message = product IotpMsg * MsgId;
19 var id, rid, sid: MsgId;

   (* Message Queue between Trading Roles *)
20 color MsgQueue = list Message;
21 color TradingRole = with Consumer | Merchant | PHandler | DHandler;
22 color TRxTRxMQ = product TradingRole * TradingRole * MsgQueue;
23 var role: TradingRole;
24 var q, rq, sq: MsgQueue;

   (* Trading Role State and Message Buffer *)
25 color InternalState = with READY | COMPLETED | CANCELLED |
26                          LISTEN | WAIT | HOLD | HOLD_WAIT | FINISHED;
27 color Exchange = with Auth | BDOfr | BIOfr | Pay | PayDlv | Deliv | NoExch;
28 val RCmax = 1;      (* Maximum number of message Re-transmissions *)
29 color R_Counter = int with 1..RCmax;      (* Message Re-transmission Counter *)
30 color MIdxRC = product MsgId * R_Counter;
31 color State = product InternalState * IotpTransType * Exchange * MIdxRC * MsgId;
32 color StaxBfr = product State * IotpMsg;
33 var s: InternalState;   var exch: Exchange;   var rc: R_Counter;
```

Fig. 5. Definitions of colour sets and variables for the revised IOTP CPN model.

value by one upon each message retransmission until it reaches the maximum value RCmax (line 28). MIdxRC (line 30) records the MsgId of the previously sent message along with its R_Counter. The last colour set in State represents the MsgId of the previously received message, used to check for duplicates in the error handling procedure. The colour set StaxBfr (line 32) combines the State of a trading role with the IotpMsg residing in the message retransmission buffer for that trading role.

4.3 The Top Level Page

Figure 6 shows the IOTP_TopLevel page that provides an abstract view of IOTP. The four substitution transitions, Consumer, Merchant, Payment Handler and Delivery Handler, represent IOTP's procedures for the corresponding trading roles. The place Transport, typed by colour set TRxTRxMQ, models the transport medium over which the trading roles communicate.

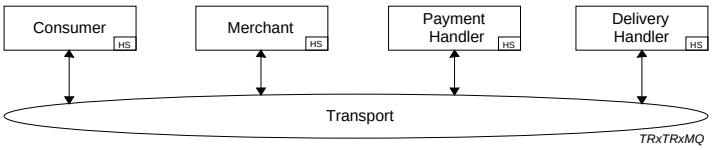


Fig. 6. The IOTP_TopLevel page.

4.4 Trading Role Pages

Both Consumer and Merchant are involved in all six transactions. Figure 7 depicts the two corresponding pages Consumer and Merchant, which we use as representative examples of the four trading role pages in the CPN model. Each transaction is abstractly represented by a substitution transition which has the same name as the transaction. Each place has a name starting with C or M, is typed by the product set StaxBfr, and models the state of the Consumer or Merchant with its message retransmission buffer in one of the six transactions. For brevity, we refer to these places as C_places or M_places. The exception is the Transport place on each page in Fig. 7 which has been described above.

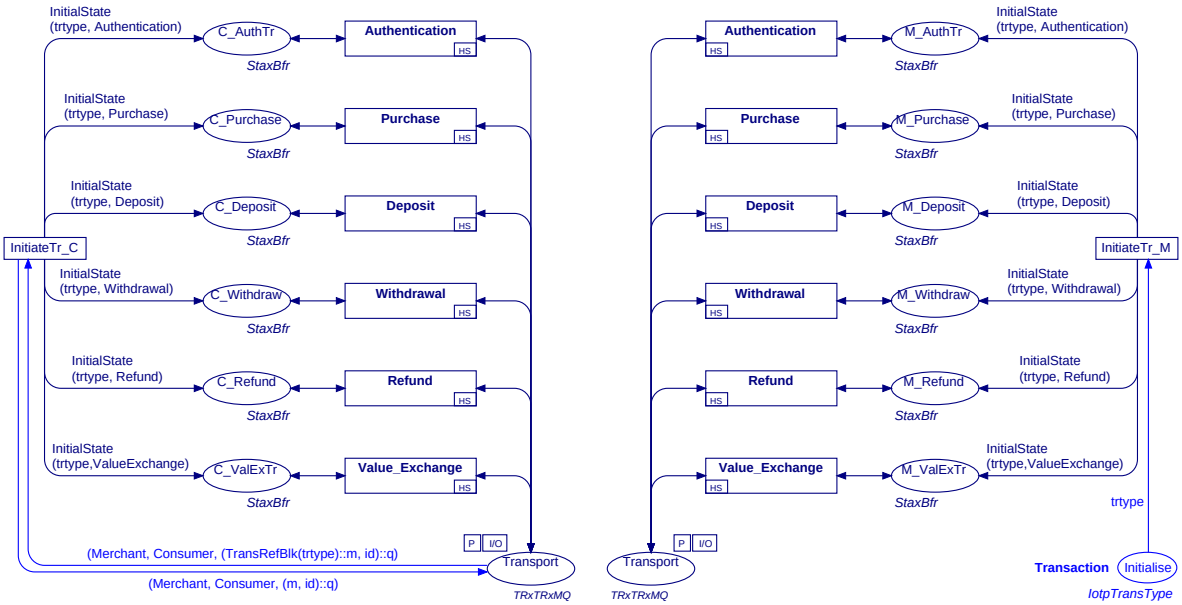


Fig. 7. Two trading role pages: (a) Consumer and (b) Merchant.

In Fig. 7 (b), the page Merchant has another place called Initialise (bottom right), which stores the transaction to be initiated by the Merchant. It is typed by the colour set IotpTransType and has an initial marking given by a constant Transaction that can be set to any of the six transactions. Place Initialise has an output arc to the only ordinary transition called InitiateTr_M. The arc is inscribed by variable trtype representing any token available in place Initialise. The arc from transition InitiateTr_M to each M_place is inscribed by a function named InitialState (see below), modelling how the initial state of the Merchant is determined upon a certain transaction type.

Figure 8 defines the function Initialise. It has two parameters: trtype and trtypevalue, both of type IotpTransType. If the two parameters have the same value, the function returns 1‘((READY,trtype,NoExch,(0,0),0),[]) (of StaxBfr), representing the initial state of a trading role. If not, no token value is returned (defined as empty). For example, in Fig. 7 (b), if

Transaction has the value Purchase, upon occurrence of InitiateTr_M, a token that indicates the Merchant is ready to carry out a Purchase transaction will be added to place M_Purchase.

```
1 fun InitialState(trtype, trtypevalue:IotpTransType): StaxBfr =
2   = if trtype=trtypevalue
3     then 1`((READY, trtype, NoExch, (0, 0), 0), [ ])
4     else empty
```

Fig. 8. Function used to determine the initial state of a trading role.

The page Consumer in Fig. 7 (a) has only one ordinary transition, InitiateTr_C, used to initiate an appropriate transaction for the Consumer on receiving the first IOTP message from the Merchant. It has both input and output arcs associated with place Transport. The input arc has an inscription specifying the first IOTP message (represented by (TransRefBlk(trtype)::m, id)) from the Merchant. The output arc indicates that processing of a *Transaction Reference Block*, which conveys transaction type information (TransRefBlk(trtype)) in an IOTP message, is complete, and thus the block is removed from the input message buffer. Similarly, the token returned by function InitialState is added to one of the six C_places on occurrence of transition InitiateTr_C, modelling that the corresponding transaction has been initiated at the Consumer.

5 Sweep-line Exploration of IOTP

Analysis of the revised IOTP focuses on the six Authentication and Payment-related transactions. For each transaction, there are also different cases we can analyse by changing the maximum value of the message retransmission counter ($RCmax$, line 28 of Fig. 5) for each of the four trading roles. Once the number of retransmissions reaches $RCmax$, the transaction will be cancelled. However, RFC 2801 does not define a value for $RCmax$. We therefore have modelled $RCmax$ as an unbounded parameter, resulting in an infinite number of possible configurations. In [26] the revised IOTP CPN was analysed for $0 \leq RCmax \leq 3$. When $RCmax$ was increased to 4, the OG of both the Purchase and Value Exchange transactions became too large to manage with the available computer resources at that time. It is our intention to alleviate the problems of state explosion for the revised IOTP with $RCmax$ greater than 3 by applying the sweep-line method.

Two approaches were taken to define a progress mapping for the application of the sweep-line method to the revised IOTP CPN. The first was to look only at features common to many protocols, such as sequence numbers and retransmission counters. The second was to develop a progress mapping based on specific features of IOTP and hence take advantage of behavioural properties of the IOTP CPN. These progress mappings are presented in Sections 5.1 and 5.2 respectively. A third progress mapping, combining the generic and specific progress measures, is presented in Section 5.3.

5.1 Generic Progress Mapping

Sequence numbers and retransmission counters are features common to many protocols. It would be nice if a generic progress mapping, giving reasonable performance, could be derived from these common features.

Each trading role in IOTP maintains its own retransmission counter and sequence number (the message identifer) for the message it most recently transmitted. The message identifier is used to differentiate messages when interacting with other trading roles and retransmissions

are used to recover from (processing delay and transient) errors. As an example, let us consider the Consumer trading role. In a Purchase transaction, the Consumer trading role keeps this information in the token on place C_Purchase on the Consumer page. This token belongs to the StaxBfr colour set (see Fig. 5). To obtain the message identifer and retransmission counter of the Consumer we can define functions $\texttt{GetMessID}_{Consumer}$ and $\texttt{GetRC}_{Consumer}$ to extract them from the token on C_Purchase in a given marking $M \in [M_0\rangle$ of the IOTP CPN. Analogous functions can be defined for the Merchant, Payment Handler and Delivery Handler.

Let us define $TR = \{\text{Consumer, Merchant, Payment Handler, Delivery Handler}\}$ to be the set of four trading roles. One strategy to create a *generic* progress mapping would be a straight sum of the values of these variables over all trading roles, i.e. providing the mapping

$$\psi_{generic\_1}(M) = \sum_{tr \in TR} (\texttt{GetMessID}_{tr}(M) + \texttt{GetRC}_{tr}(M))$$

Intuitively, this may not be the best solution, as many combinations of message identifiers and retransmission counters will result in the same progress value. It is desirable to have many progress values to give the potential for greater reduction, but there is a trade-off with the number of regress edges introduced (we want this to be low) and also the number of states discovered with higher progress values that are not immediately explored but are nonetheless stored in memory and cannot be deleted until later. We also note that, in general, a sequence number represents a *more significant* measure of progress than a retransmission counter, as generally a retransmission counter is kept for each message, then reset for the next message. We use this knowlegde to construct a more elaborate progress mapping by giving a weight to the message identifier component of the progress values. This weight is $(RCmax + 1)$ and is one larger than the maximum possible value of $\texttt{GetRC}_{tr}$. This ensures that an increment in the message identifier is always more significant than any increment in the retransmission counter. A second generic progress mapping is then defined as:

$$\psi_{generic\_2}(M) = \sum_{tr \in TR} ((RCmax + 1) * \texttt{GetMessID}_{tr}(M) + \texttt{GetRC}_{tr}(M))$$

This is the generic progress mapping we use in our experiments in Section 6.1. Note that $\psi_{generic\_2}$ may not exhibit the optimum combination of message identifiers and retransmission counters with respect to the IOTP CPN. For example, each trading role could be given its own weight to further differentiate the progress of states, but assigning anything but an arbitrary weighting to each trading role may require IOTP-specific knowledge, hence violating our intentions for the general nature of this progress mapping.

An important feature of this progress mapping is its incorporation of the parameter $RCmax$. As $RCmax$ increases, the set of generated progress values also increases. It is hoped that this progress mapping will *scale* well with $RCmax$. Scalability is a desirable property of a progress mapping.

5.2 IOTP-Specific Progress Mapping

In IOTP, all six Authentication and Payment-related transactions are implemented via combinations of document exchanges. Within a transaction, progress is exhibited by the execution of successive document exchanges. Within a document exchange, progress is exhibited by the internal state changes of the trading roles. Accordingly, we can define a progress mapping which is based on the current document exchange in the transaction and also the internal states of trading roles within each document exchange.

Table 1. The four mappings ψ_m, ψ_c, ψ_{ph} and ψ_{dh} enumerating and ordering the internal states of the four trading roles (Merchant, Consumer, Payment Handler and Delivery Handler) in a document exchange.

Trading Role Internal States	$\psi_m(M)$	$\psi_c(M)$	$\psi_{ph}(M)$	$\psi_{dh}(M)$
READY	1	1	1	1
LISTEN	2	4	-	-
HOLD	-	5	2	-
WAIT	3	2	3	2
HOLD_WAIT	4	3	4	-
FINISHED	5	6	-	-
COMPLETED	6	7	5	3
CANCELLED	7	8	6	4
no state	0	0	0	0

For a given transaction, the sequence of internal states of each trading role is fixed. The internal state of each trading role can be obtained by considering the token on the place corresponding to the given transaction type on the corresponding Trading Role page, as described in Section 4.4 for the Consumer and Merchant. To capture the progress from the trading role internal states within a document exchange, we define four progress mappings ψ_m, ψ_c, ψ_{ph} and ψ_{dh}, which respectively enumerate the internal states of the Merchant, the Consumer, the Payment Handler and the Delivery Handler according to the progress represented by each. Table 1 lists the sequences of trading role internal states for a Purchase transaction. Not all trading roles can enter all states and so no mapping has been defined for these cases (indicated by '-' in the table.) The bottom row of Table 1 indicates that trading roles that have not yet entered any state (no state) map to 0.

We also define a mapping to capture the progress of the execution of document exchanges in a transaction. An Authentication (Auth) exchange always takes place at the beginning of a transaction (that requires authentication of the Consumer). An Offer exchange, either Brand Dependent (BDOfr) or Brand Independent (BIOfr), never occurs before an Authentication exchange but must occur before any other document exchange. A Payment (Pay1) or a Payment-and-Delivery (PayDlv) exchange is carried out after an Offer exchange only, and a Delivery (Deliv) exchange happens after a Payment exchange only. A Value Exchange transaction involves two Payment exchanges, where the second Payment (Pay2) must follow the first Payment (Pay1) exchange. It can be seen that there is always a choice between a Brand Dependent Offer and a Brand Independent Offer in any Payment-related transaction.

This knowledge of the behaviour of IOTP is useful in two ways. The first is that it allows us to reason about the order in which the document exchanges occur. IOTP is sequential in its ordering of document exchanges within a transaction and thus we can give a higher progress value to those states in which the trading roles have progressed further. Secondly, we can use this knowledge to explore the part of the occurrence graph for each document exchange combination sequentially. Conceptually, when arranging the OG in a least-progress-first manner, this is tantamount to reshaping the OG to be *long and thin* rather than *short and wide* to minimise the number of states with a given progress value, which, in our experience, is beneficial to the performance of the sweep-line method. To illustrate this, we describe the effect of reshaping the OG for a Purchase transaction. The Purchase transaction OG can be initially divided into two parts based on the choice of a Brand Dependent or Brand Independent Offer. We choose to always explore the part relating to the Brand Dependent Offer first, and then the part involving the Brand Independent Offer. Similarly, there is always a choice between a Payment followed by a Delivery exchange and a (combined) Payment-and-Delivery exchange in a Purchase transaction, and so the corresponding transaction OG following each Offer

exchange can be further divided into two parts. In this case, we choose to explore first the part with Payment followed by Delivery, then the part with the Payment-and-Delivery exchange.

By defining a mapping that reflects these observations, a transaction OG can be generated in such a way that on one hand the correct progress of the transaction is preserved (i.e. the progress mapping results in monotonic progress) and on the other hand a narrower OG is obtained. Such a mapping is defined in Table 2 as ψ_{exch_comb} which enumerates all possible combinations of document exchanges executed at each of the four trading roles. Similarly to identifying the internal state of each trading role, the document exchange that a trading role is involved in can be obtained from the token on the place corresponding to the given transaction on the corresponding Trading Role page (see Section 4.4). A trading role with no document exchange information (represented by '-') is not involved, or has not yet started a transaction. In Table 2, the value $\psi_{exch\_comb}(M)$ is incremented by 26 for each different combination of document exchanges. The offset of 26 was chosen as it is one greater than the maximum sum over all trading roles of the trading role internal state progress mappings (7+8+6+4=25, see Table 1) in a document exchange. This is important, as an offset of one greater than 25 guarantees that there will be no overlap between the progress mappings in ψ_{exch_comb} for any two combinations of document exchange states at each trading role.

Having identified sources of IOTP-specific progress, we can now define the full progress mapping $\psi_{specific}$ for IOTP as follows:

$$\psi_{specific}(M) = \psi_{exch\_comb}(M) + \psi_m(M) + \psi_c(M) + \psi_{ph}(M) + \psi_{dh}(M)$$

The performance of the sweep-line with this progress mapping is discussed in Section 6.2.

5.3 Combination of Generic and Specific Progress Mapping

The mapping $\psi_{specific}$ takes advantage of knowledge of the sequence of states that trading roles progress through, and also of the sequential nature of IOTP operations. It does, however, lack potential for scalability. Ideally, we would like a progress mapping for the IOTP CPN to incorporate the parameter $RCmax$ in such a way as to scale with $RCmax$. The generic progress mapping $\psi_{generic\_2}$ 'grows' with the parameter $RCmax$ so by combining this with the specific progress mapping we hope to obtain a progress measure with the advantages of both.

Rather than simply adding together the progress values $\psi_{generic\_2}(M)$ and $\psi_{specific}(M)$ for each state M, we take note of the fact that message identifiers and retransmission counters increment *within* a particular document exchange. When combining the progress measures, we give a weighting to the IOTP-specific progress values to make an increment in the IOTP-specific progress values more significant than any increment in the generic progress values. This weight must be one more than the maximum value of $\psi_{generic\_2}$. To determine this value we must first determine an upper bound on the message identifier values.

IOTP has 15 different (non-error and non-cancel) message types, each of which may be used at most once during a transaction. (We omit the details of these messages for brevity.) Each new message sent by a trading role is given a message identifier one greater than the previously sent message. Retransmissions of the same message are given the same message identifier as the original. The receiver of one of the 15 message types may generate an error message requesting a retransmission. Each error message is given a new (incremented) message identifer. Thus, in the worst case, sending a (non-error and non-cancel) message to a peer trading role will cause the receiving trading role's internal message identifier to increment by $RCmax$ (after $RCmax$ retransmissions) or by $RCmax+1$ if the message being sent stimulates

Table 2. The mapping ψ_{exch_comb} enumerating all possible combinations of document exchanges at each of the four trading roles in a transaction.

Merchant	*Consumer*	*PHandler*	*DHandler*	$\psi_{exch\_comb}(M)$
NoExch	-	-	-	0
Auth	-	-	-	26
Auth	NoExch	-	-	52
Auth	Auth	-	-	78
BDOfr	-	-	-	104
BDOfr	NoExch	-	-	130
BDOfr	Auth	-	-	156
BDOfr	BDOfr	-	-	182
BDOfr	Pay1	-	-	208
BDOfr	Pay1	NoExch	-	234
BDOfr	Pay1	Pay1	-	260
BDOfr	Deliv	Pay1	-	286
BDOfr	Deliv	Pay1	NoExch	312
BDOfr	Deliv	Pay1	Deliv	338
BDOfr	PayDlv	-	-	364
BDOfr	PayDlv	NoExch	-	390
BDOfr	PayDlv	PayDlv	-	416
BDOfr	Pay2	Pay1	-	442
BDOfr	Pay2	Pay2	-	468
BIOfr	-	-	-	494
BIOfr	NoExch	-	-	520
BIOfr	Auth	-	-	546
BIOfr	BDOfr	-	-	572
BIOfr	Pay1	-	-	598
BIOfr	Pay1	NoExch	-	624
BIOfr	Pay1	Pay1	-	650
BIOfr	Deliv	Pay1	-	676
BIOfr	Deliv	Pay1	NoExch	702
BIOfr	Deliv	Pay1	Deliv	728
BIOfr	PayDlv	-	-	754
BIOfr	PayDlv	NoExch	-	780
BIOfr	PayDlv	PayDlv	-	806
BIOfr	Pay2	Pay1	-	832
BIOfr	Pay2	Pay2	-	858

a response from the receiving trading role. Thus $15(RCmax+1)$ is an upper bound on message identifiers, where 15 is the number of messages and $(RCmax+1)$ is the maximum increase in message identifier induced by each message. Cancel messages terminate the transaction and are never retransmitted and thus are not taken into account.

Based on this, an upper bound on the value of $\psi_{generic\_2}$ is $4(15(RCmax+1)^2+RCmax)$ where $RCmax$ is the maximum possible value of $\texttt{GetRC}_{tr}$ and the 4 comes from the summation over all 4 trading roles. This is quite a conservative upper bound, as each trading role transmits only a subset of the set of 15 messages. The weight needs to be larger than this value, and so we obtain the following combined progress mapping:

$$\psi_{comb}(M) = \psi_{generic\_2}(M) + (4(15(RCmax+1)^2 + RCmax) + 1) * \psi_{specific}(M)$$

The performance and scalability of this progress measure is discussed in Section 6.3.

In [12] the sweep-line method was used to verify the Wireless Transaction Protocol (WTP) [36] within the Wireless Application Protocol (WAP) [37]. Conventional occurrence graph generation could only be used for retransmission counters up to 5 due to state explosion. In contrast to our method of mapping directly to $\mathbb{N}$, a monotonic progress mapping was

defined as a *progress vector* in $\mathbb{Z}^5$, embedded into the total ordering ($\leq$) on integers. The five elements of progress were the state of each of the two interacting protocol entities, the retransmission counters in each of the two interacting protocol entities and the inverse of the number of messages left in the channel once one or both of the protocol entities had reached their final states. This allowed WTP to be verified using sweep-line for retransmission counters up to 8, the value specified in [36] for WAP operating over IP networks. The progress mapping in [12] is similar to our definition of ψ_{comb} as we also use protocol entity states and retransmission counters to gauge the progress of IOTP. In addition, we use message identifiers but we do not use the number of messages left in the channel after a transaction has completed.

6 Experimental Results

The IOTP CPN was analysed with the computer tool Design/CPN, using conventional OG generation and a prototype implementation of the sweep-line method using the generic progress mapping $\psi_{generic\_2}$, the IOTP-specific progress mapping $\psi_{specific}$ and the combined progress mapping ψ_{comb}. The results are shown in Tables 3, 4 and 5 respectively. We present results from the Purchase Transaction only, as this transaction exercises all elements of the IOTP CPN model.

When analysing IOTP, one of the desired properties of IOTP is valid transaction termination. If a transaction terminates properly, each of the trading roles that have started the transaction must enter a valid terminal state, i.e. `COMPLETED`, indicating that a transaction terminates successfully, or `CANCELLED`, indicating the transaction is cancelled. We are able to check the dead markings obtained and verify that they all have this property.

To eliminate differences in performance due to the efficiency of state storage, a custom hashing function for storing states was implemented and used in both the conventional generation and the sweep-line generation. All experiments were conducted on a 2.6GHz Pentium 4 with 1Gb of memory.

It is worth noting that the node deletion mechanism in the sweep-line implementation used in this paper differs from the experimental implementation used in [6] and [24], where node deletion was initiated every time a statically defined number of new nodes were explored. In addition, as reported in [24], the former method for node deletion performed poorly and became the dominant time factor when exploring large occurrence graphs. The implementation used in this paper matches the algorithm more closely, in that states with a progress value $n \in \mathbb{N}$ are deleted as soon as the minimum progress value over all unprocessed states is greater than n.

6.1 Generic Progress Mapping

Table 3 contains the OG statistics when using the sweep-line method with $\psi_{generic\_2}$. Column 1 shows the value of the $RCmax$ parameter for which the OG was generated. The second, third and fourth columns show the number of states and arcs generated by conventional exploration (the number of states and arcs in the full OG) and the total time taken. Column 5 shows the peak number of states stored in memory at any one time using the sweep-line method. Columns 6 and 7 show the total number of states swept (explored) and arcs traversed with the sweep-line method. The total time for sweep-line exploration is shown in column 8. The number of dead markings discovered is shown in column 9. Columns 10 and 11 show the factor of reduction in space and time when using the sweep-line method as compared to conventional generation. We were limited to $RCmax = 4$ for conventional generation and

$RCmax = 5$ with the sweep-line method before computer memory and time contraints forced us to abandon generation. We were unable to generate the full OG (using the conventional method) for $RCmax = 5$ so the number of states and arcs in the full state space and the theoretical reduction are shown in brackets.

The first thing to notice is that this progress mapping is non-monotonic, as indicated by having swept more states than are in the full state space. This was not unexpected, as the message identifiers and retransmission counters reset to 0 at various times during an IOTP transaction. The factor of reduction in states is relatively constant over this range of $RCmax$ values, staying around 1.7 times more efficient in space. This is not a particularly useful reduction. It is worth noting that the progress mapping does not scale as well as we had hoped, hence the relatively constant (but worsening) reduction in space. The reduction in time is due to a lower peak state storage and so each new state does not need to be compared with as many existing states.

Table 3. Sweep-line statistics for the analysis of the IOTP CPN using $\psi_{generic\_2}$.

RCmax	Conventional			Sweep-line				Dead	Reduction in	
	States	Arcs	hh:mm:ss	Peak	Total	Arcs	hh:mm:ss	markings	space	time
0	2144	5243	00:00:03	1462	2534	6338	00:00:04	85	1.47	0.75
1	12695	37947	00:00:30	7141	17521	52675	00:00:36	85	1.78	0.83
2	47931	161437	00:04:21	26369	62158	208473	00:02:43	85	1.82	1.60
3	142499	518910	00:29:59	84853	176126	635844	00:09:37	85	1.68	3.12
4	361451	1389461	02:55:09	227012	430713	1638636	00:31:04	85	1.59	5.64
5	(816957)	(3266339)	-	523953	946571	3743931	01:33:39	85	(1.56)	-

6.2 IOTP Specific Progress Mapping

Table 4 contains the state space statistics when using the sweep-line method with $\psi_{specific}$. The table format is the same as in Table 3. This progress mapping results in a monotonic measure of progress (this is checked automatically by the tool during exploration) so the total number of states and arcs explored with the sweep-line method is identical to the total number of states and arcs in the full state space, hence we can infer the size of the full state space. The reduction in space and time is better than when using $\psi_{generic\_2}$. The space reduction worsens as $RCmax$ increases, indicating that this progress measure also scales poorly. The worsening in reduction is caused by the number of states with a given progress value growing out of proportion with the total number of states, as $RCmax$ increases. This worsening in memory reduction seems to go against the usual trend, as in e.g., $[2, 6, 12, 24]$ where the reduction in space increases with the size of the state space.

Table 4. Sweep-line statistics for the analysis of the IOTP CPN using $\psi_{specific}$.

RCmax	Conventional			Sweep-line				Dead	Reduction in	
	States	Arcs	hh:mm:ss	Peak	Total	Arcs	hh:mm:ss	markings	space	time
0	2144	5243	00:00:03	191	2144	5243	00:00:04	85	11.21	0.75
1	12695	37947	00:00:30	1266	12695	37947	00:00:27	85	10.03	1.11
2	47931	161437	00:04:21	5528	47931	161437	00:02:10	85	8.67	2.01
3	142499	518910	00:29:59	18876	142499	518910	00:08:12	85	7.55	3.66
4	361451	1389461	02:55:09	61025	361451	1389461	00:27:02	85	5.92	6.48
5	(816957)	(3266339)	-	163776	816957	3266339	01:22:36	85	(4.99)	-
6	(1690370)	(6961367)	-	385077	1690370	6961367	03:44:06	85	(4.39)	-
7	(3260531)	(13739782)	-	819848	3260531	13739782	10:24:42	85	(3.98)	-

6.3 Combined Progress Mapping

Table 5 contains the state space statistics when using the sweep-line method with ψ_{comb}. The table format is again the same as used previously. The reduction in space obtained by using the combined progress mapping is identical to using $\psi_{specific}$ for small values of $RCmax$ but as $RCmax$ increases, the reduction in space does not worsen as rapidly. The reduction in time is approximately the same as when using $\psi_{specific}$ even though the peak state storage is lower, a fact that we attribute to the more complicated progress mapping function (hence a larger overhead in calculating the progress value for each state).

An unexpected result we discovered was that ψ_{comb} is monotonic, at least for the cases we examined ($0 \leq RCmax \leq 7$). We conjecture that this is due to the events causing a decrease in progress with respect to $\psi_{generic\_2}$ are also causing an increase in progress with respect to $\psi_{specific}$. Due to the weighting we have given $\psi_{specific}$ in ψ_{comb} the increase in progress due to a changing trading role internal state or document exchange is greater than the decrease caused by the reset of a message identifier or retransmission counter.

The number of dead markings is constant for all configurations, whereas we would expect the number of dead markings to explode with $RCmax$. The reason is that the IOTP CPN resets retransmission counters upon termination. We can deduce that there are 85 ways that the IOTP CPN can terminate independently of retransmission counter values.

Table 5. Sweep-line statistics for the analysis of the IOTP CPN using ψ_{comb}.

RCmax	Conventional			Sweep-line				Dead	Reduction in	
	States	Arcs	hh:mm:ss	Peak	Total	Arcs	hh:mm:ss	markings	space	time
0	2144	5243	00:00:03	191	2144	5243	00:00:04	85	11.21	0.75
1	12695	37947	00:00:30	1266	12695	37947	00:00:27	85	10.03	1.11
2	47931	161437	00:04:21	5528	47931	161437	00:02:10	85	8.67	2.01
3	142499	518910	00:29:59	17163	142499	518910	00:08:10	85	8.30	3.67
4	361451	1389461	02:55:09	46369	361451	1389461	00:27:02	85	7.80	6.48
5	(816957)	(3266339)	-	124693	816957	3266339	01:22:08	85	(6.55)	-
6	(1690370)	(6961367)	-	294108	1690370	6961367	03:44:53	85	(5.75)	-
7	(3260531)	(13739782)	-	628845	3260531	13739782	10:14:15	85	(5.18)	-

7 Conclusions and Future Work

This paper has presented some experiments in applying the sweep-line occurrence graph method to a CPN model of the Internet Open Trading Protocol. We have particularised the sweep-line method to CPNs, where we use the Naturals as the progress values and its usual order relation ($\leq$). This allows us to just associate a progress mapping with the CPN. We then present a revised abstract generalised sweep-line algorithm for CPNs, which includes the storing of dead markings and uses set operations.

We derived three progress mappings for the analysis of the IOTP CPN model and presented our intuition and rationale behind each. The first is a progress mapping based on generic properties of protocols. The second is based on properties specific to IOTP. The third is based on a combination of the first two.

The IOTP CPN is parameterised with a maximum number of retransmissions, $RCmax$. Using the combined progress mapping, we were able to analyse the IOTP CPN for larger values of $RCmax$ than was possible using conventional occurrence graph generation. In doing so we have demonstrated that the sweep-line method can be successfully applied to a complex real-life example.

We have represented our progress values as a summation of component progress values. An alternative representation would be to use vectors (as in [12]) to represent the progress values. This would allow easier investigation of different orderings and weightings of the components making up the progress mapping. It also allows use of the compositional sweep-line method [23] for analysis of the IOTP CPN. This method takes advantage of monotonic and non-monotonic components of the measure of progress to further reduce peak state storage. We intend to investigate this approach to determine what gains in space and time can be made.

The design of optimal (or even good) progress mappings for CPN models of practical systems is an open question. When defining progress mappings for IOTP, we have used rules of thumb aimed at giving a good progress mapping with respect to the peak number of states stored and the time required for OG exploration. The rules of thumb are: 1) the progress sources should be weighted according to their significance in the system; 2) regress edges should be avoided or the number kept as low as possible; 3) the number of progress values should be as high as possible; and 4) a progress mapping that works well for small configurations of the system will also work well for larger configurations. Currently, these rules are based on intuition and a limited amount of practical experience. As part of future work it would be useful to develop some formal and/or further practical justification for these rules to obtain a better understanding of what constitutes a good progress mapping. The phenomenon observed in this paper where the reduction decreased for larger configurations suggests that 4) is not a valid assumption in general. It would also be of interest to determine the best possible progress mapping that can be defined for the system. This would allow the analyst to gauge the potential reduction that can be obtained with the sweep-line method.

References

1. J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer-Verlag, 2004.
2. J. Billington, G.E. Gallasch, L.M. Kristensen, and T. Mailund. Exploiting equivalence reduction and the sweep-line method for detecting terminal states. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):23–37, January 2004.
3. D. Burdett. Internet Open Trading Protocol - IOTP Version 1.0. RFC 2801, IETF, April 2000.
4. D. Burdett, D.E. Eastlake, and M. Goncalves. *Internet Open Trading Protocol*. McGraw-Hill, 2000.
5. S. Christensen, K. Jensen, and L.M. Kristensen. *Design/CPN Occurrence Graph Manual*. Department of Computer Science, University of Aarhus, Denmark. On-line version: `http://www.daimi.au.dk/designCPN/`.
6. S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proceedings of TACAS 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer-Verlag, 2001.
7. E.M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Logic Model Checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.
8. Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.
9. E.A. Emerson and A.P. Sistla. Symmetry and Model Checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.
10. R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, IETF, June 1999.
11. P. Godefroid, G.J. Holzmann, and D. Pirottin. State-Space Caching Revisited. *Formal Methods in System Design*, 7(3):227–241, 1995.
12. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In *Proceedings of ICATPN'02*, volume 2360 of *Lecture Notes in Computer Science*, pages 182–202. Springer-Verlag, 2002.
13. G.J. Holzmann. Tracing protocols. *AT&T Technical Journal*, 64(10):2413–2433, December 1985.
14. G.J. Holzmann. Algorithms for Automated Protocol Validation. *AT&T Technical Journal*, 69(2):32–44, 1990.

15. G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.
16. G.J. Holzmann. An Analysis of Bitstate Hashing. *Formal Methods in System Design*, 13(3):287–305, 1998.
17. InterPay I-OTP.
URL: `http://www.ietf.org/proceedings/01aug/slides/trade-1/index.html`, August 2001.
18. K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9(1/2):7–40, 1996.
19. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Springer-Verlag, 2nd edition, 1997.
20. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods*. Springer-Verlag, 2nd edition, 1997.
21. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 3, Practical Use*. Springer-Verlag, 1997.
22. L.M. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
23. L.M. Kristensen and T. Mailund. A Compositional Sweep-line State Space Exploration Method. In *Proceedings of FORTE'02*, volume 2529 of *Lecture Notes in Computer Science*, pages 327–343. Springer-Verlag, 2002.
24. L.M. Kristensen and T. Mailund. A Generalised Sweep-Line Method for Safety Properties. In *Proceedings of FME'02*, volume 2391 of *Lecture Notes in Computer Science*, pages 549–567. Springer-Verlag, 2002.
25. L.M. Kristensen and T. Mailund. Efficient Path Finding with the Sweep-Line Method using External Storage. In *Proceedings of the International Conference on Formal Engineering Methods (ICFEM'03)*, volume 2885 of *Lecture Notes in Computer Science*, pages 319–337. Springer-Verlag, 2003.
26. C. Ouyang. *Formal Specification and Verification of the Internet Open Trading Protocol using Coloured Petri Nets*. PhD thesis, Computer Systems Engineering Centre, School of Electrical and Information Engineering, University of South Australia, Adelaide, Australia, June 2004.
27. C. Ouyang and J. Billington. An improved formal specification of the Internet Open Trading Protocol. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC 2004)*, pages 779–783, Nicosia, Cyprus, 14-17 March 2004. ACM Press.
28. C. Ouyang and J. Billington. Formal Analysis of the Internet Open Trading Protocol. In *Proceedings of the 1st International Workshop on Theory Building and Formal Methods in Electronic/Mobile Commerce (TheFormEMC)*, Toledo, Spain, 1-2 October 2004, in press. Springer-Verlag.
29. C. Ouyang, L.M. Kristensen, and J. Billington. A formal and executable specification of the Internet Open Trading Protocol. In *Proceedings of 3rd International Conference on Electronic Commerce and Web Technologies*, volume 2455 of *Lecture Notes in Computer Science*, pages 377–387, Aix-en-Provence, France, 2-6 September 2002. Springer-Verlag.
30. A.N. Parashkevov and J. Yantchev. Space Efficient Reachability Analysis Through Use of Pseudo-Root States. In *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 50–64. Springer-Verlag, 1997.
31. D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.
32. Standard SMart Card Integrated SettLEment System Project SMILE Project.
URL: `http://www.ietf.org/proceedings/99mar/slides/trade-smile-99mar`, April 1999.
33. K. Schmidt. Automated Generation of a Progress Measure for the Sweep-Line Method. In *Proceedings of TACAS'04*, volume 2988 of *Lecture Notes in Computer Science*, pages 192–204. Springer-Verlag, 2004.
34. A. Valmari. A Stubborn Attack on State Explosion. In *Proceedings of CAV'90*, volume 531 of *Lecture Notes in Computer Scienc*, pages 156–165. Springer-Verlag, 1990.
35. A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.
36. WAP Wireless Transaction Protocol Specification. June 2000 Conformance Release. Available via: `http://www.wapforum.org/`.
37. Wireless Application Protocol. Specifications available via: `http://www.wapforum.org/`.
38. P. Wolper and P. Godefroid. Partial Order Methods for Temporal Verification. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer-Verlag, 1993.
39. P. Wolper and D. Leroy. Reliable Hashing withoug Collision Detection. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 59–70. Springer-Verlag, 1993.

Towards a Methodology for Modeling with Petri Nets

Christine Choppy and Laure Petrucci
LIPN, UMR CNRS 7030, Institut Galilée - Université Paris XIII
99 Avenue Jean-Baptiste Clément, F-93430 Villetaneuse, FRANCE
email: {Christine.Choppy,Laure.Petrucci}@lipn.univ-paris13.fr

Abstract

Formal specifications remain difficult to write in general, due to both the complexity of the system to be developed, and the use of a formal language. In [4], a method is proposed for specification development, with CASL, the Common Algebraic Specification Language, and CASL-LTL, an extension for dynamic systems specification, as target languages. However, this method could be used with quite a variety of modeling languages, as shown in this paper which is a first attempt to provide systematic guidelines for Petri net specification on the ground of the aforementioned specification method. It is shown how to express in terms of Petri nets the constituent features and the properties exhibited from the first specification approach. A model train specification from [2] is used as a running example.

1 Motivation

While formal specifications are well advocated when a good basis for further development is required, they remain difficult to write in general. Among the problems are the complexity of the system to be developed, and the use of a formal language. So potential helps are needed to start the specification, and then some guidelines to remind some essential features to be described. In [4], a method is proposed for specification development, with CASL[3], the Common Algebraic Specification Language, and CASL-LTL[13], an extension for dynamic systems specification, as target languages. However, this method could be used with quite a variety of target languages.

Petri nets have been successfully used for concurrent systems specification. Among its attractive features, is the combination of a graphic language and an effective formal model that may be used for formal verification. Expressivity of Petri nets is dramatically increased by the use of high-level/coloured Petri nets, and also by the addition of modularity features. Thus, quite sizable examples were specified with Petri nets.

While the use of Petri nets becomes much easier with the availability of high quality environments and tools, to our knowledge, little work was devoted to

a specification methodology for Petri nets. The aim of this work is to provide guidelines for Petri net specification on the grounds of the aforementioned specification method. A train specification [2] is used as a running example.

The structure of the paper is as follows. We first describe the train example in Section 2, then the general specification method [4] is presented in Section 3. The proposed guidelines for Petri net specification are presented in Section 4, together with their application on the train example, and the Petri net specification is given in Section 5.

2 The model train example

Our running example will be the toy railway from [2], in which a step-by-step modeling of the railway by students was described.

The project assigned to students was not only designed as an approach to parallel programming, but also to emphasize the benefits of specification and validation prior to programming. In particular, the students were asked to produce a graphical model, having the same appearance as the physical railway. This was not required for aesthetic reasons but because it greatly helps to understand whether a configuration of the railway is correct or not. This eases a boring and error-prone task of synthesizing a long sequence of transitions. It represents an important benefit for debugging. It also permits to make a direct correspondence between the physical train devices and the Petri net model.

The physical model railway is depicted in Figure 1. It consists of about 15 meters of tracks, divided into 16 sections (blocks B1 to B16) plus 2 sidetracks (ST1 and ST2), connected by four switches and one crossing. The way the trains can pass the switches and the crossing is indicated by the arrows in Figure 1. The traffic on all tracks can go both ways. Although one can notice that switch 1 (and also switch 2) is composed of two elementary ones, it is managed as a single unit, due to the short distance between the two physical components. The railway is connected to a computer via a serial port which allows to read information from sensors and send orders to trains through the tracks or directly to switches. Each section is equipped with one sensor at each end, to detect the entrance or exit of a train. The orders sent to trains can be either stop or go forward/backwards at a given speed.

Hierarchical coloured Petri nets [11] were chosen as a model, due to their tool support for hierarchies, simulation, and occurrence graphs, e.g. DESIGN/CPN [12, 10]. Hierarchies allowed a structured design, where the top-level net reflects the hardware layout. The use of high-level nets permits both capturing several cases by a single transition and representing the parameters of trains and track sections by one place. The use of an ordinary net leads to unreadable intricate models.

The model described in [2] adopts an adaptive routing strategy for the trains

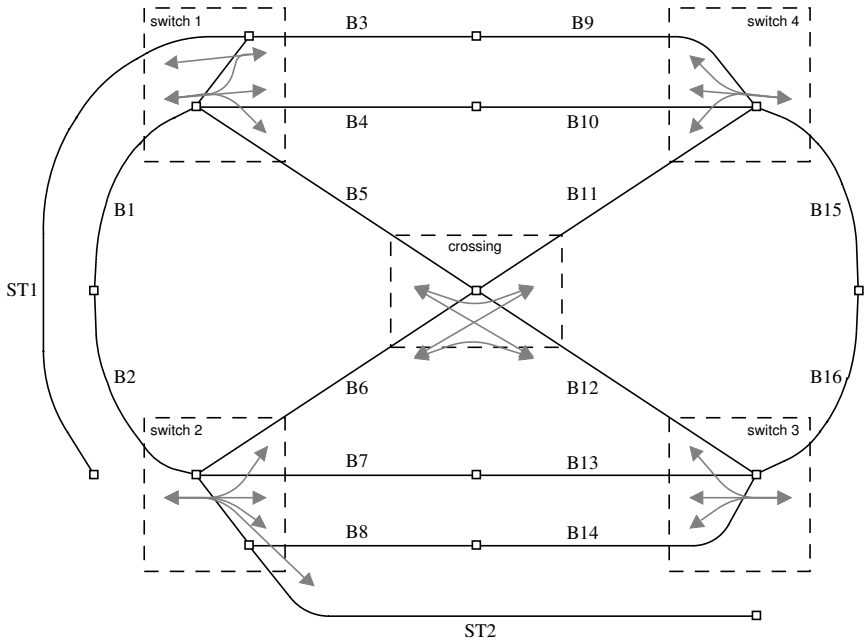


Figure 1: The tracks of the model railway.

to circulate. Hence, the behavior of trains adapts to local conditions. Namely, at each switch, the train's route can be chosen among several tracks and a train may even go back when it cannot continue forward.

Although surprising at the first glance, such behavior of trains offers several complex routing possibilities, demanding the students to design a routing policy so that safety and operational requirements are fulfilled.

3 Specification method principles

The method presented in [4] aims at helping a modeler in designing a "*software item*". It assumes that a *software item* may be either of the following:

- a *simple dynamic system* (a dynamic interacting entity in isolation, e.g., a sequential process) or
- a *structured dynamic system* (a community of mutually interacting entities, simple or also structured), or
- a *data structure* (or data type).

Items are characterized by their *parts* and *constituent features*, that are subsequently specified. For instance, the parts of simple systems are data structures, and their constituent features are states and elementary interactions def-

initions (cf. Section 3.1). The method also involves quite a precise guidance on which properties should be expressed, and in which way.

Among the various specification styles, the *property-oriented* (or *axiomatic*) and *constructive* (or *model-oriented*) ones are mostly used, and here we shall focus on the property-oriented one which is relevant at the beginning of the specification task. In any case, [4] advocates that a visual presentation should be provided to help reading the formal specification, and also that comments should be used, e.g., to accompany formulae.

Property-oriented specification The semantics of *property-oriented specification* is basically defined as follows: "a model belongs to the semantics of a property-oriented specification if and only if all formulae of the specification are valid on it".

The methodological ideas supporting this specification style are:
the item is described at a certain moment in its development by expressing all its "relevant" properties using sentences provided by the formalism (formulae).

For each software item, the *property-oriented* specification technique, is given, by providing the abstract structure of the corresponding specifications together with the related visual presentation and corresponding formal specification.

The target languages are initially CASL[3], the Common Algebraic Specification Language, and CASL-LTL[13], an extension designed for the dynamic systems specification by giving a CASL view to LTL, the Labeled Transition Logic ([1, 9]). LTL, and thus CASL-LTL, is based on the idea that a dynamic system is considered as a *labeled transition system* (shortly *lts*), and that to specify it one has to specify the labels, the states and the transitions of such a system. Recall that an *lts* is a triple $(State, Label, \rightarrow)$, where $\rightarrow \subseteq State \times Label \times State$.

Subsequent work [5, 7, 6] showed that this method could also be used with other target languages, e.g., UML. Although UML is not a formal language, the formally grounded approach used there conveys a quite systematic development for the description, and of course, OCL may be used to describe some of the properties.

In the following, we focus on simple systems items since they are used in the first step when applying our method. Structured systems will be discussed in the conclusion and addressed in further work.

3.1 Simple systems

Here the word *system* denotes a dynamic system of any kind, and so evolving with time, without any assumption about other aspects of its behaviour. Thus it may be a communicating/nondeterministic/sequential/... process, a reactive/parallel/concurrent/distributed/... system, but also an agent or an agents system. A *simple system* is a system without any internal components cooperating together.

Simple systems are seen formally as *labeled transition systems*. The states of an *lts* modeling a simple system represent the relevant intermediate situations in the life of the system, and each transition $s \xrightarrow{l} s'$ represents the *ability* of the system in the state/situation s of evolving to the state/situation s'; the label l contains information on the conditions on the external environment for this ability to become effective, and on the transformation induced on this environment by the execution of the transition, i.e., it fully describes the interaction of the system with the external environment during this transition.

To design effective and simple specification methods, the labels are assumed to have the standard form of a set of *elementary interactions*, where each elementary interaction intuitively corresponds to an elementary (that is, not further decomposable) exchange with the external environment. It is also assumed that the elementary interactions are of different types, and that each type is characterized by a name and by some arguments (elements of some data structures). Thus, *elementary interaction types* (just *elementary interactions* from now on) are constituent features of the simple systems.

The form of the states (which are the intermediate situations during the system's life) is also a characterizing feature of simple systems, therefore *state constituent features* are needed. However, they are technically different for the property-oriented and the constructive case.

Finally, to define the constituent features of a simple system, values of various *data structures* are used; they are the "parts" of the simple systems.

3.2 Simple systems property-oriented specifications

The property-oriented specification method for simple systems requires to first find the parts and constituent features, and then to express the properties. In order to keep the specification level abstract, the states are not completely described, but only a list of what should be observed is given, and thus the state features will correspond to elementary observations on the states (*state observers*). A state observer is characterized by a name, some arguments (elements of some *data structures*), and by the observed value (element of some data structure). Figure 2 shows the structure (by means of a UML class diagram[1]) of a property-oriented specification of a simple system, and Figure 3 shows how to visually depict its parts (DATA_1, ..., DATA_j) and the constituent features.

3.3 Simple systems properties

All the properties about a simple system correspond to properties on the *lts* modeling it, and thus on its labels, states and transitions. These properties may express which are the admissible sets of *elementary interactions* building a label, and link the source state, the label and the target state of a transition.

[1] To shortly explain the UML notation, the diamond connects the "Simple system specification" with its constituents, the * indicates the multiplicity (as in regular expressions), and labels on the lines provide a "role" name for each part.

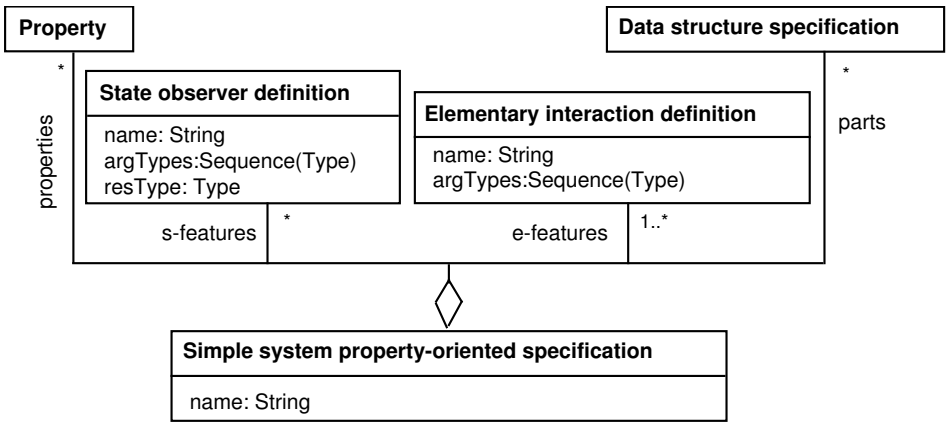


Figure 2: Simple System Property-Oriented Specification.

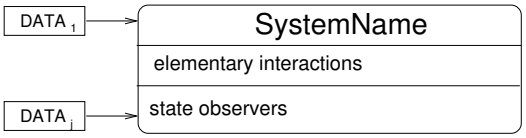


Figure 3: Visual presentation of a simple system: parts and constituent features.

The properties may also provide some information on the values observed by the various state observers on a state.

More precisely, *label properties* express when, under some condition, two different elementary interactions are incompatible, i.e., no label may contain both (cf. incompat1 and incompat2 in Figure 4). *State properties* describe conditions the values returned by the state observers should satisfy for any state (cf. value1 and value2 in Figure 4). State formulae may also include special atoms, expressing properties on the *paths* (concatenated sequences of transitions) leaving/reaching the state, that is on the future/past behaviour of the system from this state. *Transition properties* are conditions on the state observers applied to the source and target states of the transition.

Guidelines for properties follow a general tableau method which gives provision for "property cells" with respect to the system constituent features. Since the constituent features of simple systems are of two kinds, elementary interactions and state observers, five kinds of "property cells" are considered:

- properties on an elementary interaction,
- properties on a state observer,
- relationship between two elementary interactions,
- relationship between two state observers,

- relationship between an elementary interaction and a state observer.

Schemas for these five property cells are described in Figure 4, with, for each cell, the list of possible properties.

In Figures 5 and 6 the details of two schemas are given, providing, for each property, its name, an informal comment, and its formal expression in a visual presentation associated with CASL-LTL. There, *arg* stands for generic expressions of the correct types, possibly with free variables, and *cond*(*exprs*) for a generic condition where the free variables of *exprs* may appear.

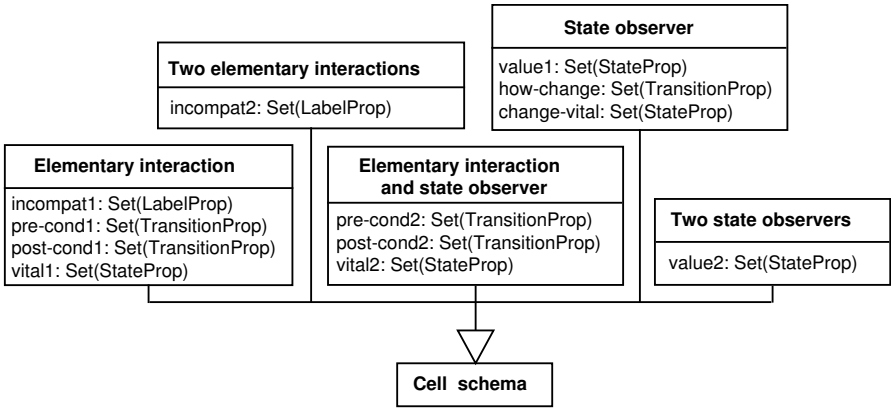


Figure 4: Simple System Cell schemas

4 Applying the specification approach to the train example

The general ideas [4] described in Section 3 were initially introduced to help designing an algebraic specification. We will show here that these principles can also be applied to coloured Petri nets, through the model train example.

4.1 Parts and constituent features

In order to apply the general ideas given in Section 3, we first need to choose what kind of software item our system is. When dealing with systems like the model train example, we may consider, in a first approach, that a single entity is involved (the railway), and therefore that it has to be specified as a *simple dynamic system*. This will lead to a general high-level design of the system, not getting into the details of trains changing sections policies.

According to Figure 2 need to find the system (sub)parts and constituent features. In both cases, the parts are the data structures required in the system.

pre-cond1 (transition property) If the source state of the transition satisfies some condition then the label of a transition contains some instantiation of *ei*.

if *cond*(*arg*) **then** *ei*(*arg*) **happens**
where some source state observers must appear in *cond*(*arg*) and the target state observers cannot appear in *cond*(*arg*).

post-cond1 (transition property) If the label of a transition contains some instantiation of *ei*, then the target state of the transition must satisfy some condition. The condition on the target state may require also the source state to be expressed.

if *ei*(*arg*) **happens then** *cond*(*arg*)
where some target state observers must appear in *cond*(*arg*) and the source state observers may appear in *cond*(*arg*).

incompat1 (label property) Two instantiations of *ei* are incompatible (i.e., no label may contain both) if their arguments satisfy some conditions.

ei(arg_1) **incompatible with** *ei*(arg_2) **if** *cond*(arg_1,arg_2)

vital1 (state property) If a state satisfies some condition, then any path (sequence of transitions) starting from it will eventually contain a transition whose label contains *ei*. Note that in these properties **in any case** may be replaced by **in one case** and **eventually** by **next**.

if *cond*(*arg*) **then in any case eventually** *ei*(*arg*) **happens**

Figure 5: Elementary interaction (*ei*) cell schema

value1 (state property) The results of the observation made by *so* on a state must satisfy some conditions.

cond, where *so* must appear in *cond*.

how-change (transition property) If the observed value changes during the occurrence of a transition, and some elementary interactions belong to the transition label, then some condition on source and target states, old and new values holds (new values are denoted with a $'$).

if $so(arg) = v_1$ **and** $ei_1, \ldots, ei_n$ **happened**
then $so'(arg) = v_2$ **and** $v_1 \neq v_2$ **and** $cond(v_1,v_2,arg)$

change-vital (state property) If a state satisfies some condition, then the observed value will change in the future. Note that in these properties **in any case** may be replaced by **in one case** and **eventually** by **next**.

if $cond(v_1,v_2,arg)$ **and** $so(arg) = v_1$ **and** $v_1 \neq v_2$ **then**
in any case eventually $so(arg) = v_2$

Figure 6: State observer (*so*) cell schema

The constituent features are the elementary interactions and the state description features (observers or constructors). At this first stage of model design, the property-oriented approach is often more relevant, thus we need state observers to start with.

The physical system is made of track sections, switches between track sections, and trains. Thus, *state observers* should provide information on the layout of tracks, i.e. which track sections are contiguous, which ones are connected by switches, whether a train is present on a track, and, when this is the case, in which direction it is traveling (this may be expressed in various ways, e.g. here, clockwise or anticlockwise).

The *elementary interactions* (that are associated with a state change of the system) are a train track section change, moving either directly between contiguous sections or between sections connected by a switch. It is admitted that the position of a track section is fixed (sic!), and that the potential connections that can be established by a given switch are also fixed, and this will be reflected in the state observers properties.

The required *data structures* are obtained through the data types used by the state observers and the elementary interactions. Quite obviously, some data type is needed to refer to track sections, and to switches. Since they are named in Figure 1 (i.e., the possible values are known and in a quite limited number), so-called enumerated types are adequate. The same principle is used

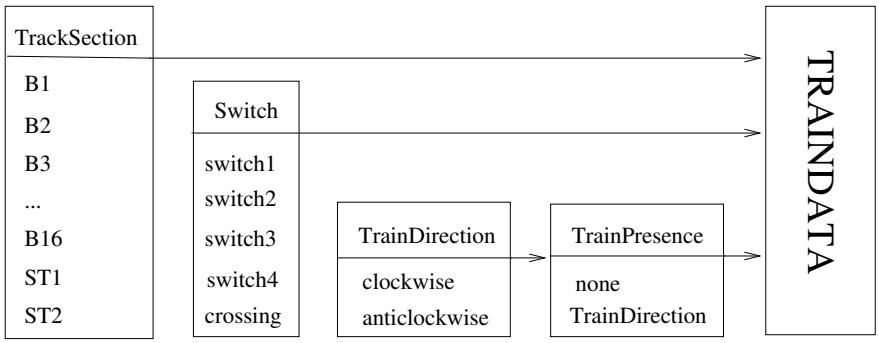


Figure 7: The data structures.

for train directions, as shown in Figure 7. We present here also the corresponding CASL specification for these data where the type name is simply followed by the enumeration of its possible values (which are constants of this type). The **free** construct insures that no property relates (e.g., equates) these values, so that they are all different. The **sort** construct is used here to express that any element of the type $TrainDirection$ is also of the type $TrainPresence$.

spec TRAINDATA =
  free type
    $TrackSection ::= B1 \mid B2 \mid B3 \ldots \mid B16 \mid ST1 \mid ST2$
  free type
    $Switch ::= switch1 \mid switch2 \mid switch3 \mid switch4 \mid crossing$
  free type
    $TrainDirection ::= clockwise \mid anticlockwise$
  free type
    $TrainPresence ::= none \mid$ **sort** $(TrainDirection)$
  end

These data will be reflected either in the names of states and transitions of the Petri net, or as colours of tokens.

The *state observers* are chosen so as to provide enough information on the state of the modeled system. For our example, observers are needed to describe the track sections layout, as well as the presence of a train with its travel direction (Figure 8). The *connected* predicate is used to express when two track sections are directly connected, and in which train direction. The *switched* predicate is used to express when two lists of track sections are connected through a switch, and in which train direction. These observers (*connected* and *switched*) are fixed once the railway topology is fixed. This is not the case for *train_present* which reflects a situation that evolves with time, and that depends on the initial state as well as the history of elementary interactions leading to the current state. The state and history type specifications are given below.

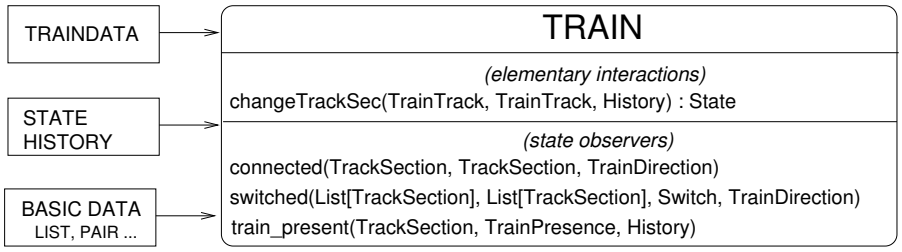


Figure 8: The train elementary interactions and state observers.

spec STATE =
  sort *State*;
  op *initial* : *State*; %% There is an initial state
    %% which will be further described in the Section 5
  end

spec HISTORY = STATE **then**
  type $History ::= initial \mid \_\_.\_\_(History; State)$;
  op $last : History \rightarrow State$;
  vars $h : History; s : State$;
  axioms
    $last(initial) = initial$;
    $last(h.s) = s$;
  end

The *elementary interactions* express the fact that a train changes track section (Figure 8).

4.2 Properties

Once the parts and constituent features of the system are specified, its properties should be expressed. Following the method described in Section 3.3, the property cells of Figure 4 should be filled. Since there is only one elementary interaction, the relationship between two elementary interactions is skipped.

Properties on a state observer

value1 (state property) Here we express the results of observations. The properties on *connected* and *switched* do not change and express the railway topology. The properties on *train_present* vary with the state.

$connected(B1, B2, anticlockwise)$
$connected(B2, B1, clockwise)$
...

$switched((ST1, B1), B3), switch1, clockwise)$
$switched((B1), (B3, B4, B5), clockwise)$
$switched((B3), (B1, ST1), anticlockwise)$
$switched((B3, B4, B5), (B1), switch1, anticlockwise)$
...
$train\_present(B1, none, initial)$ ...

how-change (transition property) As mentioned above, this concerns only *train_present* which varies when a track section change $changeTrackSec$ occurs.

if $train\_present(TS_i, TP_i, h) \wedge train\_present(TS_j, none, h)$
$\wedge\ changeTrackSec(< TS_i, TP_i >, < TS_j, none >, h)$ **happened**
then $(TP_i \neq none) \wedge train\_present(TS'_i, none, h') \wedge$
$train\_present(TS'_j, TP_i, h')$
where h' denotes $h.changeTrackSec(< TS_i, TP_i >, < TS_j, none >, h)$

change-vital (state property) This property is not relevant here.

Properties on the elementary interaction $changeTrackSec$

pre-cond1 (transition property) A track section change is defined when the two track sections are connected or "switched", when there is a train traveling in the (connection or switch) direction in the first track section, and no train in the second one.

if $(connected(TS_i, TS_j, TP_i) \vee$
$\exists sw : Switch\ s.t.\ swichted((\ldots, TS_i, \ldots), (\ldots, TS_j, \ldots), sw, TP_i))$
$\wedge\ (TP_j = none)$
then $changeTrackSec(< TP_i, TS_i >, < TP_j, TS_j >, h)$ **happens**

post-cond1 (transition property) After a track section occurred, the train is in the target track section.

if $changeTrackSec(< TP_i, TS_i >, < TP_j, TS_j >, h)$ **happens then** $(TP'_j = TP_i)$

incompat1 (label property) This property should express when simultaneous train track section changes should not occur. Since the information on the direction of the train is included in the interaction, the only case is that, at a given switch, a train cannot take simultaneously several directions.

$changeTrackSec(< TP_i, TS_i >, < TP_j, TS_j >, h)$ **incompatible with**
$changeTrackSec(< TP_i, TS_i >, < TP_k, TS_k >, h)$
if $\exists sw : Switch\ s.t.\ swichted((\ldots, TS_i, \ldots), (\ldots, TS_j, \ldots, TS_k, \ldots), sw, TP_i)$
$\wedge (T_j \neq T_k)$

vital1 (state property) There is no property here since it is not relevant here to express that a track section change will eventually happen.

There are no properties between the state observers, and the properties expressing the relationship between the elementary interaction and the *train_present* state observer are redundant with those already expressed.

In the methodology introduced here, some properties can be specified, which are not part of the Petri net model per se. For example, the modeler could specify a state property (see Figure 6) expressing that, unless otherwise imposed by the initial state, there is always a single token in each place representing a track section (which is inferred by the ***pre-cond1*** of *changeTrackSec* above).

Even though this property seems extremely simple, it is important to guide the modeler into explicitly writing down the expected properties from the system, based on the current status of the model being designed.

In later phases of system development, a simple system can evolve by refining its constituents, or by composing it with other systems. Stating expected properties is then crucial to have better insight. These properties could be verified by a model-checking tool, in order to check consistency of the model w.r.t. the intended behaviour.

5 From the specification to the coloured Petri net

The *state observers* are reflected in the Petri net in different ways. The *fixed part*, that is here the way track sections are connected, together with the potential switch connections, may be reflected by the Petri net layout, as suggested in the pedagogical project of [2], thus it will be observable on the grounds that it will be possible for a train to move from one track section to another (connected) one. More precisely, the Petri net places reflect the different track sections, and places that model adjacent track sections are connected with transitions associated with a train changing track section. Following [2], it is suggested that places and transitions are displayed so as to reflect the physical model train track and switches display.

Quite obviously then, *elementary interactions* reflecting a train changing track section are specified by the corresponding transitions.

The presence of a train together with its direction (none, clockwise or anticlockwise) comes here as a *colour* for the track section places.

The *pre-conditions* and *post-conditions* properties of the elementary interactions (see Figure 5) induce the arcs between places and transitions.

Hence, we obtain a model which is similar to the prime page of [2] presented in Figure 9.

The prime page represents the whole railway, without any consideration of the policy used to move from one section to the next. This policy is described in

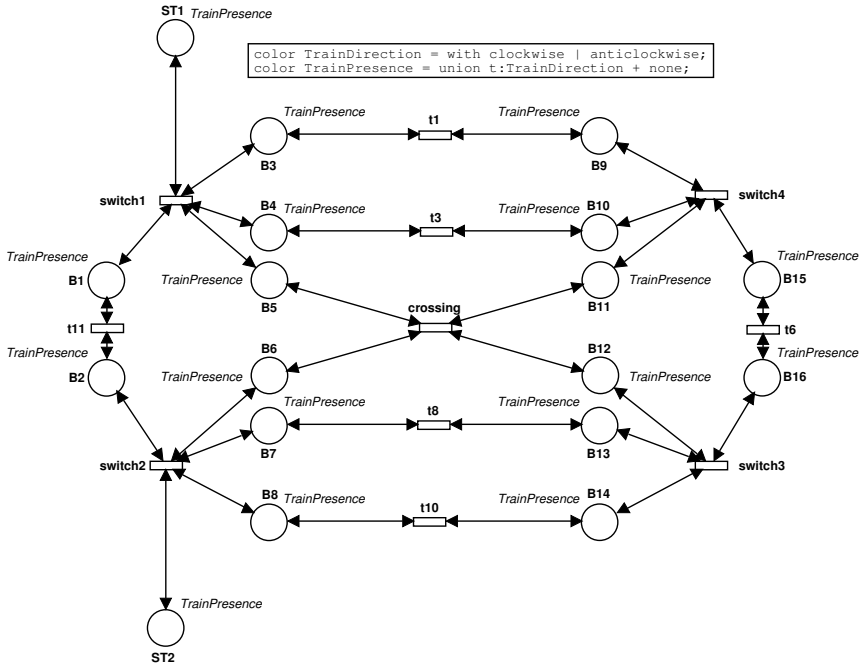


Figure 9: The prime page of the model railway hierarchical coloured Petri net.

sub-pages, corresponding to the different switches and moves between adjacent sections. A single look at this prime page shows the current state, i.e. where the different trains are located. The similarity between the physical railway model (Figure 1) and the prime page (Figure 9) is easily observed. The places represent the sections (they have the same names in both figures), while the transitions indicate the possible moves.

The colors (data types) of tokens within places are defined in the *global declaration node* (boxed text at the top of Figure 9). First, the direction of a train, `TrainDirection`, can be either `clockwise` or `anticlockwise`. Each place represents a railway section with the corresponding name and thus always contains one token of color `TrainPresence`, with a value characterizing the state of the section, that is either a train is in the section, or the section is empty. This is expressed with the union type:

```
color TrainPresence = union t:TrainDirection + none;
```

All the transitions are substitution transitions, i.e. their behaviour is explicited on the associated subpage. They precisely describe the policy used to change sections. In this paper, we will not get into the details of these policies.

Now, the initial situation chosen for the railway is that there are trains traveling in the clockwise direction on track sections B9 and B10, and trains

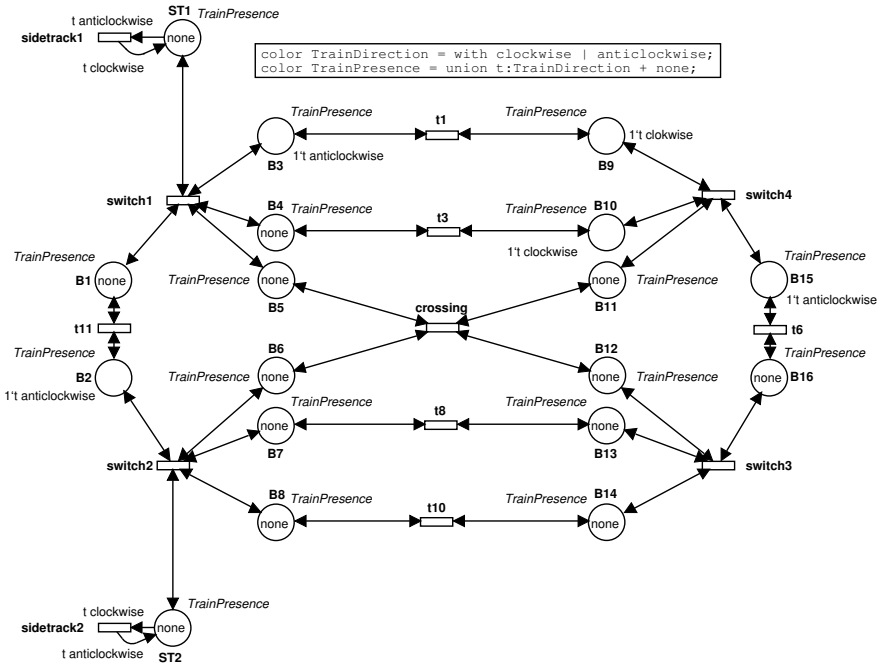


Figure 10: The prime page with the initial marking.

traveling in the anticlockwise direction on track sections B2, B3 and B15. This is reflected by the following properties:

$train\_present(B9, clockwise, initial)$;
$train\_present(B10, clockwise, initial)$;
$train\_present(B2, anticlockwise, initial)$;
$train\_present(B3, anticlockwise, initial)$;
$train\_present(B15, anticlockwise, initial)$;

which are represented by an initial marking in the Petri net of figure 10.

6 Conclusion and perspectives

In this paper, we provided guidelines for specifying "simple systems" using Petri nets. These guidelines are derived from a method developed in [4] for an algebraic specification language and an extension for dynamic systems specification. In particular, elements provided for specifying "simple systems" (consisting of a single dynamic entity) were studied for their expression with Petri nets. The (sub)parts are data structures (that can be used e.g. for the Petri net colors), and we refer to the method in [4] for their specification. The state description features may be reflected either in the Petri net layout (i.e., the way places and transitions are connected), or in the information conveyed in the places. The

elementary interactions are reflected in the transitions firings. The properties for the state descriptors and the elementary interactions may be checked against the Petri net properties or behaviour.

This first experiment with a model train seems quite promising in the direction of providing a more extensive method for Petri net specification. It shows that the methodology envisioned applies to a large panel of specification languages, which are in essence quite different.

This work should be pursued by extending the "structured systems" part of the approach described in [4]. Our approach should then be extended so as to include the communication mechanisms between modules provided by Petri nets (e.g. hierarchical CPNs [11], modular Petri nets [8]). This should also include property verification, i.e. if a general property is to be satisfied, it would be nice to know at which level of the specification process a formal analysis should (in)validate it. Applying this methodology to design step-by-step a complex case study is another important issue.

References

[1] E. Astesiano and G. Reggio. Labelled Transition Logic: An Outline. *Acta Informatica*, 37(11-12):831–879, 2001.

[2] G. Berthelot and L. Petrucci. Specification and validation of a concurrent system: An educational project. *Journal of Software Tools for Technology Transfer*, 3(4):372–381, 2001.

[3] M. Bidoit and P.D. Mosses. *CASL User Manual, Introduction to Using the Common Algebraic Specification Language*. Lecture Notes in Computer Science 2900. Springer-Verlag, 2004.

[4] C. Choppy and G. Reggio. Towards a Formally Grounded Software Development Method. Technical Report DISI–TR–03–35, DISI, Università di Genova, Italy, 2003. Available at `ftp://ftp.disi.unige.it/person/ReggioG/ChoppyReggio03a.pdf`.

[5] C. Choppy and G. Reggio. Improving use case based requirements using formally grounded specifications. In *Fundamental Approaches to Software Engineering*, LNCS 2984, pages 244–260. Springer Verlag, 2004.

[6] C. Choppy and G. Reggio. A uml-based method for the commanded behaviour frame. In K. Cox, J.G. Hall, and L. Rapanotti, editors, *Proc. of the 1st International Workshop on Advances and Applications of Problem Frames (IWAAPF 2004)*, pages 27–34. An ICSE 2004 workshop, IEEE, 2004.

[7] C. Choppy and G. Reggio. Using uml for problem frame oriented software development. In Walter Dosch and Narayan Debnath, editors, *Proc of the*

ISCA 13th Int. Conf. on Intelligent and Adaptative Systems and Software Engineering (IASSE-2004), pages 239–244. The International Society for Computers and Their Applications (ISCA), 2004.

[8] S. Christensen and L. Petrucci. Modular analysis of Petri nets. *The Computer Journal*, 43(3):224–242, 2000.

[9] G. Costa and G. Reggio. Specification of Abstract Dynamic Data Types: A Temporal Logic Approach. *T.C.S.*, 173(2):513–554, 1997.

[10] DESIGN/CPN *online*. `http://www.daimi.au.dk/designCPN`.

[11] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and practical use. Volume 1: basic concepts*. Monographs in Theoretical Computer Science. Springer, 1992.

[12] META Software and Aarhus University. *Design/CPN 3.0*, 1996. Also available as: `http://www.daimi.au.dk/designCPN`.

[13] G. Reggio, E. Astesiano, and C. Choppy. CASL-LTL : A CASL Extension for Dynamic Reactive Systems Version 1.0– Summary. Technical Report DISI-TR-03-36, DISI – Università di Genova, Italy, 2003. Available at `ftp://ftp.disi.unige.it/ person/ReggioG/ReggioEtAll03b.ps` and `ftp://ftp.disi.unige.it/ person/ReggioG/ReggioEtAll03b.pdf`.

Experience using Coloured Petri Nets to Model TCP's Connection Management Procedures

Bing Han and Jonathan Billington
Computer Systems Engineering Centre
University of South Australia
Mawson Lakes, SA 5095, Australia
Email: Bing.Han@postgrads.unisa.edu.au
Jonathan.Billington@unisa.edu.au

Abstract

The Transmission Control Protocol (TCP) is the most widely used transport protocol in the Internet, providing a reliable data transfer service to many applications. This paper presents a formal model of TCP connection management using Coloured Petri nets. The model is created to verify TCP's functional correctness (e.g., the absence of deadlocks and livelocks) rather than its performance properties. This paper also discusses different modelling approaches and issues raised during this process in the hope that it is helpful for others to specify other complex protocols.

1 Introduction

The Transmission Control Protocol (TCP) [3, 28] is a complex protocol, designed over 20 years ago to provide a reliable data transfer service, so that applications (e.g. WWW and Email) can be assured that data will be delivered in order and without loss or duplication. The operation of TCP was originally specified in RFC 793 [28] using narrative descriptions, message sequence diagrams, and a finite state machine (FSM) diagram. It was then improved and modified in [1, 3, 9, 10, 15, 21, 25]. The number of bugs reported in TCP implementations [25] spans 60 pages. This and other experience lead us to believe that a more formal approach to TCP specification may prove beneficial.

TCP comprises a connection management protocol for establishing and terminating connections and a data transfer protocol for reliable data transfer. This paper focuses on modelling the connection management protocol and the aim of the modelling is to examine its functional correctness, e.g., correct termination. The basic idea of modelling TCP is that we consider two peer TCP entities, communicating over the Internet Protocol (IP) as well as interacting with their application processes. Figure 1 illustrates the system which we specify using Coloured Petri nets (CPNs).

We specify TCP's behaviour according to the narrative description of an implementation example provided in Section 3.9 of RFC 793, which gives the details needed for investigating TCP's functional correctness. The protocol is specified using hierarchies. To ensure our specification of TCP precisely reflects the contents of Section 3.9, we not only conduct manual consistency checks but also automatic model validation by using simulation [16]. The CPN model presented in this paper is the result of incremental and iterative revision of two earlier versions of the model [13, 14]. In addition to providing an in-depth specification, we also consider the following aspects that are usually avoided in the literature. Our specification accommodates each side establishing a connection *simultaneously*, which is not included in [17, 32]. We use a realistic model of IP that can lose, delay and re-order packets. We model packet corruption indirectly by loss, since packets with retransmission errors will be discarded either by routers or by hosts, which

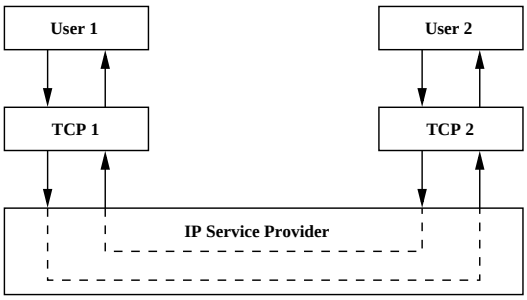


Figure 1: TCP and its environment

amounts to being lost. Duplicate packets can appear in the channel as a result of retransmissions by TCP.

This paper is organised as follows. Section 2 provides an introduction to TCP connection management. Section 3 reviews related work. Section 4 describes the scope and assumptions made for creating the CPN model. Sections 5 provides a detailed description of the CPN model that is organised in a hierarchy. Section 6 discusses alternative modelling approaches and identifies our chosen methods by showing the reasons. Finally, Section 7 concludes this paper.

2 TCP Connection Management

TCP is a connection-oriented protocol. Before data transfer begins, a connection needs to be set up between two end points. An end point is identified by a pair comprising an IP address (a 32-bit integer assigned to each host in the Internet) and a *port* number (a 16-bit integer that identifies a process within a given host). The pair comprising an IP address and a port is known as a *socket* [28]. A connection between two end points is identified by a pair of sockets.

TCP uses a *three-way handshake* [34] to open a connection, that is, three segments are exchanged by the two communicating entities. The three-way handshake is used to prevent the connection from being opened by an old duplicate segment from an earlier connection instance [36]. A TCP connection is full duplex allowing concurrent and independent data flow in both directions. After finishing sending data, each TCP entity closes the connection in one direction and can still receive data from the other direction. The connection is fully released when both ends close in an orderly manner and the procedure is known as *orderly release*. The connection establishment, release and abort procedures are known as *TCP Connection Management*, which is critical for reliable data delivery.

2.1 TCP Segment Format

A TCP segment is a sequence of 32-bit words, comprising header fields and a data field. As shown in Figure 2 (taken from Figure 3 of [28]), the first six rows make up TCP header fields that provide control information for *end-to-end* communication and the last row is the data field that contains a portion of data from an application.

At the beginning of the header fields are the 16 bit source and destination port fields. They are used to identify two communicating application processes running over a connection. Every octet of data sent by the TCP entity is assigned a sequence number. The sequence number field contains the sequence number of the first data octet in the segment, known as the sequence number of the segment, which is used to detect duplicate segments and preserve the order of the segments in the stream. The acknowledgement number field contains the next sequence number that the sender of the segment is expecting to receive. The 4 bit data offset field contains the header length in 32 bit words and it indicates where the data starts. The reserved field is kept for future use.

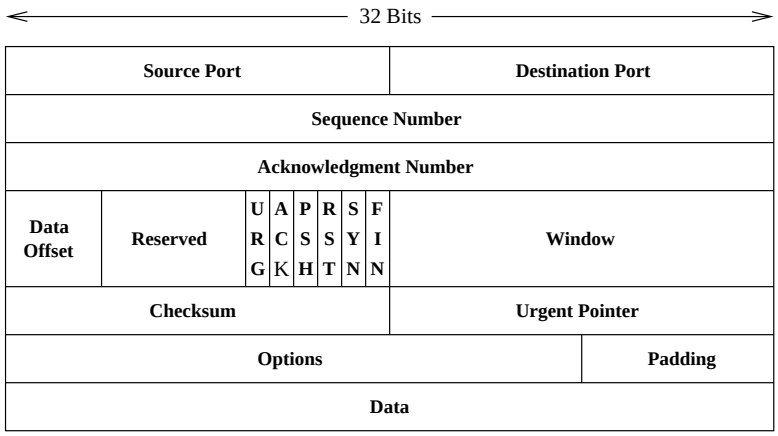


Figure 2: TCP segment format

Next to the reserved field are six 1-bit control flags: URG (urgent), ACK (acknowledgement), PSH (push), RST (reset), SYN (synchronisation), and FIN (finish). The URG flag if set, is used in conjunction with the urgent pointer field, to indicate to the receiver the position of data in the octet stream that should be processed immediately upon arrival. The ACK flag if set, indicates that the acknowledgement number field of the segment is valid. A segment with PSH on indicates that it contains 'push' data which the sending TCP entity has transmitted without waiting for its send buffer to be filled, and the receiving TCP entity should pass immediately to its application for processing without waiting for its receive buffer to be full. When set, the RST flag informs the receiver of the segment to reset the connection.

A TCP connection is initiated by a segment which has the SYN flag set. The sequence number of the SYN segment is the initial sequence number for the connection. Instead of being set to 0, the initial sequence number is selected according to the value of a clock that the host runs. This is to reduce the probability of old duplicate SYNs from earlier connection instances being accepted by the current connection. A segment with the FIN flag set indicates that the sender of the segment has no more data to send. As there can be duplicate SYNs or FINs in the channel as a result of retransmissions, a SYN and a FIN segment each is assigned one sequence number to prevent confusion as to whether the SYN or FIN has been received before. The SYN occupies one sequence number before the first data octet of the segment, whereas the FIN occupies one sequence number after the last data octet of the segment [28]. Note that the SYN and FIN are the only control bits that are assigned a sequence number.

The window field contains the maximum number of data octets that the sender of the segment is able to receive and is used for flow control. The checksum field is used to verify that the segment is received without bit errors. Next to it is the urgent pointer field that indicates where the urgent data (if there is any) ends. The options field conveys information (e.g., maximum segment size [8]), which the sending TCP entity uses to negotiate with the other TCP entity. The padding field is used to ensure that the TCP header ends and data begins on a 32-bit boundary.

2.2 Transmission Control Block

To provide reliable data delivery, TCP maintains the state of a connection by storing a set of variables in a data structure known as the *transmission control block* (TCB) [28]. Among these variables, the important ones for TCP connection management are: send oldest unacknowledged (SND_UNA), send next (SND_NXT), initial send sequence number (ISS) and receive next (RCV_NXT).

When TCP transmits a segment, it increments SND_NXT. When TCP accepts a segment, it increases RCV_NXT and sends an acknowledgement. Upon the receipt of an acknowledgement, TCP advances SND_UNA. The amount by which each of the three variables is increased is

given by the length of the segment in octets, i.e., the sequence space for both data and control bits SYN and FIN. The SYN and FIN bits each occupy one sequence number [28]. If the SYN or FIN segment does not carry data, the amount by which each variable is advanced is 1. For example, if a SYN is sent with sequence number 100, we have SND_NXT=100+1 at the sender side. When the SYN is received, we have RCV_NXT=100+1 at the receiver side. The receiver then sends its own SYN with sequence number of say 300 and acknowledgement number (denoted by SEG_ACK) of 100+1. When the sender receives the acknowledgement to its SYN, the oldest unacknowledged number SND_UNA is updated with the acknowledgement number of the incoming segment, i.e., SND_UNA=SEG_ACK.

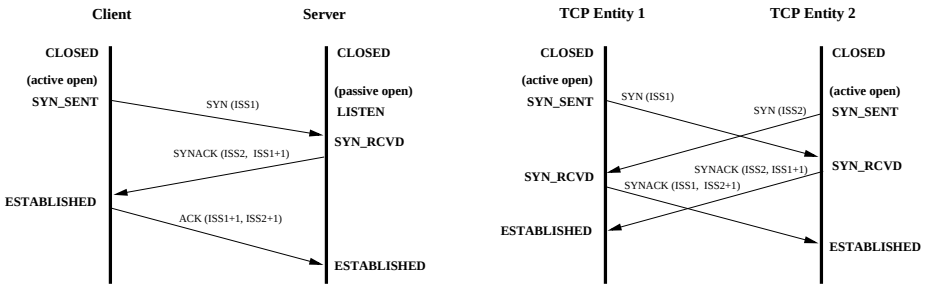
2.3 Functional Behaviour

In this section, we describe how TCP establishes and releases a connection.

2.3.1 Connection Establishment

A connection is initiated by the TCP entity (TCP client) that sends a SYN segment, and is responded to by the peer TCP entity (TCP server). The TCP server receiving a SYN segment has no way of telling whether it is a new SYN to open a connection or an old duplicate SYN from an earlier incarnation. Therefore it must ask the other side (through the exchange of segments) to verify the identity of the SYN. This process is illustrated in Figure 3 (a) with a time sequence diagram.

On the left side of the figure is the TCP client (the initiator of the connection) and on the right is the TCP server. Time progresses down the page. The client's states (CLOSED, SYN_SENT and ESTABLISHED) are written to the left of the vertical line representing the client. A similar convention is adopted for the server side. User commands (i.e., active open and passive open) are written in parentheses, indicating when they occur. In Fig. 3, the sequence number and the acknowledgement number (when relevant) are included with the segment name. For example, the ACK segment has sequence number ISS1+1 and acknowledgement number ISS2+1, and is written as ACK(ISS1+1,ISS2+1).



(a) Normal establishment (b) Simultaneous establishment

Figure 3: Message sequences for TCP connection establishment

Initially no connection exists between the client and the server, so both TCPs are CLOSED. After receiving the *active open* command from its user, the TCP client creates a TCB that stores its state information for a connection. It selects an initial send sequence number, ISS1, according to a 32-bit clock, and sends out a SYN segment with this initial sequence number. The TCP client then moves into the SYN_SENT state, waiting for a corresponding connection acknowledgement from the TCP server.

On the other side of the connection, the TCP server has already received a *passive open*

command from its user and has created its own TCB. The TCP server has changed state from CLOSED to LISTEN, and is waiting for an incoming connection request. When the SYN segment arrives, the server chooses its own ISS and sends a segment which has both SYN and ACK bits set. As shown in the figure, this segment, usually referred to as a SYNACK, has sequence number ISS2 and acknowledgement number ISS1+1, confirming the receipt of the SYN segment. The server then enters the SYN_RECEIVED (abbreviated to SYN_RCVD) state. If the *passive open* command had not been issued when the SYN segment arrives, the TCP server will remain in CLOSED and the connection will not be established.

After receiving the SYNACK segment from the server, the client sends a segment with the ACK bit set. It has sequence number ISS1+1 and acknowledgement number ISS2+1, acknowledging the receipt of the SYNACK. The client then enters the ESTABLISHED state from SYN_SENT. Upon receiving the ACK segment from the client, the server enters ESTABLISHED from SYN_RCVD. The connection is now fully set up, allowing data to be transferred in both directions.

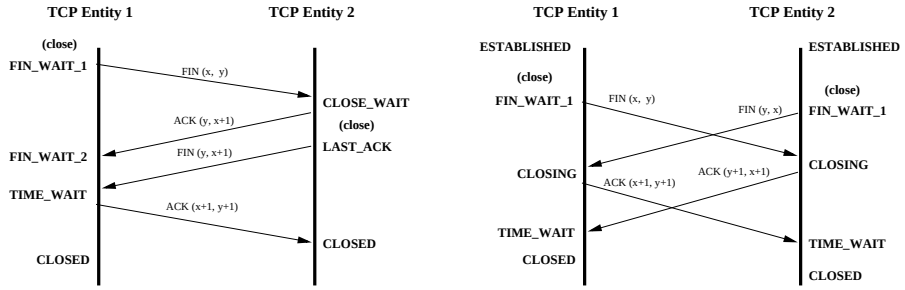
TCP also allows both sides to initiate a connection *simultaneously*. As shown in Figure 3 (b), the TCP entity on each side receives an *active open* command from its user and sends a SYN to the other side before receiving the SYN from the other side. The SYNs from each side have initial sequence numbers ISS1 and ISS2. On sending the SYN, each TCP entity changes state from CLOSED to SYN_SENT. After receiving the SYN from the other side, each TCP entity changes state from SYN_SENT to SYN_RCVD and acknowledges the SYN with a SYNACK. The SYNACK from TCP entity 1 to TCP entity 2 has sequence number ISS1 and acknowledgement number ISS2+1, and the SYNACK from TCP entity 2 to TCP entity 1 has sequence number ISS2 and acknowledgement number ISS1+1. Upon receipt of the SYNACK from the other side, both TCP entities go into ESTABLISHED, indicating the connection is fully set up.

2.3.2 Connection Release

When a user at either end of a connection is finished sending data, it closes the connection. TCP connection release is an orderly operation, which involves two transactions. Firstly, TCP entity 1 initiates the procedure and TCP entity 2 responds to it. This transaction closes the connection from entity 1 to 2. Next, once it has transmitted all its data, TCP entity 2 initiates the procedure to close the connection in the opposite direction. The orderly release ensures that all data is received before the connection is fully closed.

We now describe the release procedure in detail. As shown in Fig. 4, the procedure begins with a user issuing a *close* command to its TCP entity (i.e., TCP entity 1), which results in a FIN segment being sent out. Assume the FIN segment has sequence number x and acknowledgement number y. If no data is transferred by TCP entity 1 after the connection is established (see Fig. 3 (a)), then x=ISS1+1 and y=ISS2+1. After sending the FIN, TCP entity 1 changes state from ESTABLISHED to FIN_WAIT_1, waiting for an acknowledgement of the FIN segment. After receiving the FIN from TCP entity 1, TCP entity 2 enters the CLOSE_WAIT state (waiting for its user to close the connection) and sends out ACK segment. The ACK has an acknowledgement number x+1, that is, the sequence number of the FIN segment plus 1. The sequence number of the ACK depends on whether data has been sent before the FIN is received. If no data is sent, then the ACK has sequence number y, as shown in the figure.

TCP entity 1 receives the ACK from TCP entity 2 and changes state from FIN_WAIT_1 to FIN_WAIT_2, waiting for a connection release request from TCP entity 2. Meanwhile, data can still be transmitted from TCP entity 2 to 1 (but not vice versa) until the user TCP entity 2 issues a *close* command. After its user issues the *close* command, TCP entity 2 sends out a FIN segment. Suppose no data is sent by TCP entity 2 after it receives the FIN from TCP entity 1. The sequence number of the FIN sent to TCP entity 1 is the same as that of its preceding ACK segment. The acknowledgement number of the FIN is also x+1. Next TCP entity 2 enters state LAST_ACK, waiting for an acknowledgement of the FIN, that is, the last acknowledgement for



(a) Normal release

(b) Simultaneous release

Figure 4: Message sequences for TCP connection release

the connection.

Upon receipt of the FIN segment, TCP entity 1 enters the TIME_WAIT state and responds to the FIN with an ACK segment that has sequence number x+1 and acknowledgement number y+1. When TCP entity 2 receives the ACK from TCP entity 1, it enters CLOSED from LAST_ACK. TCP entity 1 remains in TIME_WAIT for two maximum segment lifetimes (MSL) before entering CLOSED. The MSL is the longest time that a segment can exist in the Internet (about 2 minutes).

It is possible for each end of a TCP connection to terminate the connection *simultaneously*, as shown in Fig. 4 (b). TCP entity 1 receives a *close* command from its user and sends out a FIN segment with sequence number x and acknowledgement number y. Before the FIN arrives, TCP entity 2 receives a *close* from its user and sends out a FIN with acknowledgement number x. After TCP entity 1 sends its FIN and if TCP entity 2 does not send data, then the FIN sent by TCP entity 2 has sequence number y, as shown in the figure. After receiving the FIN from the other end, each TCP entity sends out an ACK and changes state from FIN_WAIT_1 to CLOSING. Because the FIN consumes one sequence number, the sequence number of the ACK is the sequence number of the FIN from the same side plus 1. The acknowledgement number of the ACK is the sequence number of the FIN from the other side plus 1. Both TCPs enter state TIME_WAIT on receipt of the ACK from the other side, and go to CLOSED after two maximum segment lifetimes.

3 Related Work

Previous work on modelling TCP Connection Management is mainly confined to the *establishment* procedure. The modelling of the release procedure is either greatly simplified or omitted from most investigations. Almost all the previous work is based on early versions of TCP, to which continuing changes were made. The official TCP specification [3, 28] is not well examined in terms of functional correctness.

In their pioneering work on verifying TCP connection management, Sunshine and Dalal [34] address the design issues of early versions [5–7] of TCP, which was still evolving at that time. They conducted informal case studies (manual walk-through of the sequences) to investigate the functional behaviour of TCP's connection establishment procedures. However, simultaneous opening of connections is not addressed in [34], nor the orderly release procedure. This significantly simplifies the study of TCP connection management. Sunshine and Dalal [34] envisaged that to verify TCP with greater certainty, a *precise* model of the protocol would be required and appropriate analysis techniques developed.

Based on another early version [27] of TCP, Schwabe [29] specified the connection establishment protocol using the SPEX language [30]. SPEX is based on a non-deterministic state transition system. However, the simultaneous open Schwabe investigated is different from that

in the current TCP specification [28], where TCP sends a SYNACK rather than an ACK upon receiving a SYN.

Kurose and Yemini [18] specify the connection establishment protocol based on another version [26] of TCP. They only consider the client-server situation and specify the protocol using a PASCAL-like language.

In [19], Lin specifies a connection establishment protocol [34, 37] using finite state machine and verifies the connection will be eventually established through logical deduction. Again, only the client-server connection is examined. In addition, the behaviour of the TCP server is not correctly specified as it does not go through the LISTEN state.

Mehrpour and Karbowiak [22] model and analyse an early and simplified version [33] of TCP using Numerical Petri Nets [35]. They model TCP segments by their names without sequence and acknowledgement numbers. Hence, the model is not precise with these important details missing. Also the model is incomplete in that none of the arc inscriptions are given.

In [23, 24], Murphy and Shankar specifies a transport protocol using a state transition model and invariant and progress assertions. The protocol is similar to TCP only in connection establishment procedures.

Smith [31, 32] specifies TCP Connection Management using the *general timed automaton* [20] and follows a phase-based approach, which is different from the way TCP is specified in [28]. This makes it difficult to validate this model against the official TCP specification. As well as excluding simultaneous opening of connections and user aborts, Smith's specification does not address the following details that are part of TCP's functional behaviour and can have an impact on its logical correctness: (1) the release of a connection in any state including SYN_RCVD, (2) entering TIME_WAIT from FIN_WAIT_1 upon receiving a FINACK segment, as specified in RFC 1122 [3], and (3) state variable SND_UNA (send oldest unacknowledged number) that is used to check whether an ACK is a duplicate segment.

In [13], we provide a quite simple model of TCP Connection Management based on the FSM diagram [28]. The model does not include protocol details such as sequence numbers and state variables. A more detailed model enhanced with these features is given in [2, 14], which is structured according to a *state-based* approach. Using an *event processing* approach, this paper re-structures the model in [2, 14] and incorporates the retransmission mechanism and lossy channel. The contribution of this paper is two-fold: (1) providing a formal specification of TCP Connection Management, and (2) providing some insights into the modelling process, which may be helpful for specifying complex protocols in general. The analysis of the TCP Connection Management CPN is addressed in [12] and in [2, 13, 14] for previous versions of the model.

4 Modelling Scope and Assumptions

We limit the scope of our model to Connection Management. Thus any parameters or segment fields associated with data transfer are not modelled. We consider five of TCP's user commands: *active open*, *passive open*, *send*, *close* and *abort*. Command *send* is considered since it changes the state of the TCP entity from LISTEN to SYN_SENT. We do not model the *receive* and *status* commands, as they are not concerned with connection management. We consider that security and precedence are always met and only consider a single connection between users. This allows us just to model the commands without their parameters.

We only model those fields in the TCP segment header and the state variables that are related to TCP connection management. A segment contains a sequence number, an acknowledgement number and the control bits: SYN, ACK, FIN and RST. Apart from the ACK bit, there can only be one control bit set in a segment.

In contrast with data transfer, TCP connection management only consumes a small portion of the sequence number space. Thus we can choose a small value of initial sequence number for

each TCP entity so that sequence numbers will not wrap during a connection. Therefore we don't need to implement modulo arithmetic. We also assume that the receive window is always big enough to accept incoming segments, since TCP Connection Management only consumes a small portion of the sequence numbers, and omit modelling the window field in segments and implementing checks associated with window size.

Table 1 describes the TCP states and variables that we model in this paper.

	Name	Description
State	CLOSED	Connection does not exist.
	LISTEN	Waiting for a connection request.
	SYN_SENT	Waiting for a matching connection response after having sent a connection request.
	SYN_RECEIVED	Waiting for a confirming acknowledgement after having both received a connection request and sent a response.
	ESTABLISHED	The connection is opened and data transfer can begin.
	FIN_WAIT_1	Waiting for a connection release request or its connection release request acknowledgement from the remote TCP.
	FIN_WAIT_2	Waiting for a connection release request from the remote TCP.
	CLOSE_WAIT	Waiting for a connection close command from the local user after receiving a connection release request from the remote TCP.
	CLOSING	Waiting for a connection release request acknowledgement from the remote TCP which responds to the release.
	LAST_ACK	Waiting for a connection close request acknowledgement from the remote TCP which initiates the release.
	TIME_WAIT	Waiting for 2MSL (maximum segment lifetime) to close.
State Variable	SND_NXT	Next sequence number to be sent
	SND_UNA	Oldest unacknowledged sequence number
	RCV_NXT	Next sequence number to be received
	ISS	Initial send sequence number

Table 1: TCP states and variables

In the life cycle of a connection, each TCP entity goes through a subset of the states listed in the table. State variable RCV_NXT is used to validate the sequence number of an incoming segment. Variables SND_UNA and SND_NXT are used to validate the acknowledgment number of an incoming segment. The ISS variable is the initial sequence number that each TCP entity selects for a connection.

Finally we assume that segments can be lost, delayed, and re-ordered while traversing the network.

5 TCP Connection Management CPN

As shown in Fig. 5, the CPN model contains a declarations page and 19 CPN pages. The 19 CPN pages are organised into a tree structure, which comprises 4 hierarchical levels.

The root of the "tree" is the TCP_Overview page, which is at the first level of the hierarchy and provides an abstract view of TCP and its environment. The second level contains the Event_Processing page that models TCP's responses to user commands, segment arrivals and retransmission timeout. The User_Commands page comprises three subpages: open, close and abort. The Segment_Processing page models the processing of segments for each of TCP's 11 states. The Retransmissions page models TCP retransmitting various segments. The CPN model contains 7 places, 19 substitution transitions and 95 executable transitions.

5.1 Overview Page

As shown in Fig. 6, the TCP_Overview page comprises 6 places, 2 substitution transitions and 2 executable transitions. Places User_1 and User_2 model TCP user commands. A token in a user

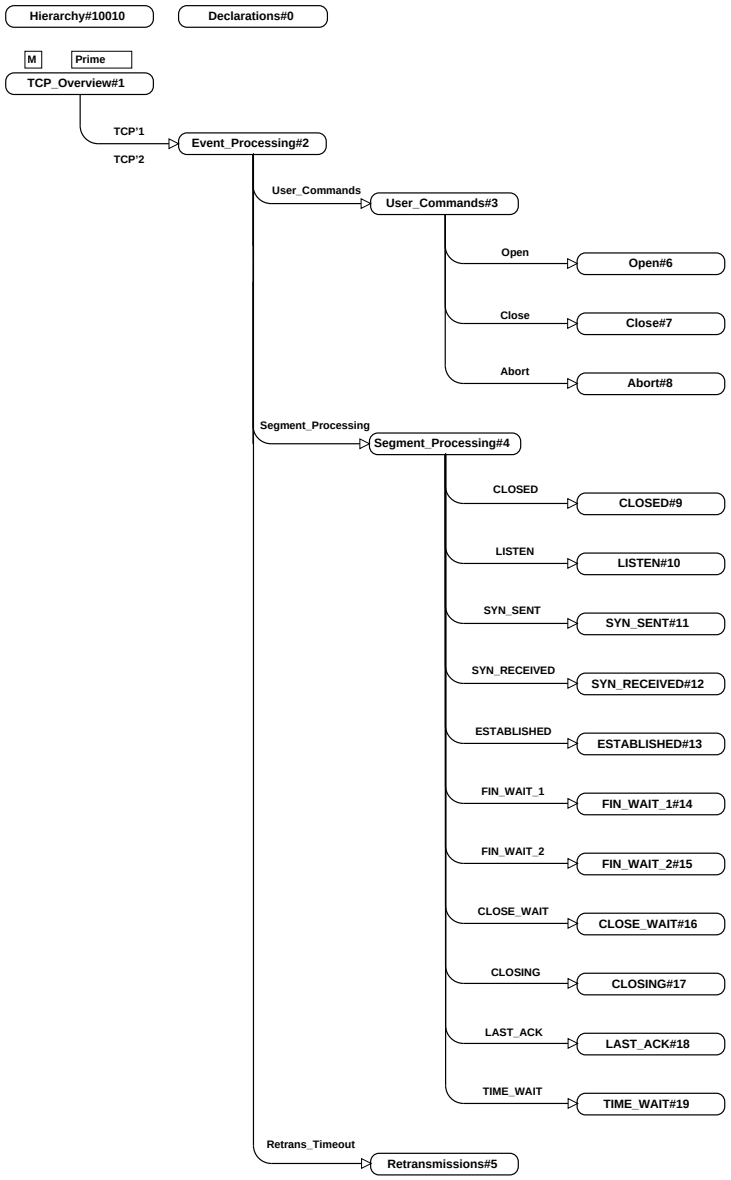


Figure 5: The Hierarchy page of TCP Connection Management CPN

place represents a command to be issued to the TCP entity. Place TCB, typed by colour set TCB, models the transmission control block that contains all the state variables for a connection. Places H1_H2 and H2_H1 model TCP buffers and all network storage (e.g., router buffers). H1_H2 indicates the data flow direction is from host 1 to host 2, whereas H2_H1 indicates data flow in the opposite direction. Transitions Lossy_Channel1 and Lossy_Channel2 can be switched on and off by their guards to model lossy and non-lossy channels respectively.

The TCP entities are modelled by two substitution transitions named TCP'1 and TCP'2. Places User_1 and User_2 are both assigned to place User in Figure 7. Place H1_H2 is assigned to place Out for TCP'1 and place In for TCP'2, and place H2_H1 is assigned to place In for TCP'1 and Out for TCP'2.

Listing 1 defines the declarations associated with the TCP_Overview page. The declarations are divided into four groups: (1) user commands, (2) TCP segments, (3) Transmission Control Block, and (4) initial send sequence numbers.

The first group has one colour set, `COMMAND` (line 2). It defines the type of places User_1 and User_2. The second group comprises lines 3 – 12. `CTLbit` (line 4) defines the four control bits SYN, ACK, FIN and RST in TCP headers. `ACKflag` (line 5) defines the status of the ACK bit, i.e, on or off. The introduction of the `ACKflag` facilitates checking the ACK status

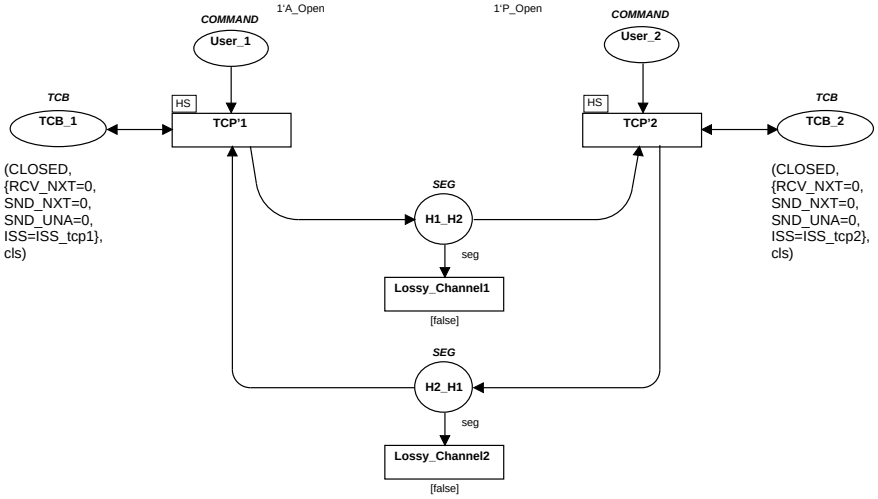


Figure 6: Top level CPN page: TCP_Overview

of an incoming segment, which is a step involved in TCP segment processing (see Section 3.9 of RFC 793). An alternative way of modelling the ACK bit is to combine it with the ACK field of a segment (line 10) by defining a union colour set, which comprises a string type and an integer type. If the ACK bit is off, then the ACK field is assigned string "null". Otherwise, it is assigned an integer. This then eliminates the need of defining `ACKflag`. This approach is more abstract than the use of the ACKflag which is aligned with the format of the TCP header, which contains an ACK bit field. `SEG_CTL` (line 6) is the product of the two colour sets `CTLbit` (line 4) and `ACKflag` (line 5). Int (line 7) defines the integer type. Places H1_H2 and H2_H1 are typed by `SEG` (line 8 – 11) which represents TCP segments. `SEG` is a record type that has three entries: SEQ, ACK and CTL. SEQ and ACK model the sequence and acknowledgement numbers respectively and each have the integer type Int. `CTL` (line 11) models the control information in the TCP header fields and it is typed by `SEG_CTL` (line 6). Finally, variable `seg` (line 12) represents any TCP segment in the channel and has the type `SEG`. An example of a TCP segment is $\{SEQ = 20, ACK = 11, CTL = (SYN, on)\}$. This segment is a SYNACK segment that has sequence number 20 and acknowledgement number 11.

The third group comprises lines 13 – 26. `STATE` (line 14 – 16) is an enumeration type that contains all the TCP states. `SV` (line 17 – 21) defines the four TCP variables `RCV_NXT`, `SND_NXT`, `SND_UNA` and `ISS` that are part of the `TCB`. Note that the ISS of a TCP entity is the basis for determining the initial values of its SND_NXT and SND_UNA for a connection and is also used to determine the initial value of the RCV_NXT of its peer TCP entity in a connection. `LISTENstat` (line 22) stores the history of the TCP state, that is, whether or not it has been in LISTEN. It contains the two values lis and cls, which indicates it has been in LISTEN or not respectively. This is used to determine the next state TCP enters from SYN_SENT or SYN_RCVD upon receiving a RST segment. `TCB` (line 23) is a product of colour sets `STATE`, `SV` and `LISTENstat`. Variable `s` (line 24) represents a TCP state. Variable `v` (line 25) represents the four-tuple of TCP state variables. An example of the value of such a variable is (21,11,10,10) where 21 is the receive next number (RCV_NXT), 11 is the send next number (SND_NXT), the first 10 is the send oldest unacknowledged number (SND_UNA) and the second 10 is the initial sequence number (ISS). Finally, variable `i` (line 26) is a variable that runs over `LISTENstat`.

The initial send sequence number for each TCP entity is represented by `ISS_tcp1` (line 28) and `ISS_tcp2` (line 29) respectively. We chose a small value of ISS for each TCP entity (i.e, 10 and 20), such that the sequence number space is within the range of `Int` and no sequence number wraps.

Listing 1: Colour sets for places on page TCP_Overview

```
1  (* User Commands *)
2  color COMMAND = with A_Open | P_Open | Close | Abort;
3  (* TCP Segments *)
4  color CTLbit = with SYN | RST | ACK | FIN;
5  color ACKflag = with on|off;
6  color SEG_CTL = product CTLbit*ACKflag;
7  color Int = int;
8  color SEG = record
9        SEQ: Int *
10       ACK: Int *
11       CTL: SEG_CTL;
12 var seg: SEG;
13 (* Transmission Control Block *)
14 color STATE = with CLOSED | LISTEN | SYN_SENT | SYN_RCVD | EST |
15                    CLOSE_WAIT | LAST_ACK | FIN_W1 | FIN_W2 |
16                    CLOSING | TIME_WAIT;
17 color SV = record
18       RCV_NXT: Int *
19       SND_NXT: Int *
20       SND_UNA: Int *
21       ISS:Int;
22 color LISTENstat = with lis|cls;
23 color TCB = product STATE*SV*LISTENstat;
24 var s: STATE;
25 var v: SV;
26 var i:LISTENstat;
27 (* Initial Sequence Numbers *)
28 val ISS_tcp1 = 10;
29 val ISS_tcp2 = 20;
```

5.2 Event Processing Page

The Event_Processing page (Fig. 7) contains 4 places and 3 substitution transitions. All the places are defined as *port places* and are linked with their corresponding socket places on the TCP_Overview page through *port assignment*, as already indicated.

Each substitution transition on the Event_Processing page is named by an event and is expanded onto the subpage bearing the same name at the third hierarchical level.

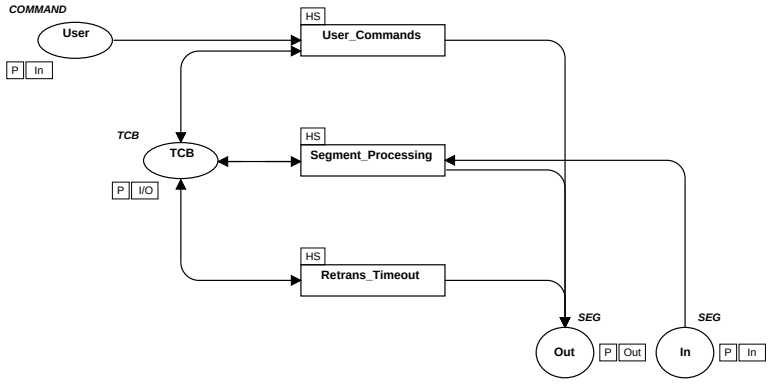


Figure 7: Second level CPN page: Event Processing

5.3 User Commands Processing

The processing of user calls *Open*, *Close* and *Abort* are modelled by the User Commands page (Fig. 8) and its three subpages: Open, Close and Abort. The processing of command *Send* is included in the Open page (Fig. 9). If the issuing of a command in a particular state results in an error message being returned to the user (which means that the operation is not allowed), then this operation is not modelled as it has no affect on TCP's functional correctness.

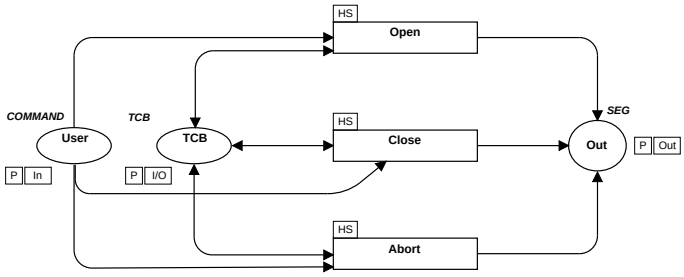


Figure 8: The User Commands page

On the Open page (Fig. 9), transition Passive_Open models the TCP server responding to the user command *passive open*. The transition is enabled if there is a token P_Open in place User and a token (CLOSED,v,i) in place TCB. When transition Passive_Open occurs, TCP changes state from CLOSED to LISTEN and variable i is set to *lis*, indicating TCP has been in LISTEN. Transition Active_Open models TCP's behaviour in response to an *active open* command, which results in a SYN being sent. Segments are modelled with ML functions and are described in Listing 2 of Section 5.4. Transition Send models TCP's behaviour in response to a *send* command, which results in a SYN being sent and TCP entering SYN_SENT from LISTEN.

The Close and Abort pages are modelled in a similar way.

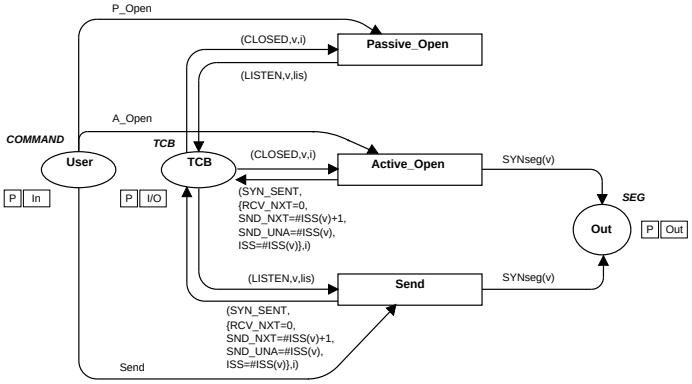


Figure 9: The Open page

5.4 Segment Processing Pages

The Segment Processing page is shown in Fig. 10. It has 11 substitution transitions, each of which is expanded onto a fourth-level subpage, describing the processing for each TCP state. We illustrate a fourth-level page using a very simple page, the LISTEN page, shown in Fig. 11.

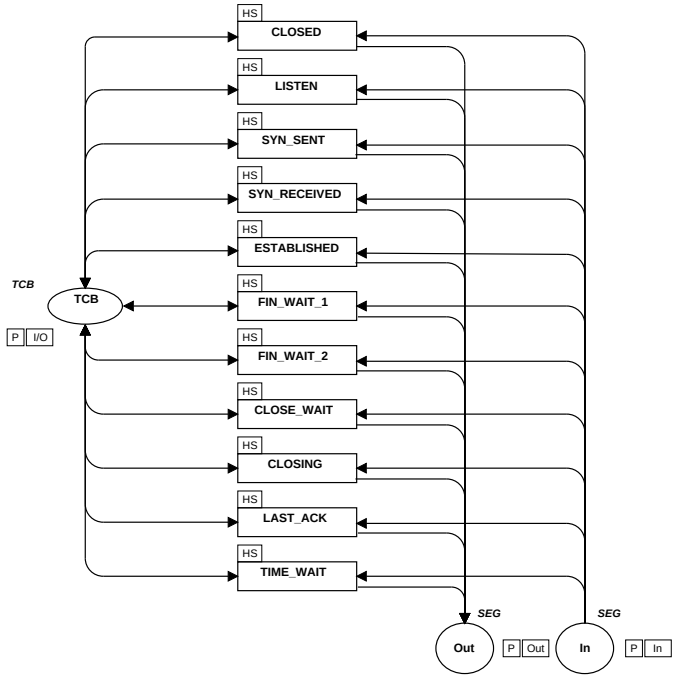


Figure 10: The Segment Processing page

Transition Rcv_ACK models TCP receiving an ACK segment which has the RST bit off and sending out a RST. TCP remains in state LISTEN with state variables unchanged. If the incoming segment contains a RST, TCP ignores it and remains in state LISTEN. This is modelled by transition Rcv_RST. Transition Rcv_SYN models TCP sending out a SYNACK segment upon receiving a SYN from the TCP client. The inscription of the arc from transition Rcv_SYN to place Out represents the SYNACK segment, which is modelled by function `SYNACKseg` that takes the record of the current state variables as its argument.

Listing 2 illustrates the ML functions that model TCP segments that have been mentioned so far. They appear as inscriptions on the arcs between transitions and the channel places In and Out. We model a segment using a function which returns a record rather than a tuple. An element of the record can then be referred to by a meaningful name rather than a number as in

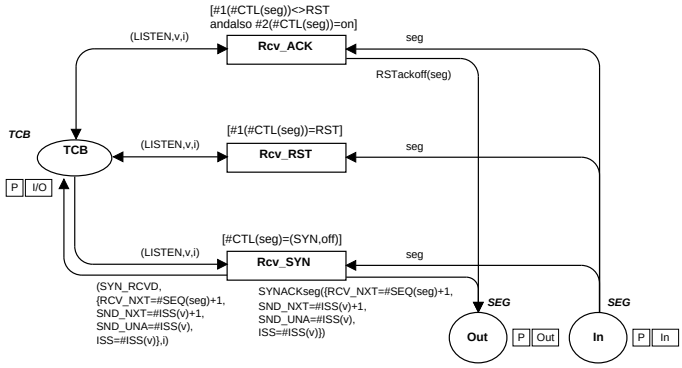


Figure 11: The LISTEN page

the case of a tuple. This makes the model more understandable. Gordon [11] adopts a similar approach in modelling the Wireless Transaction Protocol (WTP) [40].

Listing 2: ML functions for TCP segments

```
1  fun SYNseg(v: SV):SEG =
2      {SEQ = #ISS(v),
3       ACK = 0,
4       CTL =(SYN,off)};
5  fun SYNACKseg(v: SV):SEG =
6      {SEQ = #ISS(v),
7       ACK = #RCV_NXT(v),
8       CTL = (SYN,on)};
9  fun RSTackoff(seg: SEG):SEG =
10     {SEQ = #ACK(seg),
11      ACK = 0,
12      CTL = (RST,off)};
```

Function `SYNseg` (lines 1 – 4) models the SYN segment. It takes the current state variable `v` as argument and returns a value of type `SEG` (see Listing 1). The SYN's sequence number field, `SEQ`, is an initial sequence number, specified in ML as `#ISS(v)` (see Listing 1). As a SYN is the first segment sent to establish a connection, it does not have an acknowledgement number. We model this by assigning 0 to the acknowledgement number field, ACK. It is safe to do so because the control field, CTL, is assigned `(SYN,off)`, where off indicates that the number in the acknowledgement number field is invalid. The other segments are defined in a similar way.

5.5 The Retransmissions Page

The retransmission mechanism is modelled in Fig. 12. Place Retrans_Counter is a new place that models the number of retransmissions occurred in one of the five states: SYN_SENT, SYN_RCVD, FIN_WAIT_1, CLOSING and LAST_ACK. Each time a segment is retransmitted, the number in the retransmission counter is increased by 1.

Transition Timeout_Retrans models retransmitting a segment in one of the following states: SYN_SENT, SYN_RCVD, FIN_WAIT_1 and LAST_ACK, under the condition that the counter has not yet reached its maximum retransmissions value for that state. Transition Closing_Retrans models retransmitting a FIN in state CLOSING. It occurs when the TCP entity enters CLOS-

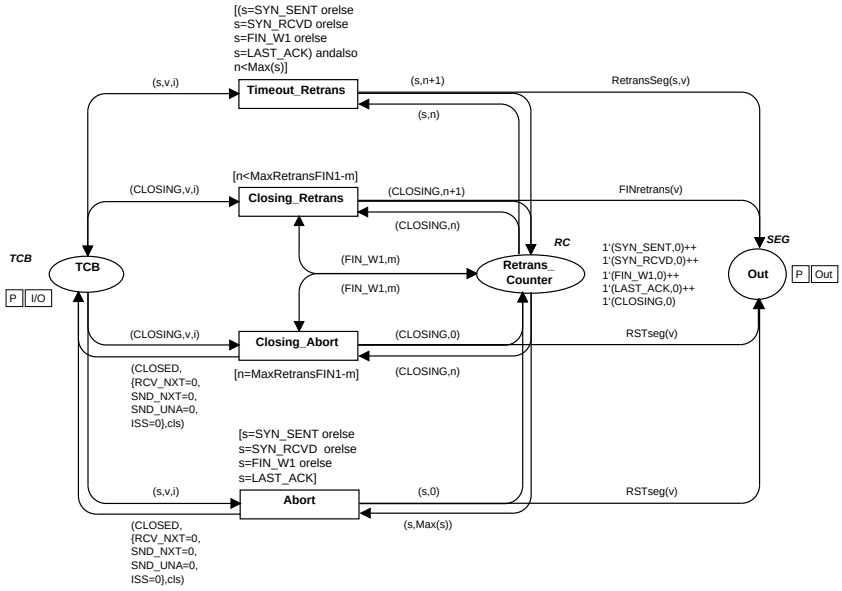


Figure 12: The Retransmissions page

ING and the number of retransmissions that occurred in CLOSING plus that occurred in FIN_WAIT_1 is less than the maximum retransmission value set on the FIN.

When the counter reaches its maximum retransmission value, the TCP entity aborts the connection, i.e., entering CLOSED and sending a RST. When the abort occurs, the number of retransmissions contained in the retransmission counter is reinitialised to 0. Transition Closing_Abort models TCP aborting the connection in CLOSING. Transition Abort models TCP aborting the connection in any of the four states, as specified in it guard.

The declarations of the CPN model are given in Listing 3.

Listing 3: Declarations for the Retransmissions page (Fig. 12)

```
color RS = subset STATE with [SYN_SENT,SYN_RCVD,FIN_W1,CLOSING,LAST_ACK];
color RC = product RS*Int;
var n:Int;
var m:Int;
val MaxRetransSYN = 2;
val MaxRetransSYNACK = 2;
val MaxRetransFIN1 = 2;
val MaxRetransFIN2 = 2;
fun Max(s:STATE) = case s
                     of SYN_SENT => MaxRetransSYN
                      | SYN_RCVD => MaxRetransSYNACK
                      | FIN_W1 => MaxRetransFIN1
                      | LAST_ACK => MaxRetransFIN2;
fun RetransSeg(s:STATE,v:SV) =
    if s = SYN_SENT then SYNseg(v)
    else if s = SYN_RCVD then SYNACKseg(v)
    else FINretrans(v);
```

Colour set RS (line 1) defines a subset of STATE, which includes the five states in which the retransmissions occur. Colour set RC (Line 2) is the type of place Retrans_Counter and it is a product of RS and Int that associates the number of retransmissions with a state. Variable n (line 3) represents the number of retransmissions occurred in a particular state. Variable m (line 4) models the number of retransmissions of a FIN occurred in state FIN_WAIT_1 while the TCP entity enters CLOSING. The maximum number of retransmissions for various segments are given in Lines 5 – 8. The official TCP specification [3, 28] does not mention the maximum number of retransmissions for SYN, SYNACK and FIN. We set the number to two for SYN and SYNACK according to Wright and Stevens [39] and the same for FIN. Note that the retransmission mechanism can be disabled by setting the maximum number of retransmissions to a value less than 0 (e.g., -1) so that transitions Timeout_Retrans and Closing_Retrans in Fig. 12 will not be enabled. Function Max() (Lines 9 – 13) takes a state as argument and returns the maximum number of retransmissions that can occur for a particular segment in that state. Finally, function RetransSeg() (Lines 14 – 17) models a segment retransmitted. It takes a state and a set of state variables as argument and returns a segment that is retransmitted in that state.

6 Discussion of Modelling Approaches

The process of modelling complex protocols such as TCP is non trivial. The first version of the model is published in [13] and then it has gone through numerous revisions and several major changes including being restructured. There can be more than one modelling approach with pros and cons existing at different stages of the process. Therefore it is necessary to weigh the advantages and disadvantages of an approach and choose the most appropriate one. However, this is only half the story. When we use one approach, we may not be aware of its potential disadvantages that can surface and cause trouble at a later stage of the modelling process. It becomes of paramount importance to discover and identify new approaches to overcome problems that occur at later stages in the modelling process, especially when it is expensive to switch to another approach. This section discusses the possible approaches that can be used during the modelling stage of TCP connection management, gives the reason behind choosing an approach, and illustrates how to overcome the drawbacks of an approach with concrete examples.

We discuss the modelling approaches in the context of TCP connection management. Nonetheless, these approaches may be useful for modelling complex protocols in general.

6.1 Modular Organisation

The idea of modular organisation is to break down a CPN model into separate modules, each having a specific functionality. Modularity reduces the complexity of specifying a protocol or system. One way to define a module's functionality is according to the phase in which the protocol is operating. Because TCP has three phases, namely, connection establishment, data transfer and connection release, the CPN model of TCP can be decomposed into three modules, each corresponding to one phase. This approach is intuitive and it is adopted in [22, 32] and our early work [13]. However, we discovered that it is difficult to check whether the CPN model is a faithful reflection of RFC 793 [28] with this approach, because Section 3.9 of RFC793 specifies the protocol on an *event processing* basis. That is, actions are taken by TCP entities in response to user commands, segment arrivals and internal timeouts. Using the phase based approach, it becomes more difficult to conduct consistency checks when the protocol is modelled in greater detail.

Another modelling approach we have used is the *state-based* approach. We use one CPN page to model TCP's behaviour for one state, for example, the processing of a user command and/or an incoming segment. This approach is close to the way TCP is specified using its state diagram and is easier to read and check for consistency between the model and the specification.

Many ITU-T Recommendations (e.g., X.25 [4]) and the WAP Forum [38] adopt the state-based approach as a standard way of specifying protocols. Gordon [11] uses the state-based approach in modelling the Wireless Transaction Protocol (WTP) [40]. However the disadvantage of the state-based approach is that the event processing common to a set of states will be replicated in the model, resulting in redundancy.

A way to overcome this problem is to fold the transitions whose occurrences are triggered by similar inputs (e.g., segment arrivals) and yield similar outputs (e.g., sending segments). In the case of TCP connection management, we achieved a significant reduction (about two thirds) in the number of transitions. However, the folding of transitions inevitably breaks the easy-to-read structure following the state-based approach and makes the model difficult to understand.

To make a trade-off between redundancy and readability, we choose to fold the transitions that are involved in TCP's processing of user commands but adopt the state-based approach for TCP's processing of incoming segments. This approach is demonstrated by the TCP Connection Management CPN in Section 5.

6.2 Hierarchical Construction

Modules are often organised in a hierarchy, which conveys a top-down design philosophy. Design/CPN supports this by allowing users to create substitution transitions that are *macros*, each of which represents a piece of net structure (i.e., module). In Fig. 5, the TCP_Overview page is called a superpage, whereas the Event_Processing page is a subpage. When two substitution transitions are expanded onto the same subpage, the subpage is then instantiated to replace each of the substitution transitions at compile time.

A model that has both superpages and subpages is a hierarchical net. Each hierarchical net has an equivalent flattened net, which can be obtained by replacing each substitution transition by its subpage. Duplicate net structures will appear when a subpage is used as a page instance. This can be seen by observing the structure of the model. For instance, in Fig. 5, we see TCP'1 and TCP'2 as inscriptions of the arc from the TCP_Overview page to the Event_Processing page. This means that the net structure on Event_Processing will appear twice when the model is compiled. It is important to check the hierarchy page of the CPN to ensure that the duplicate net structures are indeed needed. To explain this, we give an example. There are four states where TCP retransmissions can occur: SYN_SENT, SYN_RCVD, FIN_WAIT_1 and LAST_ACK. A modelling mistake that can be easily made is associating the Retransmissions page with each of the four state pages as their subpages. This results in unnecessary duplicate net structures and is in fact a modelling error.

6.3 Modelling Identical Protocol Entities

The state machine of each TCP entity is identical. Hence we only need to model one of them, which can then be instantiated twice to represent two TCP entities. This is done by using the page instance technique discussed above. As the CPN model contains over 90 executable transitions, using the page instance technique significantly reduces the number of transitions that would be needed with an ordinary approach. Correspondingly, it reduces the time of syntax checking, the time of switching from the editor to the simulator and to the occurrence graph analyser, which is used over and over again when conducting analysis. Moreover, it saves a lot of time and effort in debugging and maintaining the model. However, as we shall discuss in Section 6.4, this approach has difficulty modelling different Initial Send Sequence (ISS) numbers. We devise new methods to overcome this problem. Given that we believe its advantages outweigh its disadvantages, we adopt the page instance technique throughout our modelling process.

6.4 Modelling Initial Sequence Numbers

The TCP entity at both sides of the connection has its own ISS. It is selected by the TCP client after the client receives an *active open* command from its user, and it is selected by the TCP server when the server is in state LISTEN and receives a SYN segment from the client. In a simultaneous open situation, each TCP entity chooses its ISS on receiving an *active open* command from its user.

If we use the page instance technique to create one CPN that can be instantiated, the ISS will be identical for each side. To accommodate the more general case, i.e., modelling the TCP entities with different ISS values, we create two extra TCB places on the TCP_Overview page (Fig. 6), one for each side, and preset the value of each ISS. By changing the preset value of ISS, we can model different ISSs for each side as well as identical ISSs. We call this method *static ISS assignment*. It has been used in previous versions of the model [13, 14].

Another way of overcoming the identical ISSs problem is *dynamic ISS assignment*, which eliminates the need to create the two extra places. The ISS entry of the initial marking of place TCB on the Event_Processing page (Fig. 7) is assigned a random number function that generates a random integer each time it is evaluated. When the Event_Processing page is instantiated twice and the random number function is evaluated, each TCP entity has its own ISS, which may or may not be the same because it is chosen arbitrarily. However, due to the randomness of the initial sequence number, we find it is difficult to examine the state space, which may be different each time it is generated. This is the drawback of using a random number function.

7 Conclusions

In this paper, we have presented a formal model of TCP connection management at a significant level of detail. The model precisely captures TCP behaviour as defined in RFCs 793 and 1122. TCP segments are represented by functions that contain information including sequence numbers, acknowledgement numbers and 4 control bits; and the transmission control block is modelled as a triple comprising TCP state, a set of state variables and a listen status. The CPN model also takes into account retransmissions and lossy channel.

Due to the protocol's inherent complexity, a good structure is the key to the model's readability. We have illustrated the use of hierarchical CPNs to structure the model, taking advantage of symmetry, to just specify the connection management procedures once, but call them for each TCP entity by using page instances. This reduces the complexity of the model and eases maintenance. The detailed part of the model is structured on an *event processing* basis, that is, according to TCP's response to user commands, segment arrivals, and internal timeouts. The state-based approach is adopted in modelling TCP's behaviour in response to incoming segments. The advantage of this approach is good readability and ease of validating the model against the TCP specification. We also discussed the modelling decisions regarding modelling identical protocol entities and initial sequence numbers.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Request for Comments 2581, IETF, April 1999.

[2] J. Billington, G.Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 210–290. Springer-Verlag, 2004.

[3] R. Braden. Requirements for Internet Host — Communication Layers. RFC 1122, IETF, October 1989.

[4] CCITT. Interface between DTE and DCE for Terminals Operating in the Packet Mode on Public Data Networks, 1977. ITU Recommendation X.25.

[5] V. G. Cerf. Specification of TCP Internet Transmission Control Program, TCP (Version 2), March 1977. available from DARPA/IPTO.

[6] V. G. Cerf, Y. K. Dalal, and C. A. Sunshine. Specification of Internet Transmission Control Program. INWG Note 72, December 1974.

[7] V. G. Cerf and J. B. Postel. Specification of Internet Transmission Control Program, TCP (Version 3), January 1978. available from USC/ISI.

[8] D. E. Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, volume 1. Prentice Hall, Upper Saddle River, NJ, 4th edition, 2000.

[9] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649, IETF, 2003.

[10] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, IETF, April 1999.

[11] S. Gordon. Verification of the WAP Transaction Layer using Coloured Petri Nets. PhD Thesis, University of South Australia, Australia, November 2001.

[12] B. Han. Formal Specification of the TCP Service and Verification of TCP Connection Management. Draft PhD Thesis, University of South Australia, Australia, August 2004.

[13] B. Han and J. Billington. An Analysis of TCP Connection Management Using Coloured Petri nets. In *Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'2001)*, pages 590–595, Orlando, Florida, July 2001.

[14] B. Han and J. Billington. Validating TCP Connection Management. In *Proceedings of the Workshop on Software Engineering and Formal Methods, Adelaide, Australia*, volume 12 of *Conferences in Research and Practice in Information Technology*, pages 47–55, June 2002.

[15] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, IETF, May 1992.

[16] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, Berlin, 1997.

[17] J. F. Kurose and K. W. Ross. *Computer Networking - a Top-Down Approach Featuring the Internet.* Addison-Wesley, U.S.A, 2nd edition, 2003.

[18] J. F. Kurose and Y. Yemini. The Specification and Verification of a Connection Establishment Protocol Using Temporal Logic. *Protocol Specification, Testing, and Verification*, pages 43–62, 1982.

[19] H. P. Lin. Modelling a Transport Layer Protocol using First-order Logic. In *Proc. of the ACM SIGCOMM Conference on Communications Architecture and Protocols*, pages 92–100, September 1986.

[20] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., 1996.

[21] J. Martin, A. Nilsson, and I. Rhee. Delay Based Congestion Avoidance for TCP. *IEEE/ACM Transactions on Networking*, 11(3):356–369, 2003.

[22] H. Mehrpour and A. E. Karbouiak. Modelling and Analysis of DOD TCP/IP Protocol Using Numerical Petri Nets. In *Proc. IEEE Region 10 Conf. on Computer Communication Systems*, pages 617–622, Hong Kong, September 1990.

[23] S. L. Murphy. Service Specification and Protocol Construction for a Layered Architecture. PhD Thesis, University of Maryland, USA, May 1990.

[24] S. L. Murphy and A. U. Shankar. Connection Management for the Transport Layer: Service Specification and Protocol Verification. *IEEE Transactions on Communications*, 39(12):1762–1775, December 1991.

[25] V. Paxson. Known TCP Implementation Problems. RFC 2525, IETF, March 1999.

[26] J. Postel. Transmission Control Protocol Version 4, February 1979.

[27] J. Postel. DoD Standard Transmission Control Protocol. RFC 761, IETF, January 1980.

[28] J. Postel. Transmission Control Protocol. RFC 793, IETF, September 1981.

[29] D. Schwabe. Formal Specification and Verification of a Connection Establishment Protocol. In *Proc. of the Seventh Symposium on Data Communications*, pages 11–26, New York, NY, USA, 1981. ACM Press.

[30] D. Schwabe. Formal Techniques for Specification and Verification of Protocols. PhD Thesis, Report CSD 810401, Computer Science Department, University of California at Los Angeles, 1981.

[31] M. A. Smith. Formal Verification of Communication Protocols. *Formal Description Techniques IX: Theory, Applications and Tools*, pages 129–144, October 1996.

[32] M. A. Smith. Formal Verification of TCP and T/TCP. PhD Thesis, M.I.T., USA, September 1997.

[33] W. Stallings. *Department of Defense (DOD) Protocol Standards*, volume 3. Howard W. Sams & Company, 1978.

[34] C. A. Sunshine and Y. K. Dalal. Connection Management in Transport Protocols. *Computer Networks*, 2(6):454–473, December 1978.

[35] F. J. W. Symons. Modelling and Analysis of Communication Protocols Using Numerical Petri Nets. PhD Thesis, University of Essex, May 1978.

[36] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[37] R. S. Tomlinson. Selecting Sequence Numbers. In *Proc. of ACM SIGCOM/SIGOPS Interprocess Communications Workshop*, pages 11–23, Santa Monica, California, March 1975.

[38] WAP Forum. Wireless Application Protocol Architecture Specification. Web site: `http://www.wapforum.org`.

[39] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Vol.2 : The Implementation*. Addison-Wesley, Reading, MA, 1995.

[40] WAP Forum. Wireless Application Protocol Wireless Transaction Protocol Specification, June 2000. Web site: http://www.wapforum.org.

Application of Coloured Petri Nets in Systems Biology

Thomas Runge[1,2]

[1]Brandenburg University of Technology Cottbus, Department of Computer Science,
chair of data structures and software dependability
Postbox 10 13 44, 03013 Cottbus, Germany
thomas.runge@informatik.tu-cottbus.de
phone: +49 355 69 3885
fax: +49 355 69 3830

[2]Technical University of Applied Sciences Berlin, Department of Bioinformatics,
Seestrasse 64, 13347 Berlin, Germany

September 2004

Abstract

Computer aided analysis is necessary to improve the understanding of the complex biochemical processes. The often used kinetic models in biochemistry are based on differential equations. The results of such a kinetic model are often non-reliable on account of unreliable data or of inconsistencies in the used model. Therefore, other supplementary methods are indispensable. A detailed qualitative analysis should be done, before a quantitative (kinetic) analysis is made. This paper extends and refines the construction strategy of a coloured Petri net model of a metabolic network at a steady state, previously introduced in [11]. In a steady state the internal compounds' concentrations are constant and therefore bounded. Each chemical reaction is typically active at steady state. Therefore, its naturally to demand a bounded and live model. A systematic procedure, exploiting the place transition nets' T-invariants to construct an behavioural equivalent bounded and live coloured net, is presented. The application of P-invariant or model checking analysis techniques of such a model results in usable information, which are helpful for model validation.

Keywords:coloured Petri net, T-invariant, P-invariant, metabolic pathway, glycolysis pathway

1 Introduction

Due to the rapidly growing amount of biologically experimental possibilities and the related amount of created experimental data, it is mandatory to transmit data in simple, analysable, and possibly validated models. Therefore, bioscientists need practicable, theoretically well-founded methods to construct, prove, analyse, and simulate a model, which is based on experimental data.

Today, there exist many quantitative models, which typically employ differential equations. Such models need kinetic parameters to describe and analyse a biochemical system. A restriction on hand of these models is often the imperfect and imprecise knowledge of the kinetic parameters, because up to now it is difficult to observe the processes in a cell at molecular level in vivo. Contrary, a qualitative analysis offers the possibilities of a structural analysis based only on information of the simple, atomic chemical reactions and their stoichiometric parameters. A qualitative analysis can be used for an intermediate validation of the given model structure.

Many different approaches for qualitative or quantitative analysis methods have been developed. For example, a graph theoretical approach is described in [7]. A mathematical approach is introduced in [22], computing a set of generating vectors that describe the conical steady-state solution space for flux distributions in a metabolic network, the so-called extreme pathways. But only Petri nets have been applied for both kinds of analysis. For example, quantitative Petri net models are introduced in [8] and [6], qualitative Petri net models are described in [18], [12], [10] and [15], using place/transition nets, and in [11] and [21], using coloured Petri nets.

In the latter papers the feasibility is examined to construct a coloured Petri net model of the glycolysis and the pentose-phosphate pathways in an erythrocyte cell. A deep understanding of the given network is used for a construction by hand of an environment for the modelled reaction chains. An experimental software package, called SY written by H. Genrich [4], was used to improve and validate the model stepwise. The work done by Voss et al. and Genrich was the starting point for the current paper. An extended, more general model of the glycolysis serves as case study. Starting from unbounded and live place/transition net a behavioural equivalent bounded and live coloured Petri net will be constructed.

The contributions of this paper are the following: (1) The basis of modelling relative reaction rates, which are necessary to get a bounded system model, is the dynamic conflict avoidance principle. Two systematic, but different dynamic conflict avoidance principles are introduced to get a bounded and not live coloured Petri net, the so-called core model. (2) The core model is extended to a system model, which contains additionally an environment behaviour. An automatic way to construct a similar environment as the environment, constructed by Voss et al. by hand, is introduced, exploiting the place/transition nets' T-invariants to construct a behavioural equivalent bounded and live coloured net, the so-called system model.

It is assumed that the reader is familiar with place/transition (P/T) nets and coloured Petri nets (CPN). Otherwise, related literature is recommended [20], [13], and [14].

The paper is organized as follows. In the next section, the essential biochemical and Petri net terms and concepts are recalled. In section three the motivation to combine place/transition nets and coloured Petri nets is described. The used case study - an extended glycolysis - is introduced in section four. Section five presents two modelling strategies to get a coloured Petri net core model and compares them shortly. Additionally, some possible problems during the construction are explained. Section six introduces an automatic algorithm to compute an environment to extend the coloured Petri net core model and reach the aim of a bounded and live model. Finally, conclusions are given in section nine.

2 Biochemical and Petri Net Prerequisites

Metabolic networks are one of the main types of molecular biological networks. The term *metabolism* refers to the processes, which acquire and utilize energy (e.g. in form of ADP and ATP) and small building units (e.g. R5P). In general, a metabolic network consists of many interconnected atomic reactions. An atomic chemical reaction is described by its input compounds (also called educts), its output compounds, and the stoichiometric relations between them. A *metabolic pathway* is defined by its set of involved reactions and its input and output compounds. The educts, the intermediates, and the products are called *metabolites*. According to the applied abstraction level, different specifications of an atomic reaction can be used. An observed metabolic pathway is characterized by a set of *external* and *internal* metabolites. An external metabolite is a substance, which can be supplied and removed to/from the model/pathway. A suppliable external metabolite is called *source* and a removable external metabolite is called *sink*. All other metabolites are internal and can only be transformed into another internal or external metabolite(s). The so-called *ubiquitous* molecules are an exception. Those are the small molecules like H_2O, NADH, ADP, and CO_2 found in sufficiently large amounts in all organisms. These metabolites can be treated as external or as internal metabolites, depending on the desired environment behaviour. For ease of distinction, Voss et al. named the remaining substances *primary* [21].

The regulation of the reaction rates of a reaction is controlled by one or more enzymes. In a qualitative model it is assumed that the system is in a *steady state*. A steady state is a special system state, in which all internal substance' concentrations are constant. The total production rate of each internal metabolite is equal to its total consumption rate at a steady state. Therefore, the enzyme regulation is not explicit considered in a qualitative analysis. For this reason, only the set of atomic reactions with their stoichiometric parameter are necessary to construct an analysable qualitative model and to perform a qualitative analysis.

Reaction Types

There exist three types of chemical reactions. The three types result in three different model components of an atomic reaction, shown in the glycolysis model later.

The classical, not reversible chemical reaction is named as *irreversible reaction*.

Two reactions are hidden behind the so-called *reversible reaction*. They are two complementary reactions catalyzed by the same enzymes, but often located in different compartments of a cell. The point is that, if they are in the same compartment, only one of them is thermodynamically preferred on account of the

surrounding irreversible reactions. *Equilibrium reactions* are similar to reversible reactions. The difference is that both reactions may be active at the same time and at the same location of a cell.

These three reaction types are important for a correct quantitative analysis. The equilibrium and reversible reactions have different effects on the dynamic behaviour of the model. The behaviour of a reversible reaction is mostly similar to the behaviour of an irreversible reaction. But in biochemical context both reactions are often not distinguished. A reason may be the assumption that an equilibrium reaction can also be considered as irreversible reaction, if it is enclosed by irreversible reactions.

How to represent Metabolic Networks with Petri nets?

Metabolites are modelled as places and by convention the primary metabolites are represented by a larger place and the ubiquitous molecules with a smaller one. Chemical reactions are modelled as transitions and the stoichiometric relations as weighted arcs between places and transitions. The token of a net represents a unit of the corresponding metabolite of the given place. Reddy et al., Koch et al., and Heiner et al. have applied these simple transformation rules on biochemical systems to get P/T nets.

Some Petri Net Properties and their Biochemical Interpretation

The invariant analysis of Petri nets plays a special role in biochemical context. Only positive invariants are considered. The positive *place-invariant* (P-invariant) is defined by the integer solution vectors y (place vector) of the equation $y \cdot C = 0, y > 0$, whereby C is the (PxT) incidence matrix. The symbol $>$ means that no component of the vector is smaller than zero and at least one component of the vector is greater than zero. In biochemical context the mass conservation law is represented by a P-invariant. The mass conservation law states that the mass of an isolated system will always remain constant, regardless of the processes acting inside the system. The amount of tokens on a place states for the mass of the corresponding metabolite.

Today, great attention in biochemistry lies on the minimal positive *transition-invariants* (T-invariant) in the Petri net theory, elementary modes in biochemistry, respectively. T-invariants are defined by the positive integer solution vectors x (transition vector) of the equation $C \cdot x = 0, x > 0$. The P- resp. T-invariant z is called *minimal*, if there exists no P- resp. T-invariant $w = 0$ with $supp(w) \subset supp(z)$, whereby $supp(x)$ (read as support of x) describes the set of non-zero components in x, and the largest common divisor of all components of x is equal to one. A T-invariant gives structural insights in the represented pathway. The following definition of minimal biochemical pathways corresponds to the minimal T-invariant of Petri nets.

Elementary modes have been defined as the minimal set of enzymes that could operate at steady state. These modes can be calculated using the convex analysis with special conditions. A tool, which is able to calculate elementary modes, is METATOOL, which is described by Pfeiffer et al. in [16].

In the following, only minimal T-invariants are considered. In metabolic Petri net models T-invariants (elementary modes) can be classified into two groups and for each group two types.

- *Trivial T-invariant*
  - **Environment T-invariant**
    If the environment strategy of type I is used, explained below, a trivial T-invariant exists for each pair of supply and removal transitions of a ubiquitous compound.
  - **Reaction T-invariant**
    A trivial T-invariant exists for each equilibrium reaction. These invariants are internal cycles. In a place/transition nets, a trivial T-invariant exists also for a reversible reaction.
- *Non-Trivial T-invariant*
  - **IO T-invariant**
    An IO T-invariant describes the exchange fluxes from one or more primary source metabolites to one or more primary sink metabolites.
  - **Internal T-invariant**
    An internal T-invariant represents an internal cycle within the modelled system. Reaction T-invariants are special internal T-Invariants.

The following additionally introduced notions are essential for this paper. A *marking* of a Petri net assigns a multi-set of tokens to each place. Each marking represents a system state of the net. A marking, which is reachable from the initial marking, is called *dead marking*, if no transition is enabled.

Informally, a *structural conflict* is present, if at least two transitions exist, which have at least one common pre-place. A *dynamic conflict* is present, if a marking, reachable from the initial marking that realizes the structural conflict, exist. A *critical dynamic conflict* is informally defined as a dynamic conflict, whereby at least one alternative solution can be result in a dead marking. An example is shown in figure 2.

3 Modelling Aspects

A direct modelling of the given set of reactions yields normally a bounded, not reversible and not live Petri net. It is called **core model**. The core model is place-bordered, because each pathway starts and ends with a set of metabolites. This model is not sufficient for a detailed analysis. So, an extension of the core model by an environment is necessary. The extended model is called **system model**.

With the knowledge about sources and sinks it is possible to extend the core model by an environment. Such an environment is only used to increase the set of useable results by applying of Petri net analysis possibilities, for example reachability graph/occurrence graph analysis and/or model checking techniques. Two useful types of environments are introduced.

Environment Type I

The simplest model of an environment is a transition bordered Petri net with an empty marking. This means that for each primary source a pre-transition without pre-places and for each primary sink a post-transition without post-places are added. It is assumed that each ubiquitous compound is a source and a sink. Reproducing the empty marking of the core model is the reason for this assumption, which is equal to the assumption that all supplied molecules must be, possibly in another form, be removed. An unbounded and possibly live Petri net model is the result of this modelling. The liveness property depends on the source/sink specification of the primary metabolites and the modelled pathways. If no diseases are modelled, which is currently done, then the net must be live for the empty initial marking, otherwise the selected reactions and source/sink specifications are unfavourably chosen. For example, no molecule can be transformed into another without a foregoing supply of them.

The resulting model can be analysed by calculation of T-invariants, whereas the T-invariants can be classified by four types, which were previously described. Each calculated EA-T-invariant is expected to reproduce the empty marking. No useable result is produced by applying of P-invariant and other extensive analysis methods (e. g. model checking), because the resulting model is always unbounded. Some isolated case studies can be found in [12], [10] and [19]. To fill the lack of P-invariant analysis the second type of environment was developed.

Environment Type II

On account of the inspection of metabolic networks at a steady state the intermediates' concentrations are constant and therefore bounded. For this reason the aim of the second environment type is to get a bounded and live Petri net model, which is behavioural equivalent to the model with environment type I. The net with environment type II is behavioural equivalent to a P/T net with environment I of the same considered metabolic network, if the T-invariants/elementary modes (without border transitions) of the P/T net are also included in the coloured net, but on account of the environment type possibly in a summarized form. To reach the aim of a bounded and live model without an explicit enzyme control, it is necessary to limit the amount of supplied metabolites and to include relative reaction rates in an arbitrary way in the model. The relative reaction rates are used to avoid dead markings, which can arise through the restriction of the amount of supplied source molecules and through an unfavourable solved dynamic conflict. An example is given in figure 2. To avoid dynamic conflicts in a compact description, coloured Petri nets are used for this case study.

The environment is used to conserve the steady state. Unfortunately, through this type of environment it is possible that only one minimal T-invariant, a summarized version of all IO-T-invariants of the system model with environment type I, exists within a coloured Petri net.

The first attempt by using coloured Petri nets was made by Voss et al. [21]. The construction of the environment was made stepwise and by hand with much knowledge about the modelled system. In this paper an automated calculation of the environment of type II is shown by using knowledge from a P/T net system model with environment type I.

The resulting model can now be analysed by P-invariant, model checking or other analysis techniques that require a finite state space or the boundness property. The results of the analysis are useful for model validation and naturally these results increase the knowledge about the considered system.

Combination of Place/Transition Nets and Coloured Petri Nets

In this paper a combination of analysis, simulation and modelling techniques of coloured and P/T Petri nets is used to get a new coloured model, which fulfils the requirements of quantitative and qualitative analysis. Only a small subset of the coloured net possibilities, provided by Design/CPN [1], is necessary for a qualitative modelling and analysis of a metabolic network. Only properties or features of coloured Petri nets, especially of Design/CPN, are used, which enables an unfolding to a P/T net.

The place transition nets' T-invariants are exploited to construct a behavioural equivalent bounded and live coloured net. The behavioural equivalence has no relation to an (un-) folding process.

It is well known that an equivalence relation ((un-) folding) exists between coloured and P/T nets. On account of the equivalence of the P/T nets and coloured Petri nets, each of the described environment types can be expressed with both net classes. But the combination of the advantages of both classes makes a modelling and validation easier.

The tool Design/CPN [1] is used to construct the coloured Petri nets and the tool PED [3] is used to construct the P/T nets.

Useful Properties of Both Net Classes

An advantage of the P/T nets is the possibility to calculate invariants in a simple way. INA, which can be found in [2], is such a calculation tool.

Compactness is one of the great advantages of coloured Petri nets. Compactness means that a smaller net for the same content as for P/T nets is reachable, if coloured Petri nets are used. Especially, for the later presented biochemical models it gets a clearer model. For example, after an unfolding of the coloured net, additional transitions or places in a P/T model may exist for each token colour. For this reason a coloured Petri net is longer human readable as a P/T net by increasing the net size.

The possibility to execute code, which is able to modify the state of the net, or a time concept are examples of some extensions of coloured Petri nets, provided by Design/CPN. These possibilities will not be explained in detail now, but these are the reasons why a coloured net can be used for quantitative analysis and simulation. It is a great advantage, if the fundamental model/data structure must not be changed for quantitative and qualitative analysis.

Another disadvantage of P/T nets are shown in [19]. It was shown that for P/T nets not every possible P-invariant is biochemically interpretable by using the direct mapping from the reaction formulas to the P/T net. Each P-invariant of a biochemical coloured Petri net model must be biochemically interpretable, if additional knowledge about metabolite conservations are used during the modelling process. The modelling of such knowledge is very easy and without adding new places or transitions realizable in contrast to a P/T net. For example, the conservation of the ADP-part of ATP can be very easy expressed by a special arc inscription to/from a ubiquitous compound. The figure 1 shows an example reaction with such conservation knowledge. The constant value P and the other arc inscriptions are used to distinguish the transport of molecules. Hence, a P-invariant for ATP and ADP should be exists by using this additional knowledge about conservations.

Until now, a disadvantage of coloured Petri nets is the absence of some analysis tools, which are able to calculate P- or T- invariants. First attempts to verify P- and T-invariants of coloured Petri nets was made by Genrich, with an experimental software package (called SY), and Voss et al. [21]. It should be noticed that only a verification of an expected T- or P-invariant could be realized. No calculation of a T- or P-invariant is possible with SY.

It should be noticed that a T-invariant of a P/T net contains only an amount of occurrences of transitions. A T-invariant of a coloured net contains additionally information about binding elements (a concrete variable - value assignment) for a transition. To distinguish between them, a T-invariant (P/T) stands for a T-invariant in a P/T net and a T-invariant (CPN) stands for a T-invariant in a coloured net.

The construction of a bounded and live coloured Petri net using two different methods will be demonstrated in the next sections.

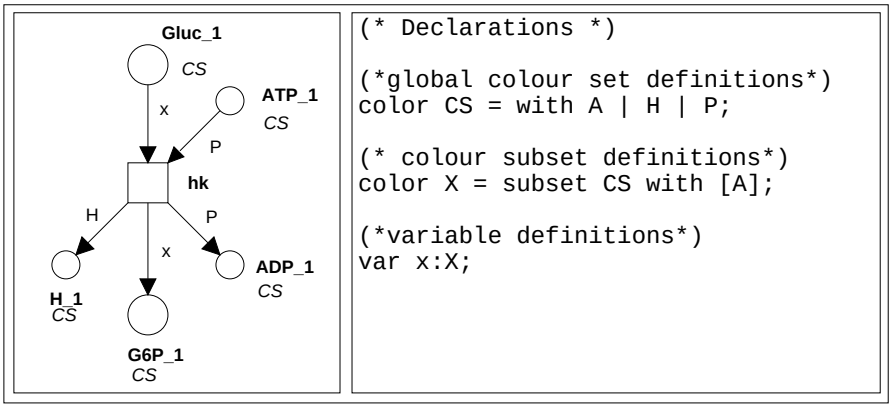


Figure 1: An irreversible Reaction expressed by a CPN
reaction hk: $Gluc + ADP \rightarrow G6P + ATP + H$

4 Case Study Glycolysis

The glycolysis is one of the main metabolic processes in human cells. In this paper we use the following selected pathways, which are described in standard biochemistry books, for a case study.

GP
hk: Gluc + ATP → G6P + ADP + H
pgi: G6P → F6P
pfk: F6P + ATP → FBP + ADP + H
al: FBP → DHAP + GAP
tpi: DHAP ↔ GAP
gapA: GAP + Pi + NAD → NADH + H + BPS
pgk: BPS + ADP → PG3 + ATP
bpgm: BPS → DPG + H
bpgp: DPG + H2O → PG3 + Pi
gpm: PG3 → 2PG
eno: 2PG → H2O + PEP
pyk: PEP + ADP + H → ATP + Pyr
ldh: Pyr + H + NADH ↔ Lac + NAD
F1PP
scrK: Fruc + ATP → F1P + ADP + H
f1pa: F1P → DHAP + GA
tk: GA + ATP → GAP + ADP + H
F6PP
hk2: Fruc + ATP → F6P + ADP + H
GGIP
galK: Galac + ATP → Galac1P + ADP + H
gal: Galac1P → G1P
pgm: G1P ↔ G6P
PPP
g6pdh: G6P + NADP → 6PL + NADPH + H
6pgl: 6PL + H2O → 6GP + H
6pgd: 6GP + NADP → NADPH + CO2 + Ru5P
rpi: Ru5P ↔ R5P
rpe: Ru5P ↔ Xu5P
tkt: Xu5P + R5P ↔ GAP + S7P
tal: GAP + S7P ↔ F6P + E4P
tkt2: Xu5P + E4P ↔ GAP + F6P

Table 1: Formulas of Atomic Reactions, the Modelling Basis

The considered pathways are the glycolysis pathway (GP), the pentose-phosphate-pathway (PPP), the fructose-1-phosphate-pathway (F1PP), the fructose-6-phosphate-pathway (F6PP), and the galactose-glucose interconversion pathway (GGIP). Fructose (Fruc), galactose (Galac), and their pathways interact with the glycolysis. All the individual reactions take place in the cytoplasm of a cell. Glucose-6-phosphate and/or fructose-6-phosphate are intermediate products of all described pathways. The described pathways start always with glucose (Gluc), fructose (Fruc) or galactose (Galac). Reaction products are lactate (Lac), pyruvate (Pyr) and Ribose-5-phosphate (R5P). The gluconeogenesis - the nearly inversion of the glycolysis - is not modelled, because some reactions of them do not take place in the cytoplasm of a cell. Moreover, the gluconeogenesis is only active in liver cells. [5] serves as biochemical reference for this paper. The table 1 shows the considered set of atomic reactions. Some simple sequences will be later summarized.

5 Modelling Strategies of the Core Model

Two steps are necessary to get a system model. The first one is to construct a core model with all biochemical information, which are available. The second step is to calculate an environment for a core model. A third verification step, using the notions of effects and defects, can be additionally made to get a stronger confidence with the system model (not shown). In this section two systematic methods of modelling metabolic

networks to get a core model are introduced and discussed. Before doing this a short description of the problems is given, which arise by the limitation of source metabolites.

To get a live model it is necessary to avoid each possible dead marking under the prerequisites that the source metabolites are bounded. However, the general behaviour, represented by the T-invariants of the P/T net with environment I, must be conserved. Two strategies to construct a core model are introduced by avoiding each critical dynamic conflict.

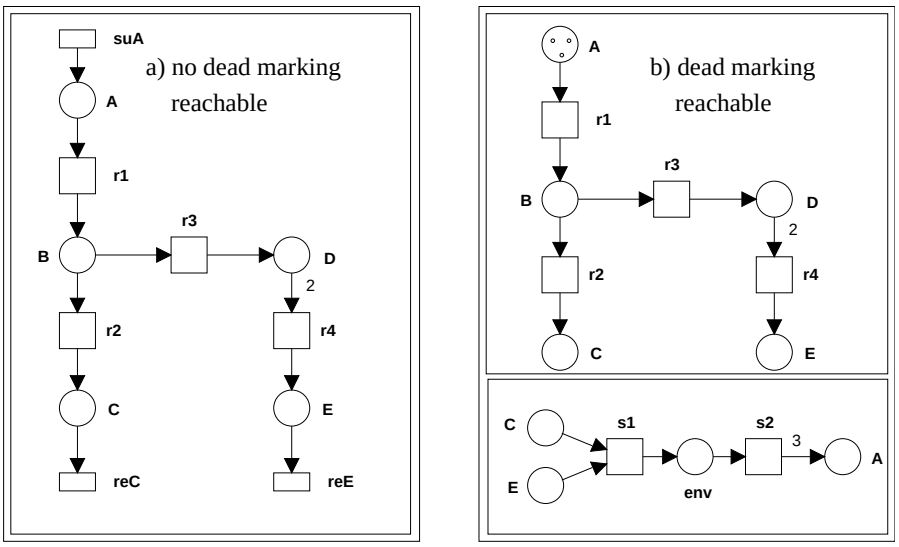


Figure 2: P/T Nets with Environment Types I (a) and II (b)
a) The general behaviour is defined by the T-invariants: T1 = {suA, r1, r2, reC}; T2 = {2*suA, 2*r1, 2*r3, t4, reE}. The net is live and unbounded.
b) A behavioural equivalence is not reached on account of a critical dynamic conflict between r2 and r3 by marking (3*B). E. g. the occurrence sequence o1 = (r1, r1, r1, r3, r2, r2) yields in a dead marking. Therefore, the net is bounded and not live. The T-invariant T3 is the summarized version of the IO-T-invariants T1 and T2. T3 = {s2, 3*r1, r2, 2*r3, r4, s1}

The figure 2 shows two short P/T nets with two different environment types to demonstrate the problem of unmeant dead markings. The point is that the general behaviour of the net b) must be the same as the behaviour of the net a). But the P/T net b) contains at least one dead marking.

There exist two techniques to model relative reaction rates and thereby avoid dead markings. The *first technique* is to change the firing rule and to use information about the relative reaction rates. E. g. a time concept can be used to model the given relative reaction rates. The *second technique* is to avoid critical dynamic conflicts and use an initial marking of the net, which contains indirectly information about relative reaction rates. This technique is demonstrated in this paper. The construction of a core model is described in this chapter and the calculation of an initial marking is described in the next chapter. The first technique is not considered in this paper, but additional information can be found in [17].

Dynamic conflict detection is very expensive, but a dynamic conflict can only be realized on a structural conflict. For this reason no dynamic conflict is calculated, but each structural conflict is intensive considered. Each outgoing arc of a conflict place will be coloured to avoid a dynamic conflict. It should be noticed that only conflict places are considered, which represent primary metabolites. It exist no dynamic conflict at a ubiquitous compound, because it is assumed that the ubiquitous compounds are available in high enough concentrations. Anymore, the primary metabolites determine the main pathways. All arc inscriptions from and to ubiquitous places contain only special token colours with a given multiplicity, which represent conservations of molecules.

Voss et al. have constructed a CPN model avoiding dynamic conflicts. A coloured Petri net with conflict avoidance has the same structure as the P/T net b) in figure 2, but some additional inscriptions, especially the arc inscriptions, are used. The CPN model is very compact and easy to read in contrast to a P/T net, which would be the result of an unfolding process of the CPN model.

By using different token colours, a dynamic conflict can be avoided. A token colour has to represent the information about the pathway on which the token has to go along. This strategy is not biochemically motivated, because there exist no difference of the same molecule. But in biochemical context alternative paths result often in different overall reactions and in different relative reaction rates, which allow us to discriminate molecules of the same type.

The following two core models are constructed by hand, but systematically. Therefore, an algorithm is easy to imagine for an automatic construction. The first one is the application of the method used by Voss et al. in [21]. The second one uses additional information about T-invariants to reduce the effort of construction. The conflict avoidance principle plays a large role during the construction. A calculation of the environment is performed, later in this paper.

Conventions

First, a reduction of each sequence is done. A sequence results in no significant structural information. See abbreviations for the reduced sequences.

Secondly, the following naming conventions are used to obtain clarity. Each occurrence of a logical place (or fusion place) must have its own unique name (prerequisite of Design/CPN). Use the fusion set name with an appended "_x", whereby x is the x-th occurrence of a place in a fusion set, as the name for a place.

Thirdly, by a reversible or equilibrium reaction the transition, which represents the main reaction direction, becomes the enzyme name, which catalyzes the reaction. The other direction becomes the same name with the suffix "_rev". If more than one reaction (different educts, products) is catalyzed by one enzyme, then an additional identifier must be used. The different reaction rates are the biochemical interpretation of the different names.

Fourthly, Design/CPN does not allow place or transition names, which start with another character as a letter. Therefore, another abbreviation as regular must be sometimes used (e. g. 3PG → PG3).

Fifthly, transitions, which have no relation to an IO T-invariant and which are a part of a reversible reaction, are removed. Those transitions have no contribution to the observed system. Therefore, the transitions *ldh_rev* and *rpi_rev* are removed.

Sixthly, each place of the coloured net has the same standard colour set *CS*. A restriction of token colours is indirectly given by the surrounding arc inscriptions of a place.

Variant I

The main construction principle of a core model, exploiting the P/T net with environment I, is the conflict avoidance principle, used by Voss et al. Much knowledge about the modelled system is necessary. It is only a principle and not a rule, because some non-critical dynamic conflicts must not be avoided. Look at figure 4 at place *GAP*. The conflict between the transition *tpi_rev* and all other post-transitions of *GAP* must not avoided, because it exists an internal T-invariant (*tpi_rev* and *tpi*), which reproduces the same marking as it was before *tpi_rev* has been occurred.

```
color CS = with P|NP|C|H|HO|N|
               A1|A5|A6|A7|A8|A9|A10|
               A27|A28|A29|A210|
               A37|A38|A39|A310;
color A2 = subset CS with [A27,A28,A29,A210];
color A3 = subset CS with [A37,A38,A39,A310];
color I = subset CS with [A1,A5,A6,A7,A8,A9,A10,
                          A27,A28,A29,A210, A37,A38,A39,A310];
color B = subset CS with [A1, A27,A28,A29,A210, A37,A38,A39,A310];
color D = subset CS with [A37,A38,A39,A310, A1, A6];
color E = subset CS with [A27,A28,A29,A210, A37,A38,A39,A310,
                          A7,A8,A9,A10];
color E1 = subset CS with [A29,A210,A39,A310,A9,A8];
color E2 = subset CS with [A27,A28,A37,A38,A7,A8];
color E3 = subset CS with [A27,A29,A37,A39,A7,A9];
color F = subset CS with [A7,A8,A9,A10, A5];
color G = subset CS with [A6, A7,A8,A9,A10 ,A5];
color K = subset CS with [A5,A6,A7,A8,A9,A10];
color C1 = subset CS with [A1,A5,A7,A8,A9,A10,
                           A27,A28,A29,A210, A37,A38,A39,A310];

var b:B;var d:D;var i:I;var e:E;var f:F;var g:G;var k:K;
var a2:A2; var a3:A3;var e1:E1;var e2:E2; var e3:E3;var c1:C1;
```

Table 2: Declarations of Coloured Petri Net Core Model, Variant I

A token colour by this variant of modelling represents the sink as target and the back end of the path, on which has to go on the net. For each involved arc of a conflict a separate token colour should be used. To assure this, variables with disjoint ranges must be used. In this paper only variables are used to resolve the conflicts. The figure 4 shows the resulting model, using only the conflict avoidance principle, and the figure 2 shows the corresponding global declarations of the net.

General Procedure - short abstract form

- First, each structural conflict must be determined and all primary paths are temporary coloured by the same token colour.
- Secondly, determine all primary paths *w*, starting backwards from a conflict place *p* to the source places, to get the set of token colours $oldTC$, which can arrive the conflict place *p*. Introduce new token colours *newTC* for each output arc of the conflict place. Combine each token colour of *newTC* with each of $oldTC$ and change the arc inscription along the paths *w*. For each successor path of the outgoing arc of conflict place *p* write the token colour, which corresponds to outgoing arc, until another conflict place or a sink is reached.
- Thirdly, repeat step 2 for each conflict.

Descriptions/Exceptions

Fifteen token colours are used to avoid all critical dynamic conflicts. An example of conflict solving follows. The lower part of the model from *BPS* to pyruvate and lactate is observed. It is easy to see that from *BPS* to *Pyr* two paths exist. Each *Pyr* can be transformed into *Lac*. For this reason there exist four possible paths in the lower part. Two paths transform *BPS* to *Pyr* and two paths transform *BPS* to *Lac*. Four colours are needed for this part. Therefore, each token that reach *BPS* must also represent one of the four paths in the lower part. For example, $A2$ represents the path through $g6pdh, rpe, tkt, tal, tkt2, pfk$ and would be represented by one token colour, if the lower part would not be exist. But with the lower part the token colour $A2$ must be extended with the information about the four sub-paths, represented by an appended constant.

A special operation for this model is done. The place GAP_Ioc corresponds to no fusion set, although all tokens on this place represent a *GAP* molecule like the other places in the fusion set *GAP*. If GAP_Ioc would be correspond to the fusion set *GAP*, it would be possible that the token on this place are consumed by $gapA$ before the transition tal has occurred. If so, it is possible that not enough tokens (*GAP*) are available to transform all metabolites of the pentose-phosphate pathway (no occurrence of tal) into pyruvate or lactate. This problem can be classified. It exists, if an intermediate product occurs more than once in the partial net representation of a considered T-invariant. The figure 3 shows the pattern of this problem.

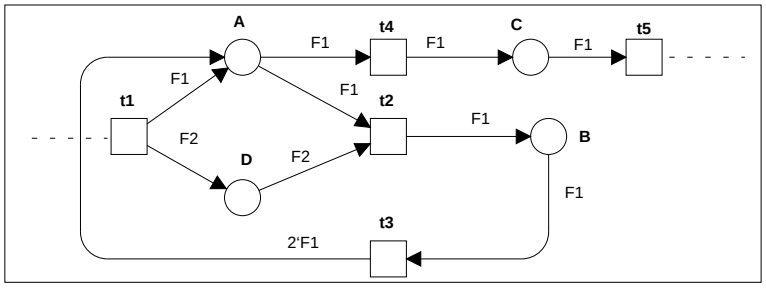


Figure 3: Pattern of a conflict within a minimal T-invariant.
The transitions t1, t2, t3, t4 are a part of a minimal T-invariant. If the transition t1 occurs, then it exist a dynamic conflict between t2 and t4. If t4 occurs before t2 occurs and both belong to the same minimal T-invariant, then a dead marking is possible. Otherwise, the occurrence sequence t1, t2, t3, t4 is harmless.

The problem (the conflict) can be avoided as shown with a special local place. A transition in relation to their guard is able to transform a token from one colour to another colour. This can also be used to avoid such problems. But this solution is not adequate enough and much more complex as a solution with a local place. Voss et al. have an easier model, which does contain such a problem within the easiest form. It was solved by such a transformation from only one token colour to only one another token colour. But if more token colours at the involved place are possible, this solution can not be applied by using only arc inscriptions.

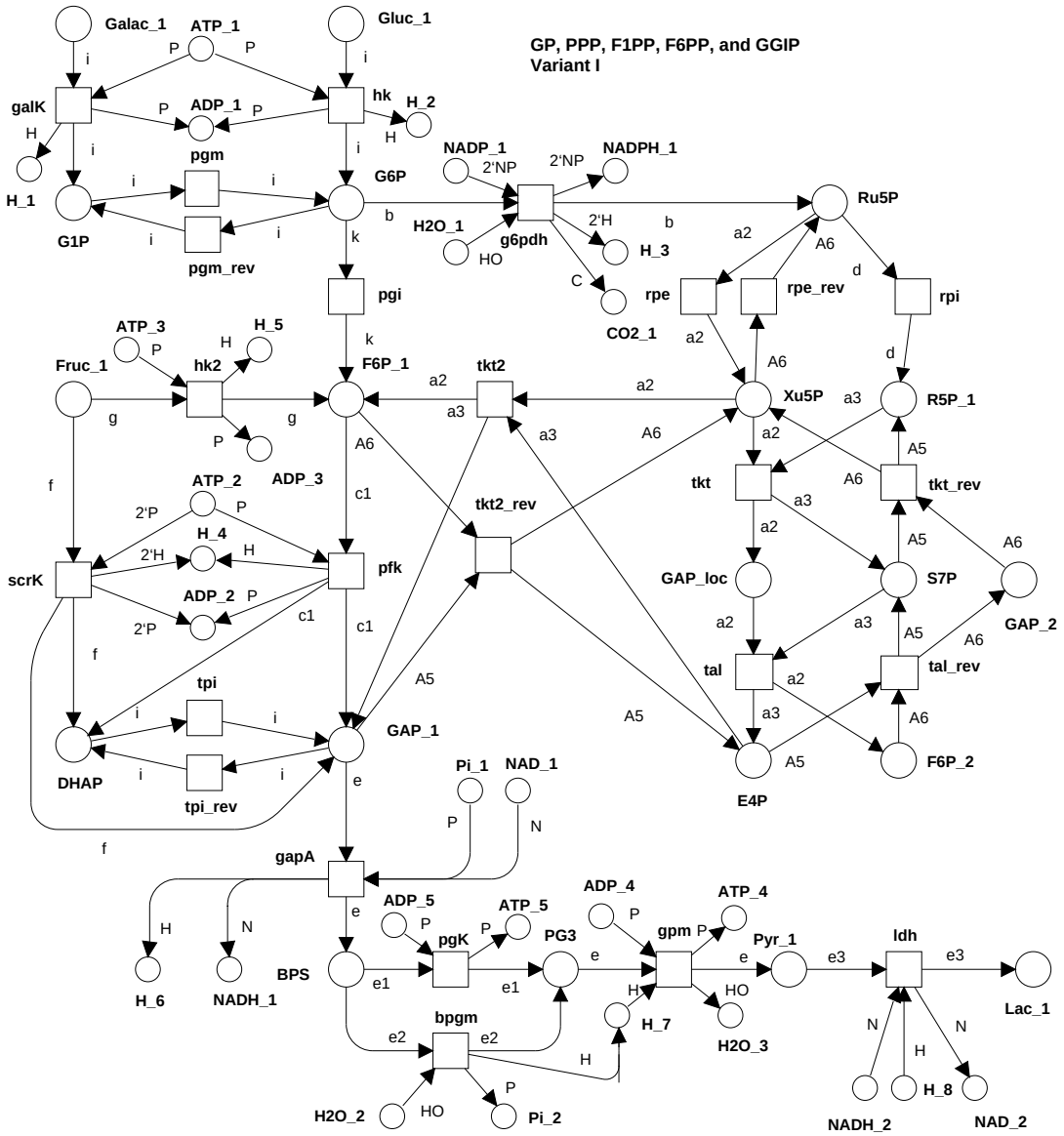


Figure 4: Core Model, Variant I
conflict places: Pyr, BPS, GAP, Fruc, F6P, R5P, E4P, Xu5P, Ru5P, G6P, S7P

By this variant of systematic modelling only the information of structural conflicts of the previously constructed P/T net with environment I is used. A discrimination of the T-invariants by using the token colours is not possible. There exists no relation between them.

Variant II

Another idea to construct a core model from a P/T net with environment I is now introduced. As previously suggested, minimal T-invariants of a P/T model can be used to construct a coloured Petri net core model. The construction of a P/T net is very easy and straightforward. It is a direct reflection of the atomic reactions with their stoichiometric parameter. The resulting P/T model must now be extended by the environment of type I. Hence, a T-invariant analysis is now possible. The calculated non-trivial minimal T-invariants (P/T) represent the general behaviour and the basic structure of the modelled system. These T-invariants are used to construct a core model without critical dynamic conflicts. It should be noticed that the T-invariant calculation depends on the source/sink specification. Therefore, for the given case study all ubiquitous molecules are not observed during the calculation of minimal T-invariants, because elsewhere much more invariants, but with no more new structural information, would be calculated. In other words a sensible selection of minimal

T-invariants (elementary modes) in relation to the biochemical context is done. The tool INA [2] calculates 40 minimal T-invariants, whereby 8 minimal T-invariants are trivial (reversible or equilibrium reactions). After inspection of these T-invariants it was realized that the transitions *ldh_rev* and *rpi_rev* only occur in a trivial T-invariant. For this reason they are removed.

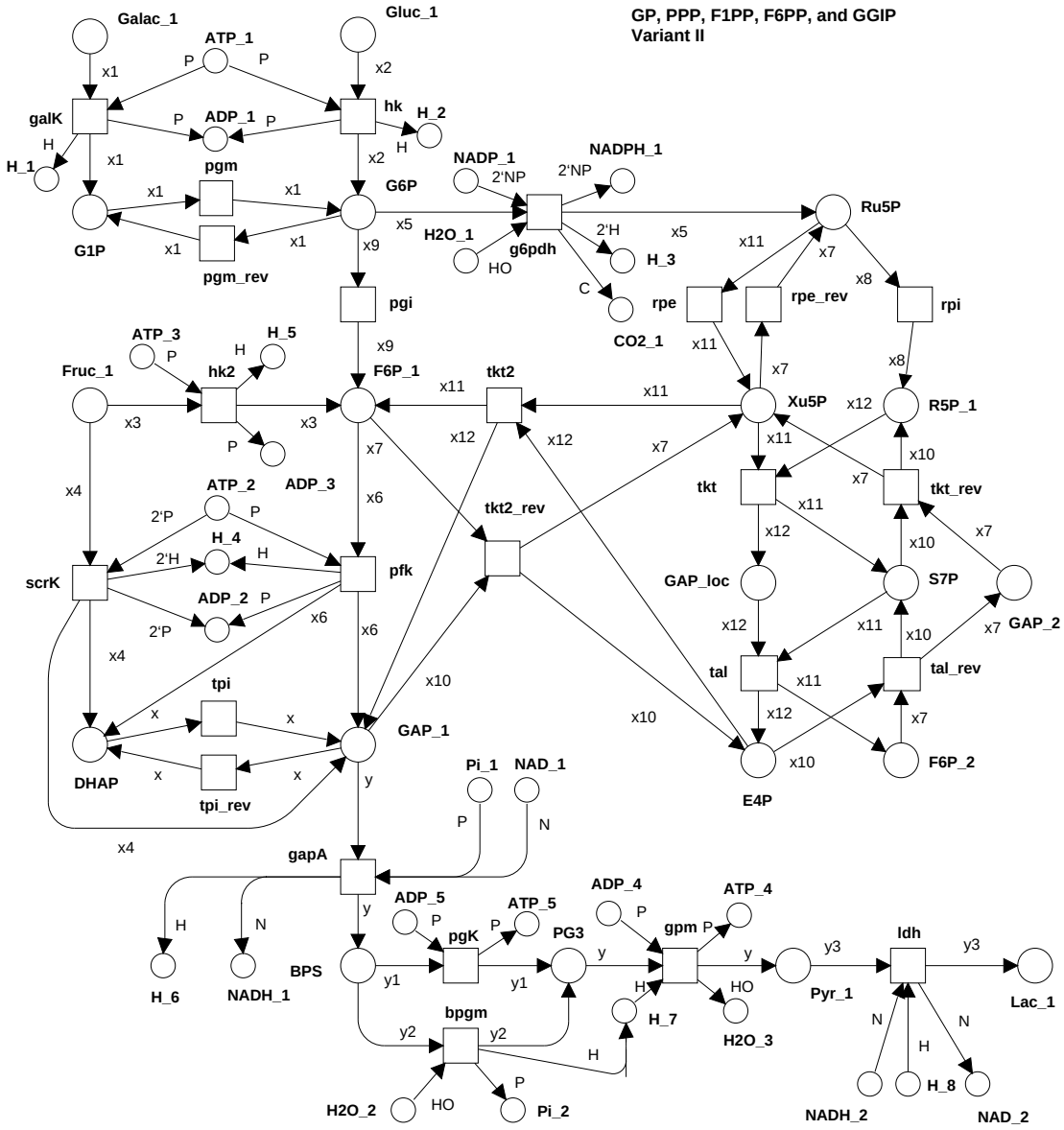


Figure 5: Core Model, Variant II

The basic idea is that a direct relation must be exists between the token colour and a T-invariant. A token colour corresponds only to one minimal IO-T-invariant. For this reason almost each conflict can be avoided, because each token denotes its pathway from a source to a sink. The following selected example, the minimal IO-T-invariant $t38$ demonstrates an exception.

*t38 = {3*hk, 3*g6pdh, 2*rpe, 1*rpi, 1*tkt, 1*tal, 1*tkt2, 2* pfk, 2*tpi, 5*gapA, 5*bpgm, 5*gpm}*

After occurring of three times of *hk* and $g6pdh$, $Ru5P$ contains three tokens. Now a critical dynamic conflict exists between rpi and rpe within the partial virtual net representation of the T-invariant $t38$, containing only nodes that correspond to the given T-invariant. To avoid such a conflict, more token colours must be used to set the path, on which token colour belongs to. This is done by using colour $C38A$ and $C38B$ for the given example. Both colours belong to the given T-invariant, but each of them corresponds to a specific path within the partial virtual net representation. Only such conflicts must be resolved under the assumption that a token

colour exists for each minimal T-invariant. Therefore, the most effort of search a pathway from a conflict place backward to a source is avoided.

The figure 5 shows the resulting model of the current modelling strategy. The corresponding declarations can be found in table 3. Omit the arc inscriptions without multiplicity and we get the corresponding P/T net without an environment. In such a coloured Petri net an equilibrium reaction must result in the same arc inscriptions (same token colours) at both corresponding transitions (*tpi*, *tpi_rev* and *pgm*, *pgm_rev*). Otherwise no equilibrium of such a reaction can be reached. A reversible reaction must result into two transitions with different arc inscriptions (different token colours), because the transitions correspond to different T-invariants.

```
color CS = with P|NP|C|H|HO|N|
                C9|C10|C11|C12|C13|C14|C15|C16|C17A|C17B|C18|C19|C20|
                C21|C22A|C22B|C23|C24|C25|C26|C27A|C27B|C28|C29|C30|
                C31| C32A|C32B|C33A|C33B|C34A|C34B|C35A|C35B|C36A|C36B|
                C37A|C37B|C38A|C38B|C39A|C39B|C40A|C40B;
color X1 = subset CS with[C9,C16,C23,C24,C25,C27A,C27B,C26,C33A,C33B,
                C34A,C34B,C35A,C35B,C36A,C36B];(*galK**)
color X2 = subset CS with[C10,C15,C18,C19,C20,C21,C22A,C22B,C37A,C37B,
                C38A,C38B,C39A,C39B,C40A,C40B];(*hk*)
color X3 = subset CS with[C17B,C28,C29,C30,C31,C32A,C32B];(*hk2*)
color X4 = subset CS with[C11,C12,C13,C14,C15,C16,C17A];(*scrK*)
color X5 = subset CS with[C9,C10,C33A,C33B,C34A,C34B,C35A,C35B,C36A,
                C36B,C37A,C37B,C38A,C38B,C39A,C39B,C40A,C40B];(*g6pdh*)
color X6 = subset CS with[C18,C19,C20,C21,C22A,C23,C24,C25,C26,C27A,
                C28,C29,C30,C31, C32A,C33A,C33B,C34A,C34B,C35A,C35B,
                C36A,C36B,C37A,C37B,C38A,C38B,C39A,C39B,C40A,C40B];
                (*pfk*)
color X7 = subset CS with[C15,C16,C17B,C22B,C27B,C32B];
                (*tkt2_rev von F6P*)
color X8 = subset CS with[C9,C10,C15,C16,C17B,C22B,C27B,C32B,C33B,C34B,
                C35B ,C36B,C37B,C38B,C39B,C40B];(*rpi*)
color X9 = subset CS with[C15,C16,C18,C19,C20,C21,C22A,C22B,C23,C24,
                C25,C26,C27A,C27B];(*pgi*)
color X10 = subset CS with[C15,C16,C17A,C22A,C27A,C32A];
                (*tkt2_rev von GAP*)
color X11 = subset CS with[C33A,C34A,C35A,C36A,C37A,C38A,C39A,C40A];
                (*rpe*)
color X12 = subset CS with[C33B,C34B,C35B ,C36B,C37B,C38B,C39B,C40B];
                (*tkt*)
color Y = subset CS with[C13,C14,C20,C21,C25,C26,C30,C31,C35A,C36A,
                C39A,C40A,C35B,C36B,C39B,C40B,C11,C12,C18,C19,C23,C24,
                C28,C29,C33A,C34A,C37A,C38A,C33B,C34B,C37B,C38B];
                (*gapA = Y1 & Y2*)
color Y1 = subset CS with[C13,C14,C20,C21,C25,C26,C30,C31,C35A,C36A,
                C39A,C40A,C35B,C36B,C39B,C40B];(*pgK*)
color Y2 = subset CS with[C11,C12,C18,C19,C23,C24,C28,C29,C33A,C34A,
                C37A,C38A,C33B,C34B,C37B,C38B];(*bpgm*)
color Y3 = subset CS with[C11,C13,C18,C20,C23,C25,C28,C30,C33A,C35A,
                C37A,C39A,C33B,C35B,C37B,C39B];(*ldh*)

var x1:X1;var x2:X2;var x3:X3;var x4:X4;var x5:X5;var x6:X6;var x7:X7;
var x8:X8;var x9:X9;var x10:X10;var x11:X11;var x12:X12;
var y1:Y1;var y2:Y2;var y3:Y3;
var x:CS; var y:Y;
```

Table 3: Declarations of Coloured Petri Net Core Model, Variant II

Conventions

For this model the following additional conventions are made. First, each token colour contains the identifier (integer number) of the corresponding minimal T-invariant. Secondly, if more than one token colour is necessary, then they will be discriminated by a non-numeric suffix

General Procedure - short abstract form

- First, a simple and straightforward construction of a P/T net with environment type I must be done. After them, a calculation of minimal T-invariants must be performed.
- Secondly, for each transition/reaction collect the minimal T-invariants identifiers, which contains this transition. Create a variable on the corresponding arcs of the transition, which range represents the collected T-invariant identifiers. Examples of the resulting colours are shown in table 3. On account of the restriction of variables no additional guards are necessary.
- Thirdly, it is necessary to compute structural conflicts of a P/T net representation of each minimal T-invariant. This can be easy done by using INA. For each detected conflict within a T-invariant avoid them by adding new token colours to the coloured core model. This can be done by using the conflict avoidance method of variant I, but only within the P/T net representation of a T-invariant. The T-invariants 17, 22, 27, 32, 33, 34, 35, 36, 37, 38, 39, and 40 are examples of such conflicts (to save paper space not shown). Much of the conflicts are at the same place, whereby the effort is very small in difference to the conflict solving by variant I.

Additionally, this modelling strategy in comparison to variant I is faster. The model can be automated constructed, if the set of atomic reactions and the source/sink specification are given. The same problem pattern as described and shown by variant I at the local place *GAP_loc* appears in the current model, too.

A verification of the dynamic conflict avoidance in the both constructed core models can additionally be realized. Let us consider the simple conflict at the place *BPS*. There exist two post-arcs with the variables $y1$ and $y2$ as arc inscriptions. The potential dynamic conflict in relation to a P/T net is avoided, if the following condition is fulfilled. $y1 \cap y2 \equiv \emptyset$ (intersection of the ranges of the variables is empty)

Generalized it may be said, if the intersection of all colour sets of outgoing arcs of a place is equal to the empty set, then no dynamic conflict exists at this place. Now, we have two core models, which are bounded, but not live. The next section introduces an algorithm, which results can be used to get an environment of type II for each of the core model.

6 Computation of the Environment

To get a bounded model, it is necessary to limit the arbitrary supply of metabolites, which is not fulfilled by the environments of type I. This can be done by replacing the supplying and removing border transitions of environments I by two transitions, each for one task (supplying or removing). The figure 6 illustrates a transformation from a P/T net to a coloured net. The environment contains only two additional transitions, a start and a stop transition[1], and one additional place. The additional place "env" contains maximal one token. In relation with the start/stop transition it is used to get an empty core model within the system model. If the place "env" contains a token, no other token is in the net. The start transition supplies all necessary metabolites and the stop transitions removes all produced or transformed metabolites.

Starting from a core model and the information about T-invariants (P/T), a marking can be calculated, which contains the information how much metabolites and ubiquitous molecules must be supplied to realize an elementary mode. Such a marking should be called start marking. If all problematic conflicts are avoided in the core model, then each start marking is transformed by occurrence of the corresponding T-invariant (P/T) without border transitions into the so-called end marking. These marking pair is used to determine the arc inscriptions of the start and stop transitions. At this point a new so-called selection parameter is introduced. A selection parameter corresponds to one minimal T-invariant (naming convention: *t* extended with T-invariant number) and enables us to specify how often a T-invariant should occur by multiplying the start/stop marking with the selection parameter. An example is shown in figure 6. The calculation of such markings and the transformation into arc expressions are automatic.

By using each T-invariant of the P/T net with environment I, it is possible to select each non-negative integer linear combination of the minimal T-invariants for a simulation or analysis of a system model. If only one T-invariant (P/T) is selected (ti = 1; tj = 0; j $\neq$ i), then the resulting T-invariant (CPN) is equivalent to the corresponding T-invariant of the P/T net.

[1] In Design/CPN start and stop are not allowed as transition names. For this reason the names "s1" for stop and "s2" for start are used.

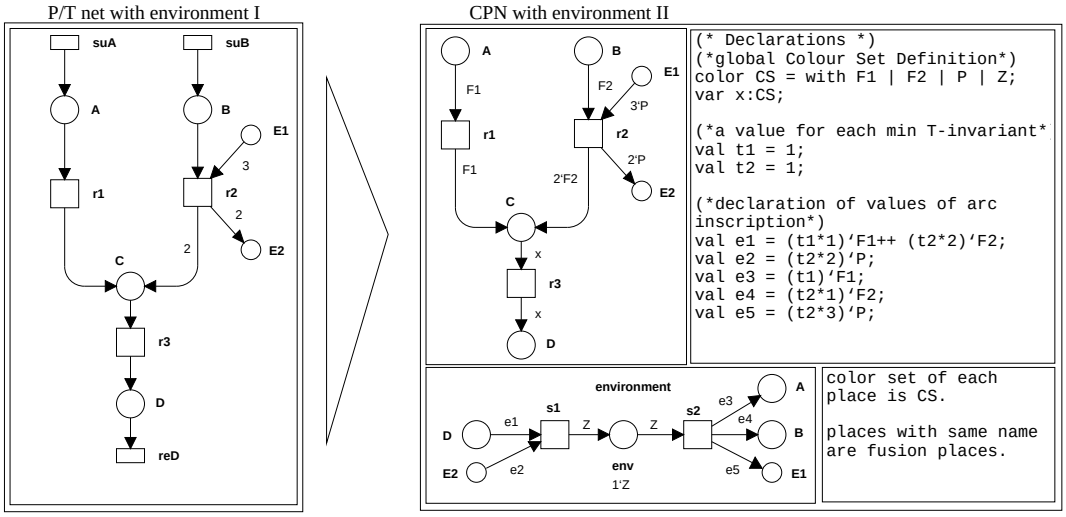


Figure 6: Example P/T Net with Environment I, which is Transformed into a CPN with Environment II
P/T net invariants: T1 = suA, r1, r3, reD; T2 = suB, r2, r3, reD corresponding CPN invariant: T3 = s2, r1, r2, 1*r3(x=F1), 2*r3(x=F2), s1 The calculated start and stop marking for each T-invariant are direct transformed into arc inscriptions of the start or stop transitions. For each T-invariant (P/T) exists a selection parameter (t1 and t2), which indicates how often a T-Invariant must be occur. With a modification of the constant values t1 and t2, each minimal IO-T-invariant can be separately activated.
start marking for t1of CPN core model = A:1* F1; stop marking for t1 = D: 1*F1
start marking for t2 of CPN core model = B:1* F2, E1:3*P; stop marking for t2 = D: 2*F2, E2:2*P

Computation

To get a bounded model, which allows the selection of elementary modes, only the core model, the source/sink specification, and the previously calculated IO-T-invariants (P/T) are necessary. If the core model of variant II is used, these data are completely available. Under the assumption that each critical conflict is avoided and no disease is modelled, it is possible to calculate a start and stop marking for each IO-T-invariant.

The given abstract algorithm, shown in table 4, is similar to a construction algorithm of a run of a single T-invariant (P/T), but the current algorithm has no initial marking as prerequisite. Instead, the source/sink specification is used to identify, on which places a token can be added or removed. The algorithm simulates each T-invariant and memorizes each supplied and produced token separately. During the simulation, each transition of the given T-invariant (P/T) has to occur in relation to its partial order and its weight, whereby each possible binding element of the transition is considered. The calculation is finite, because the given T-invariant is finite.

Results

Using the described algorithm for the computation of start/stop marking pairs for the core models (variant I and II), a transformation into arc inscriptions is also performed. Up to now the graphical representation of the environment, shown in figure 7, must be additionally constructed by hand up to now.

The core model must be only extended by the graphical representation of the environment and the calculated declaration must be added to the global declaration node. To save paper space, only two example declarations for the model of variant II are shown. Two different types are shown. *pre*1 corresponds to a primary metabolite and *pre*2 corresponds to a ubiquitous molecule, which pre- and post-arcs have always the same colour.

On account of the knowledge of the modelled system some new questions are recognized. The calculated markings realize a corresponding T-invariant, but an IO-T-invariant has often more than one interleaving sequence. Therefore, is it possible to calculate a start and stop marking under the assumptions that no potentially concurrency is limited (*maximal concurrency*) or that the concurrency is maximal limited (*minimal concurrency*)? These markings can be biochemically useful. For the current case study only the concentrations of the ubiquitous molecules have an influence to minimal or maximal concurrency.

Input:
$tInv$: TInvariant; P/T net invariant without border transitions (tn)
cpn : coloured Petri net; core model
$sources$: set of places;
$sinks$: set of places;
Output:
$(pre : marking;\ post : marking;\ )$ pre => start marking;post => stop marking
Initialisation:
$post = \emptyset;\ pre = \emptyset;$
tn : transition; current considered transition
$prePostTN$: PrePostMarkings; set of marking pairs (pre, post) of a tn
$lastStep$: PrePostMarkings; $lastStep = \emptyset;$ set of marking pairs, before a tn occur
$currentStep$: PrePostMarkings; $currentStep = onlySources;$ marking pairs, after a tn occured
onlySources -> special initialisation
pet : list of Transitions; possibly enabled transitions
$pet = possibleExtensions(\emptyset);$ which tn will be enabled
$pet = pet \cap tInv;$ observe only net representation of current T-inv.
$pet = removeNotSufficientMarkedTN(pet, post);$ remove tn, which are not enabled by post marking
Main Procedure without Error/Dead Marking Detection/Handling:
while $(tInv \neq \emptyset)$ **do**
$lastStep = currentStep;$
$currentStep = \emptyset;$
$tn = selectTN(pet);$
$prePostTN = tn.getPrePostMarkings();$
for $(int\, i = 0;\ i < prePostTN.length;\ i++)$ **do** prove each tn colour
for $(int\, j = 0;\ j < lastStep.length;\ j++)$ **do** prove each last marking pair
if $(lastStep[j].getPost().covers(prePostTN[i].getPre()))$ is current tn colour enabled
using only source spec. and last marking pair
then simulate occurrence of tn
$pre : marking;\ post : marking;$
$pre = lastStep[j].pre \cup (prePostTN[i].pre \setminus (prePostTN[i].pre \cap lastStep[j].post));$
$post = prePostTn[i].post \cup (lastStep[j].post \setminus (prePostTN[i].pre \cap lastStep[j].post));$
combination is allowed and new Marking is calculated
$currentStep.addPrePostMarking(pre, post);$
fi
od
od
$tInv.occurOnce(tn);$ modify T-invariant to memorize, which part is not currently considered
$pet = pet \cup possibleExtensions(tn);\ pet = pet \cap tInv;$
$pet = removeNotSufficientMarkedTN(pet, currentStep);$
od
$(pre, post) = getMarkingPair(currentStep);$

Table 4: Abstract Algorithm to Get Environment Type II

Existing Problems

On account of a bug of the used Design/CPN tool or CPN/Tools, respectively, a construction of the occurrence graph was not possible up to now. Therefore, the expected liveness property could not be proved. The problem, described by a smaller example, was reported to the "CPNTools-support".

Using the effect and defect notions is another way to increase the confidence in the constructed model (not shown here). By using the defect calculation, it was verified that the net is bounded. Furthermore, it could be a T-vector constructed (for each system model), which is a covering T-invariant of the system model.

The introduced modelling techniques to get a system model with environment type II work fine for all known case studies. But an artificial example can be constructed, whereby the modelling techniques do not work using only minimal T-invariants. Up to now no solution for the artificial example could be derived.

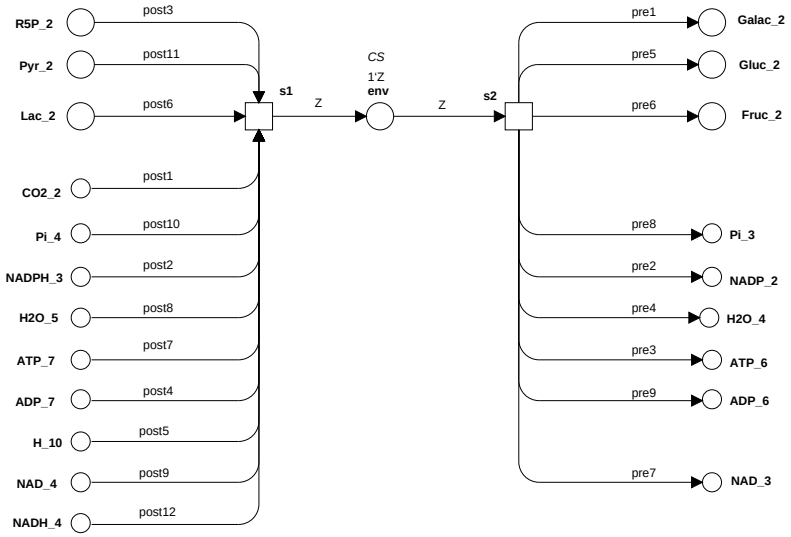


Figure 7: Environment of Both System Models
(the differences lie on the definition of the values on the arcs)

```
val pre1 = (1*t9)‘C9++ (4*t16)‘C16++ (1*t23)‘C23++ (1*t24)‘C24++
           (1*t25)‘C25++ (1*t26)‘C26++ (1*t27)‘C27A++ (4*t27)‘C27B++
           (1*t33)‘C33B++ (2*t33)‘C33A++ (1*t34)‘C34B++ (2*t34)‘C34A++
           (1*t35)‘C35B++ (2*t35)‘C35A++ (1*t36)‘C36B++ (2*t36)‘C36A;
           (*Galac*)
val pre2 = (2*t9 + 2*t10 + 6*t33 + 6*t34 + 6*t35 + 6*t36 + 6*t37 +
           6*t38 + 6*t39 + 6*t40)‘NP;(*NADP 1)
```

Table 5: Declaration Part of the System Model, Variant II

7 Conclusions

Starting with a P/T net and its minimal T-invariants, a behavioural equivalent bounded and live coloured Petri net was constructed. The construction of the coloured net was divided into two separate steps. First, the core model was derived. After that, an automatic calculation of an environment to extend the core model to a system model was performed. The system models of variant I and II represent the same T-invariants/elementary modes as the P/T model with environment type I.

It was shown that a combination of the analysis techniques of P/T nets and coloured Petri nets can be used to get a more useful and sensible model of a metabolic network. The constructed coloured Petri net models are bounded and live. Therefore, we are now able to get new insights into the modelled pathway using additional qualitative analysis techniques, for example model checking or P-invariant analysis. Moreover, by using extensions of the coloured Petri net tool Design/CPN, we are able to perform a quantitative analysis of a qualitatively analysed model without changing the model class.

The construction of a core model must now be implemented. An additional feature of the implementation would be an extraction of pathways from a database [9]. Moreover, a main problem is the selection of the set of atomic reactions, the specification of source and sink metabolites, and in relation to them the treatment of the ubiquitous molecules. The elementary modes/minimal T-invariants depend on this specification. It has to be scrutinized in more detail, how the treatments of the ubiquitous molecules have an influence on the elementary modes. But this is a task for biochemists.

Additional case studies should be performed to increase the confidence in the application of Petri nets in biochemistry and modelling technique presented is this paper.

Acknowledgement

This work is partly supported by the Federal German Ministry of Education and Research (BMBF), BCB project 0312705D. The results are a part of my master thesis, supervised by Monika Heiner, Brandenburg

University of Technology Cottbus, and Ina Koch, Technical University of Applied Sciences Berlin. I would like to thank Monika Heiner and Ina Koch for the fruitful discussions and hints. Furthermore I would like to thank the anonymous referees for their constructive comments.

References

[1] Design/CPN, http://www.daimi.au.dk/designCPN/.

[2] INA - Integrated Net Analyzer v2.2, http://www.informatik.hu-berlin.de/~starke/ina.html.

[3] PED - Petri net Editor, http://www-dssz.informatik.tu-cottbus.de/ wwwdssz/.

[4] Software Package SY, private communication.

[5] BERG, J. M. ; TYMOCZKO, J. L. ; STRYER, L.: *Biochemistry 5th Edition*. W. H. Freeman, New York, 2002

[6] CHEN, M. ; HOFESTADT, R.: Quantitative Petri Net Model of Gene Regulated Metabolic Networks in the Cell. In: *In Silico Biol* 3 (2003), Nr. 3, S. 347–365

[7] EHRENTREICH, F. ; SCHOMBURG, D.: Dynamic Generation and Qualitative Analysis of Metabolic Pathways by a Joint Database/Graph Theoretical Approach. In: *Funct Integr Genomics* 3 (2003), Nr. 4, S. 189–196

[8] GENRICH, H. ; KÜFFNER, R. ; VOSS, K.: Executable Petri Net Models for the Analysis of Metabolic Pathways. In: *International Journal on Software Tools for Technology (STTT)* 3 (2001), Nr. 4, S. 394–404

[9] GEVORGYAN, A. ; HEINER, M. ; KOCH, I.: Japet: an Integrated Tool for Recreating KEGG Data into Hierarchical Petri Net. In: *5th International Conference on System Biology - ICSB 2004, October 9 - 13, 2004, Heidelberg/Germany* (2004)

[10] HEINER, M. ; KOCH, I.: Petri Net Based Model Validation in Systems Biology. In: *LNCS* Bd. 3099, Springer, 2004, S. 216–237

[11] HEINER, M. ; KOCH, I. ; VOSS, K.: Analysis and Simulation of Steady States in Metabolic Pathways with Petri Nets. In: *Proceedings CPN Workshop, Univ. of Aarhus*, 2001, S. 15–34

[12] HEINER, M. ; KOCH, I. ; WILL, J.: Model Validation of Biological Pathways Using Petri Nets - Demonstrated for Apoptosis. In: *Journal BioSystems* 75 (2004), Nr. 1-3, S. 15–28

[13] JENSEN, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1-3. Monographs in Theoretical Computer Science*. Springer-Verlag, Berlin, 1992-1997

[14] JENSEN, K.: An Introduction to the Practical Use of Coloured Petri Nets. In: *LNCS* Bd. 1492, Springer, 1998, S. 237–292

[15] KOCH, I. ; JUNKER, B. H. ; HEINER, M.: Application of Petri Net Theory to Model Validation of the Sucrose Breakdown Pathway in the Potato Tuber. In: *submitted and accepted by Bioinformatics* (2004)

[16] PFEIFFER, T. ; SANCHEZ-VALDENEBRO, I. ; NUNO, J. C. ; MONTERO, F. ; SCHUSTER, S.: METATOOL: for Studying Metabolic Networks. In: *Bioinformatics* 15 (1999), Nr. 3, S. 251–257

[17] POPOVA-ZEUGMANN, L. ; HEINER, M. ; KOCH, I.: Modelling and Analysis of Biochemical Networks with Time Petri Nets - Extended Abstract. In: *CONCURRENCY proc. SPECIFICATION AND PROGRAMMING CS & P'2004, Caputh, Germany, 24-26 September 2004*

[18] REDDY, V. N. ; LIEBMAN, M. N. ; MAVROVOUNIOTIS, M. L.: Qualitative analysis of biochemical reaction systems. In: *Comput Biol Med* 26 (1996), Nr. 1, S. 9–24

[19] RUNGE, T.: Qualitative Path Analysis of Metabolic Pathways Using Petri Nets for Generic Modelling / Technical Report, Brandenburg University of Technology Cottbus, Department of Computer Science, Germany. 2004

[20] STARKE, P. H.: *Analyse von Petri-Netz-Modellen*. B. G. Teubner Stuttgart, 1990

[21] VOSS, K. ; HEINER, M. ; KOCH, I.: Steady state analysis of metabolic pathways using Petri nets. In: *In Silico Biol.* 3 (2003), Nr. 3, S. 367–387

[22] WIBACK, S. J. ; PALSSON, B. O.: Extreme Pathway Analysis of Human Red Blood Cell Metabolism. In: *Biophysical Journal* 83 (2002), Nr. 2, S. 808–818

Abbreviations

Metabolites / Compounds
(chemical formulas from [5] and [22])

2PG	2-Phosphoglycerate	C3H4O7P	Galac	Galactose	C6H12O6
6GP	6-Phosphogluconate	C6H10O10P	GAP	Glyceraldehyde-3-phosphate	C3H5O6P
6PL	6-phosphoglucono-δ-lactone	C6H9O9P	Gluc	Glucose	C6H12O6
BPS	1,3-Biphosphoglycerate	C3H4O10P2	H	Hydrogen Ion	H
CO2	Carbon Dioxide	CO2	H2O	Water	H2O
DHAP	Dihydroxyacetone phosphate	C3H5O6P	Lac	Lactate	C3H5O3
DPG	2,3-Biphosphoglycerate	C3H3O10P2	PEP	Phosphoenolpyruvate	C3H2O6P
E4P	Erythrose-4-phosphate	C4H7O7P	PG3	3-Phosphoglycerate	C3H4O7P
F1P	Fructose-1-phosphate	C6H11O9P	Pi	Orthophosphate, ionic form	HO4P
F6P	Fructose-6-phosphate	C6H11O9P			
FBP	Fructose-1,6-biphosphate	C6H10O12P2	Pyr	Pyruvate	C3H3O3
Fruc	Fructose	C6H12O6	R5P	Ribose-5-phosphate	C5H9O8P
G1P	Glucose-1-phosphate	C6H11O9P	Ru5P	Ribulose-5-phosphate	C5H9O8P
G6P	Glucose-6-phosphate	C6H11O9P	S7P	Sedoheptulose-5-phosphate	C7H13O10P
GA	Glyceraldehyde	C3H6O3			
Galac1P	Galactose-1-phosphate	C6H11O9P	Xu5P	Xylulose-5-phosphate	C5H9O8P
ADP	Adenosine diphosphate				C10H13N5O10P2
ATP	Adenosine triphosphate				C10H13N5O13P3
NAD	Nicotinamide adenine dinucleotide, oxidized form				C21H28N7O14P2
NADH	Nicotinamide adenine dinucleotide, reduced form				C21H29N7O14P2
NADP	Nicotinamide adenine dinucleotide phosphate, oxidized form				C21H29N7O17P3
NADPH	Nicotinamide adenine dinucleotide phosphate, reduced form				C21H30N7O17P3

Correspondence between Petri net transitions, abbreviations, and enzymatic reactions

Tn-name	enzyme name	reduced sequences / included reactions	
hk	Hexokinase		
pgi	Phosphoglucose isomerase		
pfk	Phosphofructokinase	al	Aldolase
tpi	Triose phosphate isomerase		
gapA	GAP dehydrogenase		
pgK	Phosphoglycerate kinase		
bpgm	Bisphosphoglycerate mutase	bpgp	Bisphosphoglycerate phosphatase
gpm	Phosphoglycerate mutase	eno	Enolase
		pyk	Pyruvate kinase
ldh	Lactate dehydrogenase		
scrK	Fructokinase	f1pa	Fructose 1-phosphate aldolase
		tk	Triose kinase
hk2	Hexokinase		
galK	Galactokinase	gal	Galactose 1-phosphate uridyl transferase, UDP-Galactose 4-epimerase
pgm	Phosphoglucomutase		
g6pdh	Glucose 6-phosphate dehydrogenase	6pgl	Lactonase
		6pgd	6-Phosphogluconate dehydrogenase
rpi	Phosphopentose isomerase		
rpe	Phosphopentose epimerase		
tkt	Transketolase		

Composite Event Specification in Active Database Systems: A Petri Nets Approach

Xiaoou Li, Joselito Medina Marín
Sección de Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN, Mexico City, Mexico
lixo@cs.cinvestav.mx

Abstract

Event detection is the first and the most important step for Event-Condition-Action (ECA) rule execution in active database systems. Composite event detection is not easy for most existing active database systems. In this paper, a Conditional Colored Petri Net model (CCPN) is proposed for composite events specification and detection. Composite events are detected by checking *composite* transition enabling and verifying the temporal condition attached on the transition. On the other hand, rule execution are realized by verifying *rule* transition enabling and verifying the rule condition attached on the transition. In this way, both composite event detection and ECA rule execution are integrated in the same CCPN model. Furthermore, examples and implementation issues are discussed. Comparisons show that CCPN is a general model for active database system, and it can be used as an independent engine in many active database systems.

Keywords: active database system, Petri nets, ECA rules, composite events.

I. Introduction

Rules are used in active database systems to monitor situations of interest and to trigger a timely response when these situations occur. They can enforce integrity constraints, compute derived data, control data access, gather statistics and much more. The most general form of these rules is the so-called ECA (Event-Condition-Action) rules. An ECA rule has three basic parts that are event, condition and action. The condition of a rule is evaluated whenever its triggering event occurs. If the condition is satisfied, the specified action will be executed. One rule may trigger or activate another one, and rule behavior depends on both the database transactions and its rule interrelations. By investigating rule interrelations and database state one can analyze rule base properties such as termination, confluence, etc..

Since the event part is what really triggers a rule, event specification and detection is very important in active database systems. Generally events are classified into primitive events and composite (or complex) events. Primitive event may be modification or retrieval operations provided by the database manipulation language of the underlying database system, such as insert, update, delete, abort a transaction, etc. To react on more complicated situations composite events are introduced. Composite events are defined from primitive ones by using event operators such as disjunction, sequencing, conjunction, etc.. Some existing active database systems have considered composite events, and corresponding syntax and semantics were defined [13], [10], [9], [7], [8], [5]. However, most of them are application dependent which means that their semantics are private, and cannot be migrated to other systems. On the other hand, not all types of composite events were considered.

Petri nets are a good modeling technique for describing logic relations. A modified colored Petri nets model CCPN (Conditional Colored Petri Nets) was proposed in our early publications for revealing ECA rule structure and their interrelation [2]. And database states may be abstract as tokens of the Petri net model. The CCPN can not only model interactions between ECA rules, but also can demonstrate dynamic triggering and activation behavior through its firing mechanism [2], [3], [4]. However, only conjunction and disjunction composite events were considered in that work. In this paper, we want to report our advance on specifying composite events with our CCPN model.

There is little result on using Petri nets to specify composite events. To the best of our knowledge, SAMOS is the unique existing active database system that uses Petri nets as event detector [7]. In SAMOS, a colored-Petri-net-liked

model was defined for composite event specification. In order to achieve a correct model of a composite event, many additional places and transitions have to be used to represent the temporal information. Reference [11] is another interesting relational research although it is about network management systems rather than active database systems. In [11] colored Petri nets are used to specify the dependence between events. But, their models are much larger than those in [7] since they have to put extra structure to express temporal relations between primitive events. Our CCPN model will overcome the disadvantage of using redundant structure to specify temporal relation between primitive events since these information can be considered as a condition on transitions.

The paper is organized as following: Section II introduces basic concepts of active database systems and events in ECA rules. CCPN model is defined in Section III. Composite event specifications are described in Section IV. A software tool ECAPNSim (ECA-Petri Nets Simulator) is developed based on CCPN in Section V. Finally, section VI gives comparisons and conclusion.

II. Events in active database systems

ECA rules are not only used in active database systems, but also in other active systems such as network management systems, workflow management systems, etc.. In this paper we only consider ECA rules in active database systems.

A. ECA rules

An active database management system (ADBMS) integrates event-based rule processing in traditional database functionality. Active database consists of a normal (passive) database and an active rule base. The most popular active rules is so-called *event-condition-action* (*ECA*) rules (or *trigger*), which specifies an action to be executed upon the occurrence of one or more events when a condition holds. Generally, an ECA rule is defined as **ON** *event* **IF** *condition* **THEN** *action*. Figure 1 shows a simple architectural view of an active database system [11]. The events corresponding to update operations on the database performed by *user transactions* and other events (such as method execution, time) are reported to the event detector. If a rule fires, the C-A part may be executed as database transactions, if C and A contain database operations. Various transactions models for rule execution have been proposed, that deal with the coupling and synchronization of user-invoked transactions and system-triggered rules. For example, the triggering and triggered transaction can be coupled as *immediate*, *deferred* and *separate*. In the *immediate* coupling mode the fired rule is executed immediately as a *subtransaction* of the *top level transaction* of the triggering transaction. If multiple rules fire and there is an imposed order, then all the rules are executed in that order, otherwise, in arbitrary order. The rules in the *deferred* mode are scheduled to be executed at the end of the transaction, but before the *commit* point (*integrity constraints* are normally executed in deferred mode). The rules in the *separate* mode coupling are executed in a totally separate top level transaction.

ECA rules have an explicit event part. It determines when a rule is executed, what condition acts as filter, which action is internal or external. In almost all existing ADBMS, active rule processing syntax can be classified into two models: knowledge model and execution model. Knowledge model indicates active rules and essentially supports the description of active functionality. The features dealt with this model often have a direct representation within the syntax of rule language [6]. An example of an event may be "update of amount on `BONUS`" or "insert on `SALES`"; A *condition* can be a predicate on database state or query. One or more application procedure-calls (or method-invocations) can also be conditions. For example, "`BONUS`_amount $>$ 100". An *action* can contain many cases, such as data modification and retrieval in relational DBMSs, transaction operations (for example commit and abort), method invocation in OODBMSs, procedure calls in relational DBMSs, and rule operations. For example, "update `EMP` set rank = update.rank+1 where emp_id = update.emp_id".

Execution model of ECA rules specifies how a set of rules is treated at runtime, and it is closely related to aspects of the underlying DBMS (e.g., data model, transaction manage). See reference [6] for more details.

Let´s see an example of ECA rules in active database system.

Example 1: The database is based on the following tables:

```
EMP(emp_id, name, rank, salary)
```

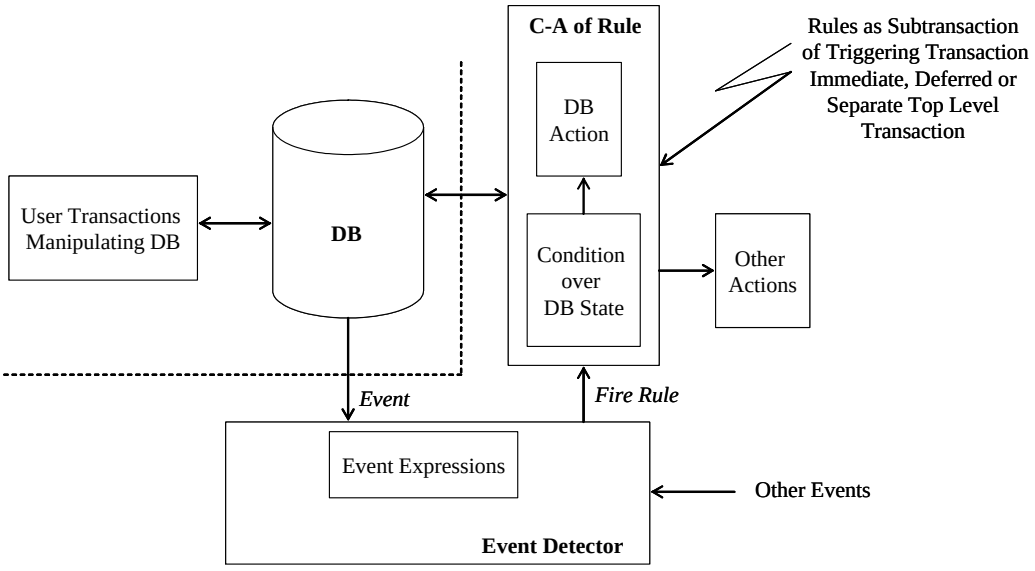


Fig. 1. A simple architectural view of execution of ECA rules

```
BONUS(emp_id, amount)
SALES(emp_id, month, number)
```

The rule base contains 4 rules, which are described in text as following:

Rule 1: When an employee's bonus is increased by more than 100, then the employee's rank is increased by 1.

Rule 2: When an employee's rank is updated, then increased by 1, then the employee's bonus is increased by 10 times the new rank

Rule 3: When an employee posts sales greater than 50 and its rank lower than 15, his bonus is decreased by 100.

Rule 4: When an employee's rank reaches 15, increases the employee's salary by 10%.

In order to see the rule relation clearly, we analyze the EVENT, CONDITION and ACTION parts of these rules, and rewrite them in ON-IF-THEN style as following:

Rule 1:

ON `update of amount on BONUS`
IF `BONUS_amount > 100`
THEN `update EMP set rank = update.rank+1 where emp_id = update.emp_id;`

Rule 2:

ON `update of rank on EMP`
IF `EMP_rank > 5`
THEN `update BONUS set mount = old.amount+rank*10 where emp_id = update.emp_id;`

Rule 3:

ON `insert on SALES`
IF `SALES_number > 50`
THEN `update EMP set rank = old.rank+1 where emp_id = insert.emp_id;`

Rule 4:

ON `update of rank on EMP`
IF `rank = 15`
THEN `update EMP set salary = old.salary*1.1 where emp_id = insert.emp_id;`

B. Events in active database systems

The *event* determines a great width the expressive power of the rule mechanism of an active database. In their more general form they includes database event (such as insert, delete and update),control events (such as begin transaction, commit and abort), temporal events which perhaps are absolute or relative, periodic or aperiodic, and user defined events. Primitive events can be combined by a event algebra which includes operators for sequence, disjunction, conjunction, negation, history, closure, *etc.* An event typically can start more than one rule, and an event can also take part in various event compositions.

An event is an occurrence in the database, and application's environment. An event occurs at a point in time where time is modeled as a discrete sequence of points. The following primitive events are generally supported in an ADBMS:

- Events relating to database manipulation operations such as retrieve, insert, delete, modification;
- Transaction events;
- Explicit time events such as *14:00, Nov. 27*, *5 minute*.
- Method or procedure execution events which may be signalled at the beginning or end of the execution of a method.
- External events raised from outside the database environment. Examples of such events are (abstract) events raised from an application, events defined by a user, events reported from a sensor, etc.

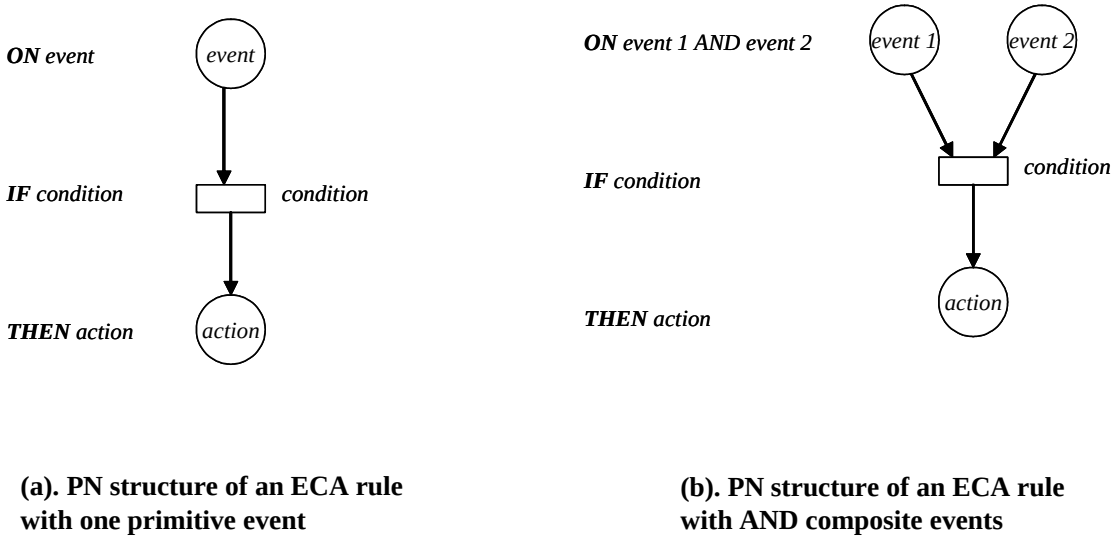
An event may have typed formal arguments which are bound to actual values when the event is detected. For example, the insert event may have as arguments the name of the relation and the inserted tuple. These attributes can then be passed to the *condition* or the *action* part of the ECA rule. A rule may be fired as soon as a single basic event happens. But this is not sufficient for many applications, where complex sequences of events may need to be detected for rule firing. Complex sequences of interrelated events form what is called a *composite event* (also known as *event pattern*). A *composite* event refers to primitive or other composite events occurring at time points other than the time when the specified composite event happens. Composite events are specified using a composite event algebra which allows one to relate events occurring at different time points. For example, selling events of a stock, where the maximum values of the sell price of the stock are sampled at the end of every 30 minute intervals every day from 9AM to 5PM.

The range of event operators varies from system to system. The most common composite events are eight [6] [5]:

1) Disjunction (e_1 *or* e_2): this composite event occurs when either e_1 or e_2 has occurred;
2) Conjunction (e_1 *and* e_2): this composite event occurs when both e_1 and e_2 have occurred in any order;
3) Sequence (*seq(*e_1, e_2*)*): this composite event occurs when event e_1 occurs before e_2;
4) Simultaneous (*sim(*e_1, e_2*)*): this composite event occurs when events e_1 and e_2 occur at the same time;
5) History (*times(n,* e_1*)*): this composite event is signaled when event e_1 occurs n times during the time interval Int;
6) Negation (*not* e_1 *in Int*): this composite event detects the nonoccurrence of the event e_1 in time interval Int;
7) Closure (*closure* e_1 *in* Int): this composite event is raised only once the first time event e_1 is signaled, regardless of later occurrences of e in the time interval Int;
8) ANY (*ANY(m,*e_1*,*e_2*,...,*e_n*)*): this composite event is raised when m of n different events $e_1, e_2, \ldots, e_n$ have occurred, where $m \leq n$.

Rule firings in response to the occurrence of a single primitive event, where the events are only database update events, are supported in commercial systems such as Sybase, Oracle, and DB2. The SQL3 standard defines a *triggering* mechanism where a rule is fired in response to a single primitive database operation event. A number of composite event specification languages have been proposed by researchers: ODE[9], SAMOS[7] , Snoop[8], EPL [14], CEDAR[11], NAOs [10], Chimera [13]. However, composite events handling presents challenges in terms of semantics and efficiency that don't have been fully covered.

structure



(a). PN structure of an ECA rule with one primitive event

(b). PN structure of an ECA rule with AND composite events

$_2.pdf$

Fig. 2. Intuitive PN interpretation of ECA rules

III. Conditional Colored Petri nets

Petri Nets are a graphical and mathematical tool for modeling concurrent, asynchronous, distributed, parallel, indeterministic, and/or stochastic systems. As a member of Petri nets family, colored Petri net (CPN) is widely used in industrial applications since they combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. Petri nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values. CPN has an intuitive graphical representation which is appealing to human beings.

Let´s see how to represent ECA rules with CPNs. In order to express clearly and exactly the three components of an ECA rule, we need to analyze ECA rule execution again. When an ECA rule is executed, event detection is a process to get a result 1 or 0 (corresponding the event is detected or not). Therefore, it is more convenient to model it as a place, and the event may be modeled as a "color". If the detection result is 1, then a colored token is deposited into this place. Furthermore, an action of a rule maybe an event of another rule (i.e., the conjunction of the set of actions and events is not empty), modeling actions as places also obey human cognition. The condition part of an ECA rule will be evaluated after event detection, so it is like a guard of a transition. Here, we call this transition a *conditional transition* in order to emphasize the importance of conditions. If there are tokens in each input place, then the transition is enabled, and if its condition is evaluated TRUE, then the transition is firable. Above idea may be explained intuitively as shown in Figure 2-(a) and Figure 2-(b). In Figure 2-(a), *rule* is mapped into a transition, *event* and *action* are mapped into input and output places of the transition. Finally *condition* is attached to the transition as a guard. In Figure 2-(b), *event* is a AND composite event, so both *event 1* and *event 2* are mapped into input places of the rule transition.

Based on above analysis, we developed a Conditional Colored Petri Net (CCPN) to model, simulate and analyze ECA rule execution. Furthermore, an interface ECAPNSim was developed to implement CCPN.

A. CCPN Structure

As shown in Figure 2-(a), a simple ECA rule with only one primitive event may be modeled as a CPN. But for complicated composite events, basic elements of CPN are not sufficient. For example, composite events *sequence* ($\mathrm{seq}(e_1, e_2)$), *times* ($Times(e,\ Int)$) cannot be modeled by an ordinary CPN directly. For this reason, we defined some new elements on CPN specially to characterize ECA rules features. Figure 3 shows a list of all CCPN elements.

List of CCPN elements:

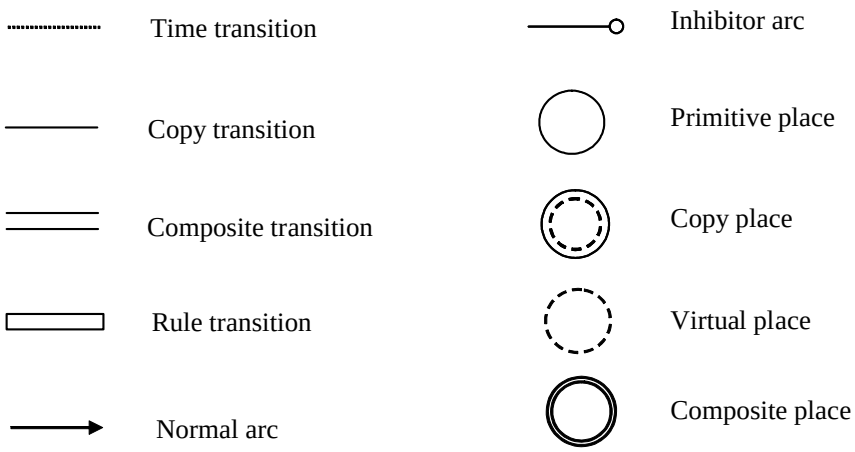


Fig. 3. CCPN elements

IN CCPN, places are classified into *primitive*, *virtual*, *copy* and *composite* places. Primitive and composite places map primitive and composite events; A virtual place is used for OR composite events; Copy places are used when one event trigger more than one rule, we make copies of the event so that when the event is detect all rules triggered by this event can be enabled. Transitions are classified into *rule*, *composite*, and *copy* transitions. A rule transition is a map of a rule, Composite transitions are used to generate composite events from primitive events, and copy transitions are used to generate copies of events (maybe primitive or composite events). Arcs are classified into *normal* arcs and *inhibitor* arcs. Inhibitor arcs are used for negation composite events.

Now we give a formal definition of CCPN. Conditional colored Petri nets is a modified colored Petri nets. It is based on a mathematical concept multi-set that used in reference [12].

Definition 1: A multi-set m, over a non-empty set S, is a function $m \in [S \rightarrow N]$ which we represent as a formal sum $\sum_{s \in S} m(s)'s$. By S_{MS} we denote the set of all multi-sets over S. The non-negative integers $\{m(s) \mid s \in S\}$ are the coefficients of the multi-set. $s \in m$ iff $m(s) \neq 0$.

Some other notations of CPN such as *element of a type*, T, *type of a variable*$Type(v)$, the boolean type $\mathcal{B}$ will be used in this paper.

Definition 2: A conditional colored Petri net (CCPN) is a 11-tuple

$$CCPN = \{\Sigma, P, T, A, N, C, Con, Action, D, \tau, I\}$$

where

(1) Σ is a finite set of non-empty types, called *color* sets.

(2) P is a finite set of *places*. For better graphical representation, P is divided into four subsets, i.e.,

$$P = P_{pri} \cup P_{com} \cup P_{vir} \cup P_{cop}$$

where P_{pri}, P_{com}, P_{vir} and P_{cop} are sets of primitive, composite, virtual and copy places.

(3) T is a finite set of *transitions*. T is divided into three subsets, i.e.,

$$T = T_{rule} \cup T_{copy} \cup T_{comp} \cup T_{time}$$

where T_{rule}, T_{copy}, T_{comp} and T_{time} are sets of rule, copy, composite and time transitions.

(4) A is a finite set of *arcs* such that

$$P \cap T = P \cap A = T \cap A = \Phi.$$

$A = A_{inh} \cup A_{nor}$, where A_{inh} and A_{nor} represent the sets of inhibitor and normal arcs respectively.
(5) N is a *node* function. It is defined from A to $P \times T \cup T \times P$.
(6) C is a *color* function. It is defined from P to Σ.
(7) Con is a *condition* function. It is defined from either T_{rule} or T_{comp} into expressions such that

$$\forall t \in T_{rule} : [Type(Con(t)) = \mathcal{B}]$$

where Con function evaluates the rule condition;

$$\forall t \in T_{comp} : [Type(Con(t)) = \mathcal{B}]$$

where Con function evaluates the temporal condition.
(8) $Action$ is an *action* function. It is defined from T_{rule} into expressions such that:

$$\forall t \in T_{rule},\ p \in t^{\cdot} : [Type(Action(t)) = C(p)_{MS}]$$

(9) D is a *time interval* function. It is defined from T_{comp} to a time interval $[d1, d2]$, where $t \in T_{comp}$, and $d1$, $d2$ are the initial and final interval time, respectively.
(10) τ is a *time stamp* function. It is defined from $M(p)$ to $\{0\} \cup \mathbb{R}^{+}$, which assign each token in place p a time stamp corresponding to natural clock with the form $year : month : day - hour : minute : second$. For example, a token has time stamp $2003 : 11 : 10 - 11 : 16 : 46$.
(11) I is an *initialization* function. It is defined from P into closed expressions such that

$$\forall p \in P : [Type(I(p)) = (C(p)_{MS},\ \tau(C(p)_{MS})]$$

B. CCPN Execution

In CCPN, a transition is firstly verified if it is enabled, then is verified if it is firable, then make token transition. The following definitions specify enabling, firing conditions of a transition.

Definition 3: In CCPN, a token element is a 4-tuple $(p, c, data, timestamp)$ where $p \in P$, $c \in C(p)$ tells the color, $data$ is the color information corresponding the color structure of c, and $timestamp$ specifies the natural time when the token is deposited into place p. The set of all token element is denoted by TE. A marking is a multi-set over TE. The initial marking M_0 is the marking which is obtained by evaluating the initialization expressions:

$$\forall (p, c, data, 0) \in TE : M_0(p, c, data, 0) = (I(p))(c, \tau).$$

The sets of all markings is denoted by M.

Here we introduce a new notation $N_{Color}(p)$ is the number of token colors in place p. If $p \in P_{vir}$, then tokens in p may possess various colors that take from its antecedent places. If $p \in P_{pri} \cup P_{com} \cup P_{cop}$, then all tokens in p have the same color.

Definition 4: A transition $t \in T$ is enabled at a marking M iff

$$1).\ \forall p \in {}^{\cdot}t : |M(p)| = 0, \quad type(t) = Negation$$
$$2).\ \forall p \in {}^{\cdot}t : |M(p)| \geq 1, \quad else$$

Definition 5: When a transition $t \in T$ is enabled, enabled function $C_{enabled}$ is defined *from $P \times T$ into expressions such that:*

$$\forall t \in T,\ p \in {}^{\cdot}t : [Type(C_{enabled}(p, t)) = C(p)_{MS}]$$

When transition t is enabled, an enabled function $C_{enabled}$ is defined to specify what token elements transition t is enabled about. In CCPN, a copy transition is firable if it is enabled. But, enabled composite transitions and rule transitions fire conditionally. A composite transition fires once it is enabled and the temporal condition is satisfied. And a rule transition fires once it is enabled and the rule condition is satisfied.

Definition 6: When a transition $t \in T_{comp}$ is enabled composition function $C_{composition}$ is defined from $T \times P$ into expressions such that

$$Type(C_{composition}(t, p_o)) = Type(t)(C(p_k)_{MS})$$

where $p_k \in {}^{\cdot}t$, and $p_o \in t^{\cdot}$.

Definition 7: A transition $t \in T$ fires iff

(i) $\forall t \in T_{rule}$, t is enabled and $Type(Con(t)) = true$.

(ii) $\forall t \in T_{copy}$, t is enabled

(iii) $\forall t \in T_{comp}$, t is enabled, and $\forall p \in {}^{\cdot}t$,

$$D(t) = [d_1(t), d_2(t)] : [d_1(t) \leq \tau(M(p)) \leq d_2(t)]$$

Definition 8: (token transition) When a transition t is enabled in a marking M_1, and it fires, marking M_1 changes to marking M_2, defined by

(i) if $t \in T_{rule}$, $\forall p \in P$:

$$M_2(p) = M_1(p) - C_{enabled}(p, t) + Action(t, p)$$

(ii) if $t \in T_{copy}$, $p_1 \in {}^{\cdot}t$, $p_2 \in t^{\cdot}$:

$$\begin{aligned} M_2(p_1) &= M_1(p_1) - C_{enabled}(p_1, t) \\ M_2(p_2) &= M_1(p_2) + C_{enabled}(p_1, t) \end{aligned}$$

(iii) if $t \in T_{comp}$, $p_1 \in {}^{\cdot}t$, $p_2 \in t^{\cdot}$:

a) if $Type(t) = Negation$,

$$\begin{aligned} M_2(p_1) &= M_1(p_1) \\ M_2(p_2) &= M_1(p_2) + C_{composition}(t, p_2) \end{aligned}$$

b) else,

$$\begin{aligned} M_2(p_1) &= M_1(p_1) - C_{enabled}(p_1, t) \\ M_2(p_2) &= M_1(p_2) + C_{composition}(t, p_2) \end{aligned}$$

When a transition is enabled, it is not to say it always fires, i.e., it may fire or not fire. If it doesn't fire, we eliminate the tokens that make the transition enabled in order to utilize the memory efficiently, .

Definition 9: When a transition $t \in T_{rule} \cup T_{comp}$ is enabled at a marking M_1, but not fires because $Type(Con(t)) = false$, marking change still exists, new marking M_2 is defined as following:

$$\forall p \in P : M_2(p) = M_1(p) - C_{enabled}(p, t)$$

$M_1[t \succ M_2$ means that M_2 is directly reachable from M_1 after transition t fires.

IV. Composite Event Specification with CCPN

Both events and actions of ECA rules in an active database can be translated into CCPN places. A primitive event is modeled directly by a place , Composite events, unlike primitive events, can't be modeled as just one place, since composite events are created by the occurrence (or not occurrence in the case of negation composite event) of two or more primitive or composite events, then a composite event needs a CCPN structure to generate a place for it. In references [11], [7], such CPN structures are used to detect composite events.

In this section we will illustrate CCPN modeling by all composite events. The 8 composite events considered here are *conjunction*, *disjunction*, *negation*, *sequence*, *simultaneous*, *closure*, *history*, and *any*. e_c represents a composite event, $e_1, e_2, \ldots, e_n$ represent primitive or composite events that are composition elements of the composite event.

Conjunction: Figure 4-(a) shows the CCPN of the expression $e_c = e_1 \wedge e_2$. The composite event e_c happens when both e_1 and e_2 happen. Since $D(T1) = (-\infty, \infty)$, $Type(Con(T1)) = TRUE$, the composite transition $T1$ fires

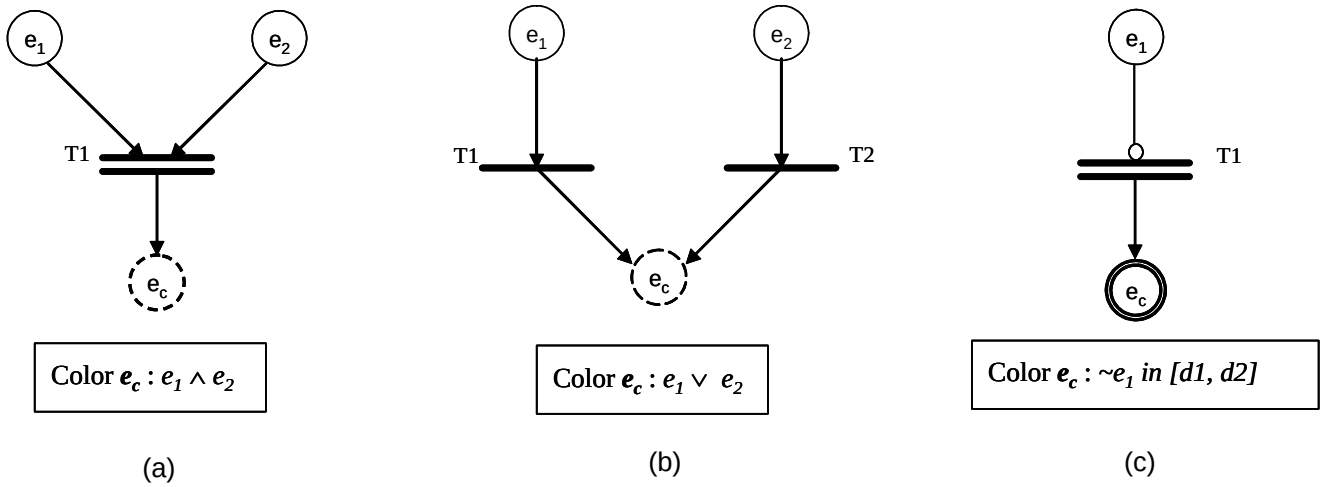


Fig. 4. CCPN structures of composite events

when tokens are available at the places marked as e_1 and e_2, that is when e_1 and e_2 occur. When $T1$ fires the corresponding token at the place e_1 and e_2 are removed and placed in the virtual place marked as e_c.

Disjunction: Figure 4-(b) shows the CCPN of the expression $e_c = e_1 \vee e_2$. The composite event e_c happens when e_1 or e_2 happens. Since $T1$ and $T2$ are copy transitions, $T1$ or $T2$ fires when a token is available at the place marked as e_1 or e_2, that is when e_1 or e_2 occurs. When $T1$ ($T2$) fires the corresponding token at the place e_1 (e_2) is removed and placed in the virtual place marked as e_c accumulating token come from all OR branch.

Negation: Figure 4-(c) shows the CCPN of the expression $e_c = \sim e_1 \ in \ [d1, d2]$. The composite event e_c happens when e_1 does not happen. In this CCPN, $N(e1, T1) \in A_{inh}$, $D(T1) = [d1, d2]$, $Type(Con(T1)) = TRUE$. So the composite transition $T1$ fires when no token is available at the place marked as e_1 in time interval $(d1, d2)$, that is when e_1 does not occur during the time interval $[d1, d2]$. When $T1$ fires the corresponding token at the place e_1 is removed and placed in the composite place marked as e_c.

Sequence: Figure 5-(a) shows the CCPN of the expression $e_c = seq(e_1, e_2)$. The composite event e_c happens when e_1 and e_2 happen in sequence. In this CCPN, $D(T1) = (-\infty, \infty)$, $Con(T1) = \{\tau(e_1) < \tau(e_2)\}$. So the composite transition $T1$ fires when tokens are available at the places marked as e_1 and e_2 and $Type(Con(T1)) = TRUE$, that is when both e_1 and e_2 occur and e_1 occurs before e_2. When $T1$ fires the corresponding tokens at the places e_1 and e_2 are removed and placed in the composite place marked as e_c.

Simultaneous: Figure 5-(b) shows the CCPN of the expression $e_c = sim(e_1, e_2)$. The composite event e_c happens when e_1 and e_2 happen simultaneously. We note that this CCPN structure is the same as that of sequence composite event, the only difference is the condition restraint on the composite transition $T1$, that is $Con(T1) = \{\tau(e_1) = \tau(e_2)\}$. So the composite transition $T1$ fires when tokens are available at the places marked as e_1 and e_2 and $Type(Con(T1)) = TRUE$, that is when e_1 and e_2 occur simultaneously. When $T1$ fires the corresponding tokens at the places e_1 and e_2 are removed and placed in the composite place marked as e_c.

Closure: Figure 5-(c) shows the CCPN of the expression $e_c = *e_1 \ in \ [d1, d2]$. The composite event e_c happens when both e_1 and e_2 happen. In this CCPN, $D(T1) = [d1, d2]$, $Con(T1) = \{\tau(e_1) \in [d1, d2]\}$, e_2 is a time point indicator, e_2 has a token when the o'clock is at the end of interval $[d1, d2]$. $T2$ is a time transition, it fires when the o'clock is at the time point it is specified. So the composite transition $T1$ fires when tokens are available at the places marked as e_1 and e_2 and $Type(Con(T1)) = TRUE$, that is when both e_1 and e_2 occur regardless how many time e_1 has happened. When $T1$ fires the corresponding tokens at the places e_1 and e_2 are removed and placed in the composite place marked as e_c.

History: Figure 6-(a) shows the CCPN of the expression $e_c = times(n, e_1) \ in \ [d1, d2]$. The composite event

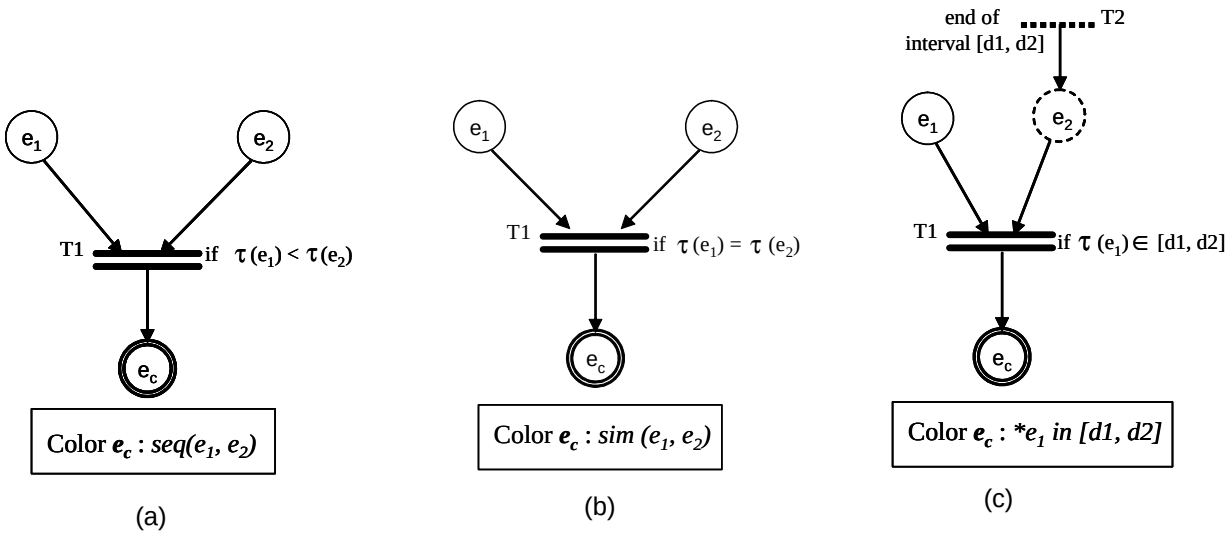


Fig. 5. CCPN structures of composite events

e_c happens when e_1 happens at least n times during the time interval $[d1, d2]$. In this CCPN, $D(T1) = [d1, d2]$, $Type(Con(T1)) = TRUE$, $w(e_1, T1) = n$. So the composite transition $T1$ fires when more than n tokens with timestamp in the time interval $[d1, d2]$ are available at the places marked as e_1. When $T1$ fires the corresponding tokens at the place e_1 is removed and placed in the composite place marked as e_c.

ANY: Figure 6-(b) shows the CCPN of the expression $e_c = ANY(m, e_1, e_2, \ldots, e_n)$, where $m < n$. The composite event e_c happens when any m events of e_1, $e_2, \ldots, e_n$ happen. In this CCPN, transitions $T1, T2, ..., Tn$ are copy transitions, $T0$ is a composite transition, e_0 is a place accumulating tokens of happened events, $w(e_0, T0) = m$ means that the necessary condition to enable $T0$ is place e_0 has at least m tokens. Furthermore, $Con(T0) = \{N_{\mathrm{col}\,or} \geq m\}$, so the composite transition $T0$ fires when there are more than m tokens available at the place marked as e_0, and they take at least m different colors. When $T0$ fires the corresponding tokens at the place e_0 are removed and placed in the composite place marked as e_c.

Modeling ECA rules with CCPN

ECA rules can be easily modeled by CCPN by the following way:

- Events and actions are modeled by places which are inputs and outputs of a transition;
- ECA rules themselves are mapped into transitions, conditions are attached to transitions.
- Rule firing corresponds to transition firing.

However, composite events detection is much more complicated than primitive events. Composite events in ECA rules have to be converted into CCPN structures mention in above section rather than into simple places. After creating events ECA rules can be converted into CCPN. The conversion algorithm from ECA rules to CCPN is showed in Figure 7.

Example 2: An example is illustrated with three active rules, where composite events are the event part of ECA rule. The rules are as follows:

Rule 1: When an employee's record is added into database or his salary is updated. if new salary is bigger than the manager's salary, then the employee's salary will be decreased, and the new salary will be the 20% of the manager's salary.

Rule 2: When an employee is new in the enterprise and in his first day get high sales, then his/her salary is increased in 10%.

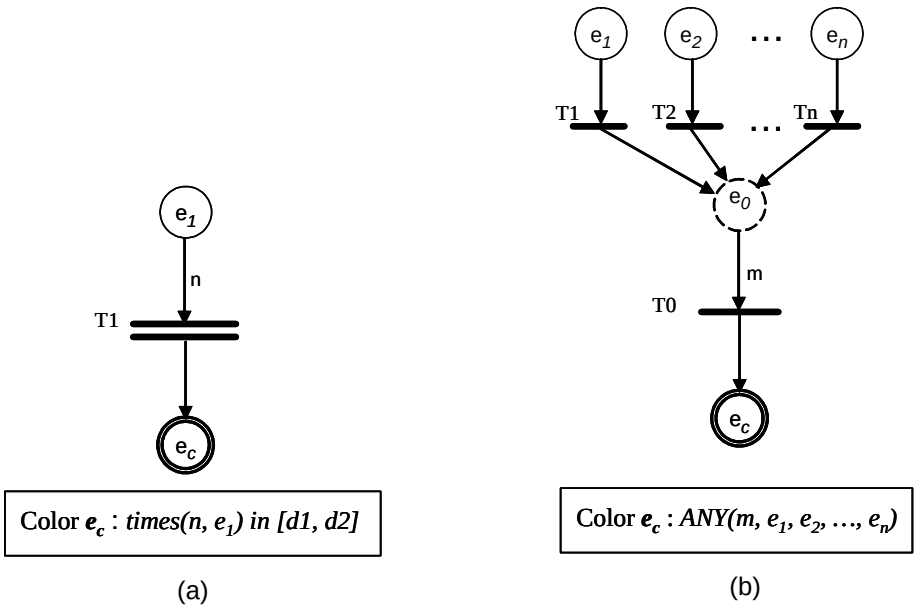


Fig. 6. CCPN structures of composite events

Rule 3: When there weren't records added into the sales table in a working day, then notify to the manager about the sales.

e0, e1 and e2 are used to represent instructions *insert employee, update employee's salary, insert sales* respectively. The above three text rules can be written in *ON event IF condition THEN action* form as following:

Rule 1:

on $or(e_0, e_1)$
if employe.salay > manager.salary
then employee.salary = manager.salary * 0.20

Rule 2:

on $and(e_1, e_2)$
if true
then employee.salary = employee.salary * 1.10;

Rule 3:

on $not(e_2)$
if true
then notify to manager

There are three composite events in this example. They are $or(e_0, e_1)$, $and(e_1, e_2)$, and $not(e_2)$.ECAPNSim will produce the three rules into a CCPN automatically as shown in Figure 8. In Figure 8, places named with letter "E#" represents primitive events, and places named with "EC#" denotes composite events. Transitions T2, and T3 are composite transitions corresponding to the event part of Rule2 and Rule3, and transitions T5, T6, and T7 are rule transitions corresponding to Rule1, Rule2 and Rule3. Places E9 and E10 represent action parts of the rules.

From this CCPN model, it is not difficult to observe that the number of places is not as many as the sum of the number of *events* and *actions*. This is because many events and actions are the same database operations.

V. ECA-PN Simulator Development

ECA-PN Simulator (ECAPNSim) was developed based on CCPN model in order to provide active behavior to a passive database, however, composite events were not taken into account in our previous work [2]. Currently,

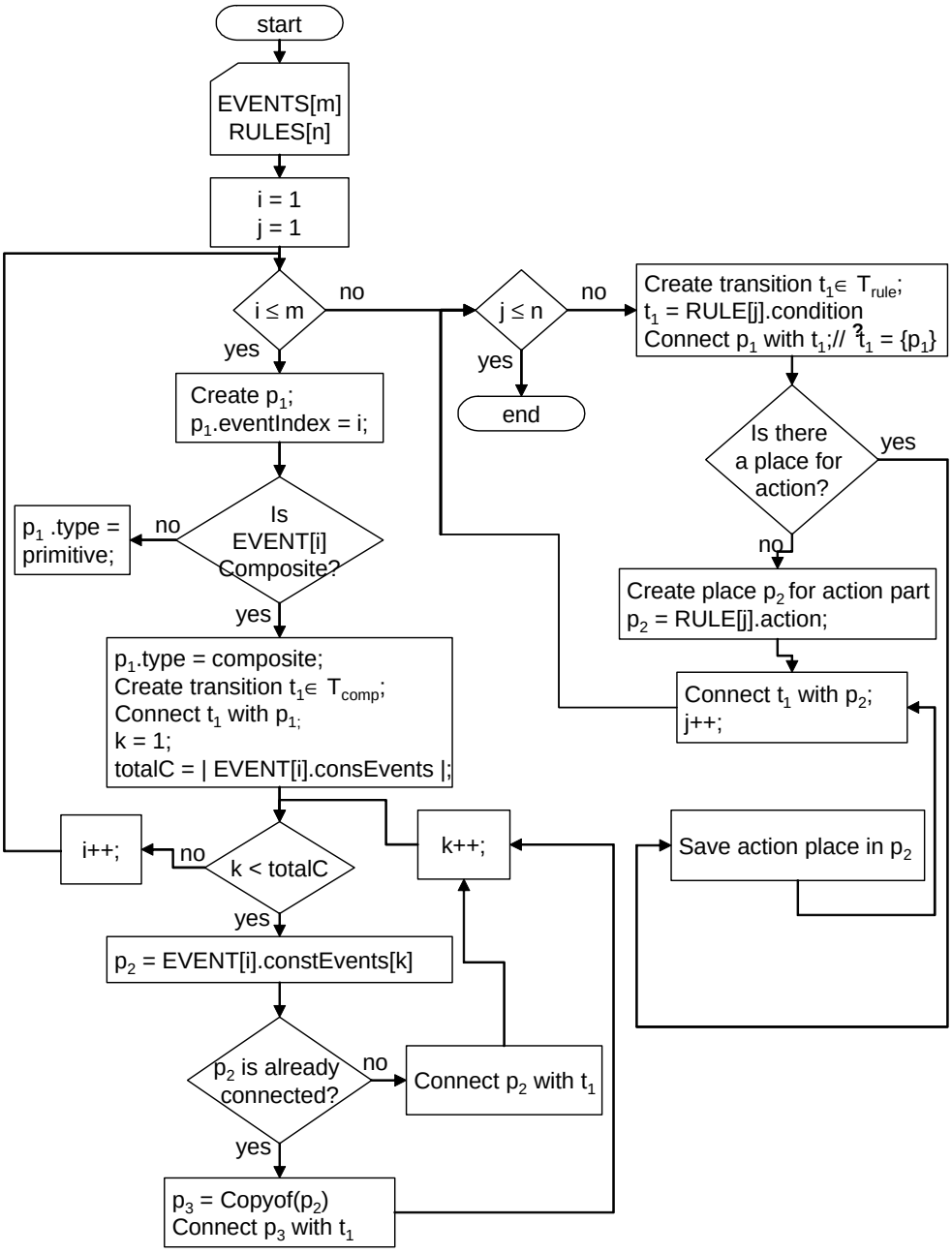
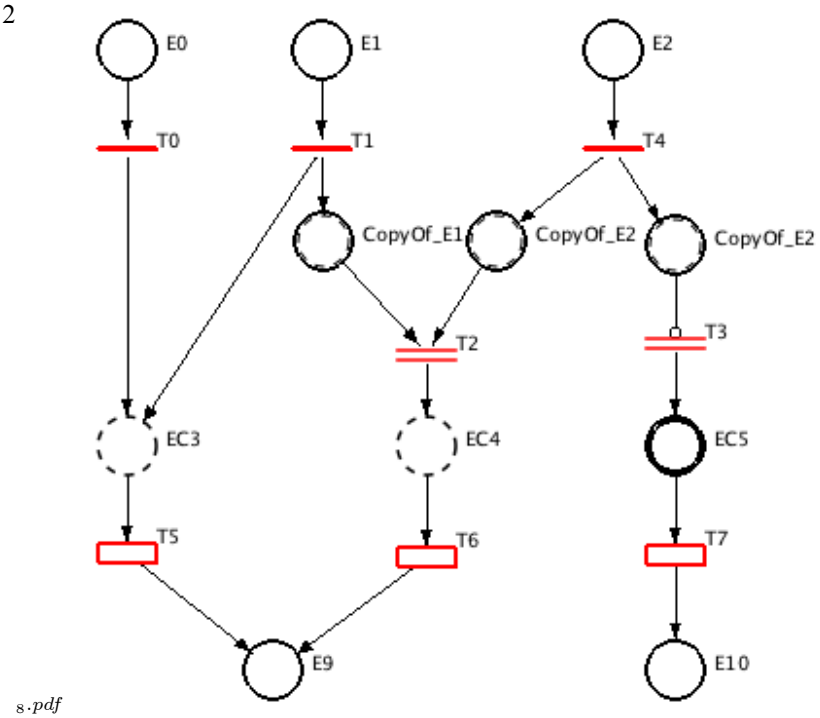


Fig. 7. CCPN conversion algorithm

ECAPNSim has been enhanced by adding new components such as composite events, termination analyzer, etc.. ECAPNSim is developed in Java, under MAC OS X Server. It is used as a layer between users and a passive database. ECAPNSim provides active behavior to traditional databases, i.e., a traditional passive database can work as an active one by communicate with ECAPNSim. For example, a postgres version 7.1 can work as an active postgres by connecting with CCPN through JDBC driver instead of utilizing the TRIGGER SEMANTICS. Firstly, ECAPNSim detects EVENT from database on its state modifications, then executes the simulator; Secondly, if some rules fire on the detected EVENT, ECAPNSim sends ACTION of the fired rules to the database; Thirdly, database makes an operation according to the action provided by ECAPNSim. We will illustrate our results with an active postgres database in the next section.

2



8.pdf

Fig. 8. CCPN model of Example 2

Figure 9 is the object model of composite events implementation in CCPN. *Class Rule* is a main class which has composition associations with the classes *Table, Event, Condition,* and *Action*. Class *Rule* represents an instance to store an ECA rule description, thus it must have the three elements of ECA rule form: an event, a condition, and an action part. Furthermore, Class *Rule* knows the table definition where the event shall occurr. *Class Table* has objects of *Class Fields*, which are used to define fields and field data types defined in the table. *Class Event* has an instance object *Class Interval*. Objects of *Class Interval* store the time interval inside specified by the event.

Class *Rule* objects are taken by the conversion algorithm to create CCPN model, therefore *Classes Place, Transition, InputArc, OutputArc,* and *Token* are used to describe the CCPN structure created from the Class *Rule* objects. Class *Token* is a composition of Classes *Place* and *Transition*, since *Token* objects are held by *Place* objects, and when a transition is enabled, it analyzes the token information, then the *Token* object must be part of the *Transition* object.

A. ECAPNSim architecture

ECAPNSim consists of two building blocks: *ECAPNSim Kernel* and *ECAPNSim tools environment,* see Figure10.

ECAPNSim Kernel provides an active functionality to the passive database. ECAPNSim kernel consists of CCPN Rule Manager, CCPN rule base, Composite Event Detector, and the Rule Execution Component.

- **CCPN Rule Manager**. When an event is detected by the Composite Event Detector, CCPN rule manager verifies wether this event belongs to event set of CCPN, which is monitored in CCPN base. When an event of an existing rule is detected, CCPN evaluates ECA rule, whose conditional part is stored in the CCPN transition. The evaluation is made by the Condition Evaluator. If the evaluation of conditional part is true, CCPN rule manager sends information of ECA rule action to rule execution component. Then action information is retrieved from CCPN rule base.
- **Composite Event Detector**. Primitive and composite events are monitored by this component. First, primitive and composite events defined by ECA rules developer are converted into CCPN structures, then they are stored in the CCPN base. Composite events recognized by composite event detector are *conjunction, disjunction, negation, sequence, simultaneous, closure, last, history,* and *any*. At runtime, composite event detectors "listen" all movements

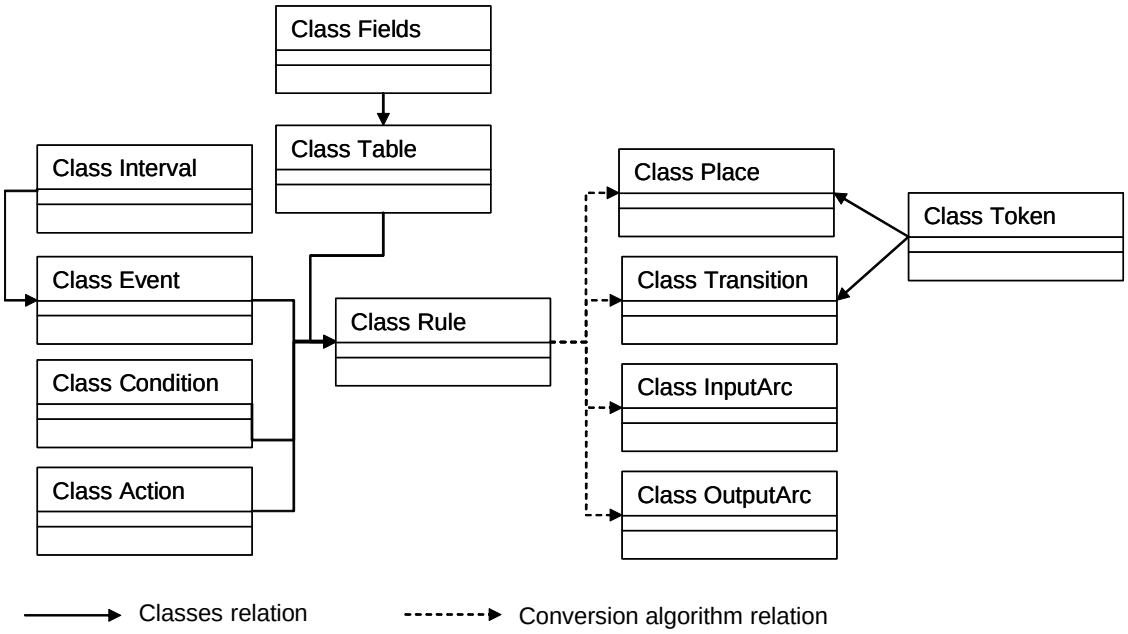


Fig. 9. Object model of composite event generation in ECAPNSim.

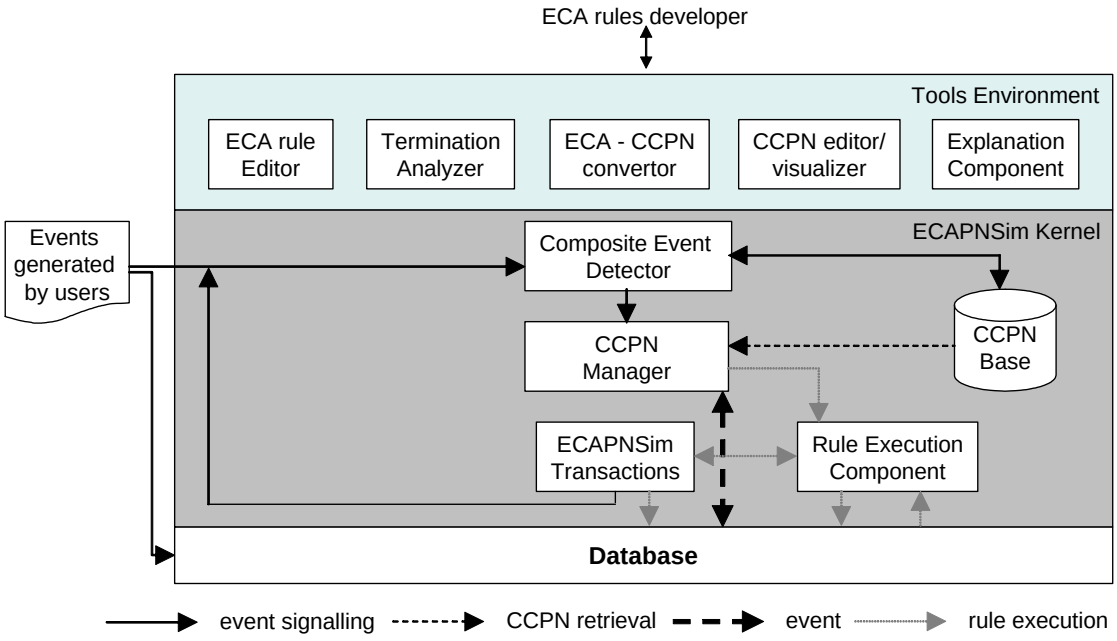


Fig. 10. ECAPNSim prototype architecture.

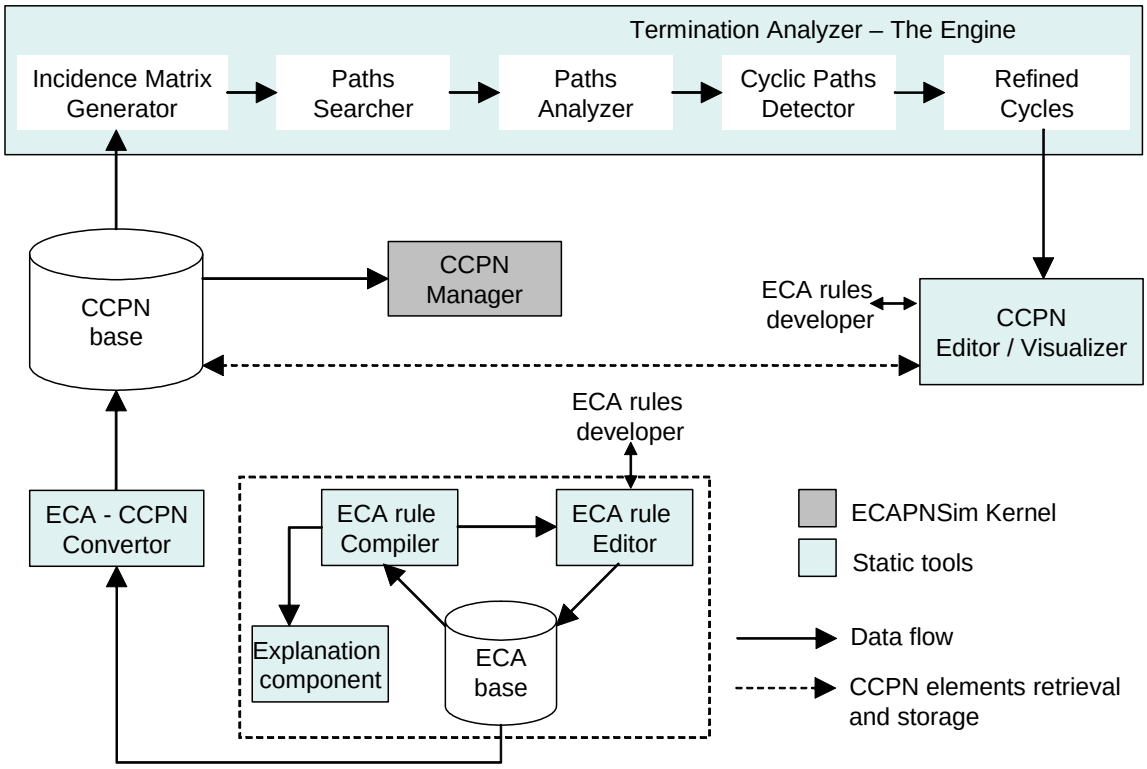


Fig. 11. Buildtime tools

that occur in the database, and verify each one with primitives events via CCPN base or composes composite events. When an important event (primitive or composite) is raised, composite event detector sends a event-detected signal to CCPN manager.

- **Rule Execution Component**. When an ECA rule event is detected by Event Detector, Rule Execution component evaluates conditional part and stores them in a CCPN transition. If the evaluation is true, action part is performed by rule execution component, and an instruction will be sent to database.
- **CCPN Base**. In ECAPNSim, ECA rules are defined by ECA rule editor. After ECA rules are converted into a CCPN, CCPN is saved into CCPN base as places, transitions and arcs. CCPN rule base is used by CCPN manager which follows ECA rule definition (event detection, condition evaluation, and action execution).

B. ECAPNSim Tools

ECAPNSim has a set of tools used by the ECA rules developer. Tools environment is made of ECA rule editor, analyzer of no-termination problem, converter of ECA rules to CCPN, CCPN visualizer/editor and explanation components, termination analyzer and runtime tools. Build-time tools are shown in figure 11. Termination analyzer engine can check no-termination problem in CCPN rule base.

Relationships among ECA rules are depicted in the CCPN model, thus it is a trivial task to detect a cyclic path visually. However, termination analyzer engine need to know the relationships among CCPN elements. Incidence Matrix can be applied to analyze Petri Net, where the connections between places and transitions are transformed into matrix. We use incidence matrix to find cyclic paths in CCPN. Paths in incidence matrix are described as ordered pair sequence. For each ordered pair sequence, *Cyclic path detector* compares every ordered pair with everyone in the sequence, and if there is any ordered pair duplicated in the sequence, then there exists a cyclic path. If there is not any ordered pair duplicated in the ordered pair sequence, then, the sequence path has an initial and a final node. Cyclic paths detector are named *Conditional Part Analyzer*, which verifies conditions stored in CCPN transitions. If

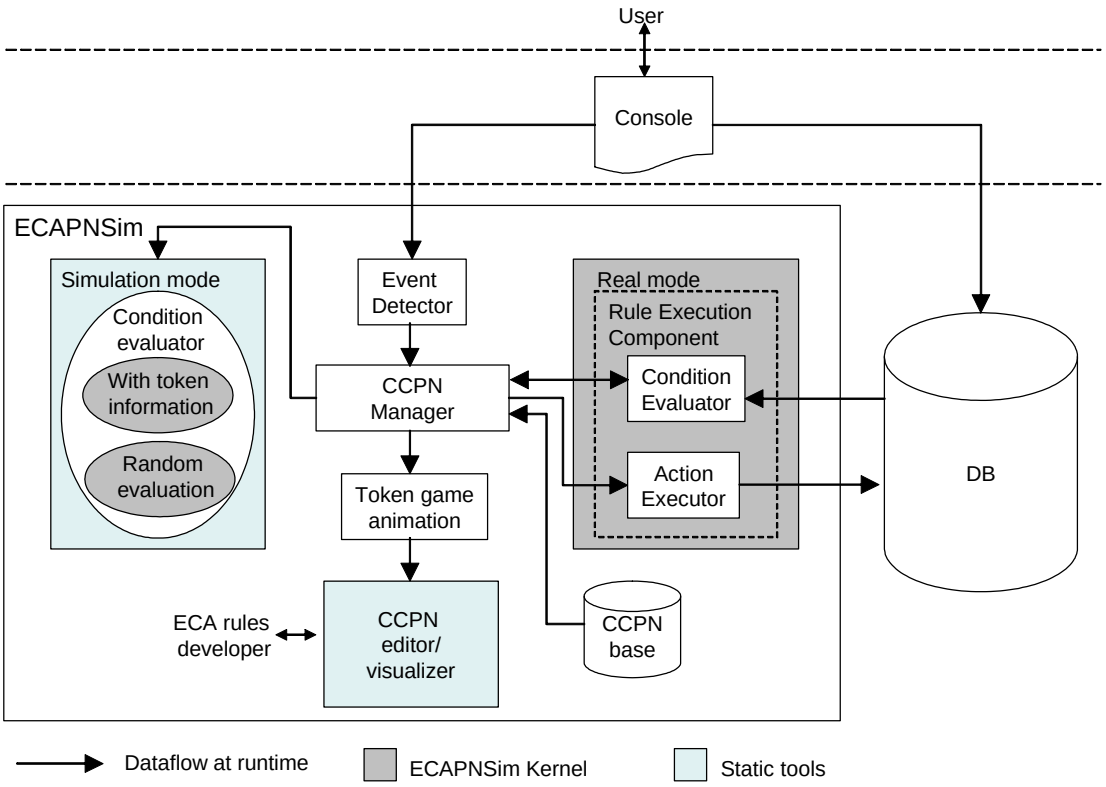


Fig. 12. Runtime tools

there is, at least, one CCPN input place whose token information resulting from a fired rule, then firing rules in this cyclic path can finish. For the limitation of space, termination analysis will be reported in another paper.

Cyclic paths which cannot finish are drawn with different colors and are shown in CCPN editor/visualizer. ECA rules developer can edit CCPN rule base to avoid the no-termination problem. To define ECA rules, ECAPNSim provides an *ECA rule editor*. Every ECA rule should be written according to general form of ECA rules, i.e., ON *event* IF *condition* THEN *action.* After ECA rule base is well written, as syntactically as semantically, "ECA-CCPN converter" begins to convert ECA rule definitions into a graphical structure. At runtime, *ECAPNSim* can be executed in two modes: **Simulation mode** and **Real mode**, see Figure 12. ECA rules developer can verify CCPN behavior in Simulation Mode before its installation over the database, for avoiding inconsistencies.

VI. An active Postgres database system

In this section, we use postgres to show how it is transformed into an active database via ECAPNSim. Our objective is to make postgres active with ECAPNSim instead of using TRIGGER semantics inside postgres system. Example 1 in Section 2 will be used to illustrate results. Originally, data stored in tables `EMP`, `BONUS` and `SALES` are shown in Figure 13, Figure 14 and Figure 15.

The rules are firstly translated into ON...IF...THEN... style (see the last part of section 2), and are saved in a text file **paper.eca**. Now we use the *Import* item of *File* menu in ECAPNSim to convert it into a CCPN as shown in Figure 17. However, Figure 17 has only the structure of rule interaction, there is no dynamic information of the whole active database system.

In fact, both the database and ECAPNSim can work independently, but they cannot work together as an active database. In order to let users work on ECAPNSim easily and directly, an additional window, called *Console* is developed as an user interface. Users can work on the database through *Console* instead of working directly on the database. Any SQL instruction on the database can be operated through *Console* window, see Figure 16.

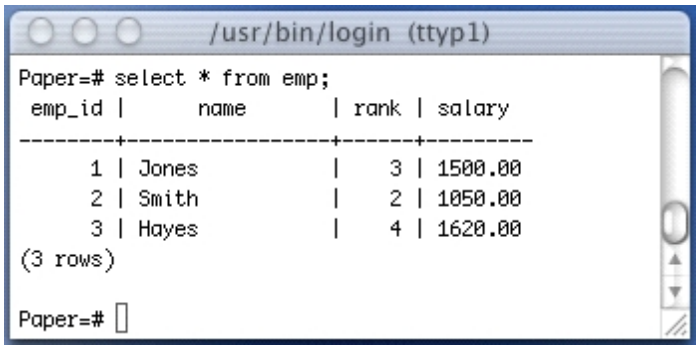


Fig. 13. Original data in table EMP

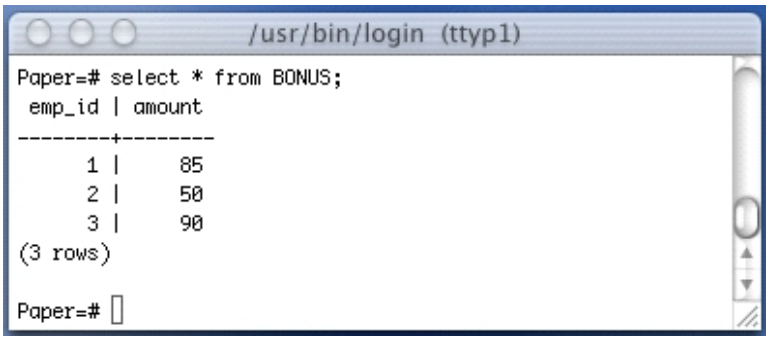


Fig. 14. Original data in table BONUS

In this example, we input a SQL instruction through the *Console,* which updates BONUS_amount whose emp_id is 3 from 90 to 200, see Figure 16. When this instruction is received by *Console*, it is transmitted to both the database and *Event Detector* of ECAPNSim. The *Event Detector* checks the database status, and you will find that the third record of column amount of tale BONUS has been modified (see Figure 18). If it is confirmed as an event of a rule in the rule base, the corresponding place of the event in CCPN, named *update_BONUS_amount,* is deposited a token (as shown in Figure 17). The token information can be seen on the right side of the window *Token Properties* when the user make a "click" on the place.

When the place *update_BONUS_amount* has tokens, transition *T10* is enabled. The condition attached to transition *T10* (IF update.BONUS_amount-old.BONUS_amount $>$ 100) is evaluated true since update.BONUS_amount-old.BONUS_am =110>100. Now T10 fires, the token in place *update_BONUS_amount* is removed, and a token with color EMP_rank=4+1=5

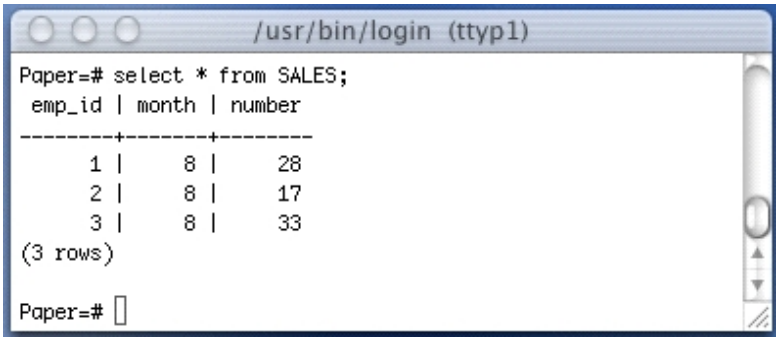


Fig. 15. Original data in table SALES

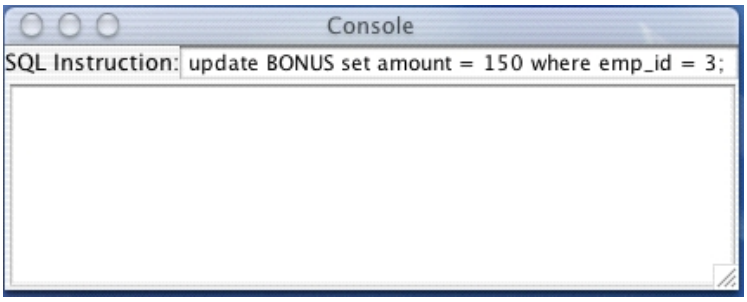


Fig. 16. An update instruction in *Console* window of ECAPNSim

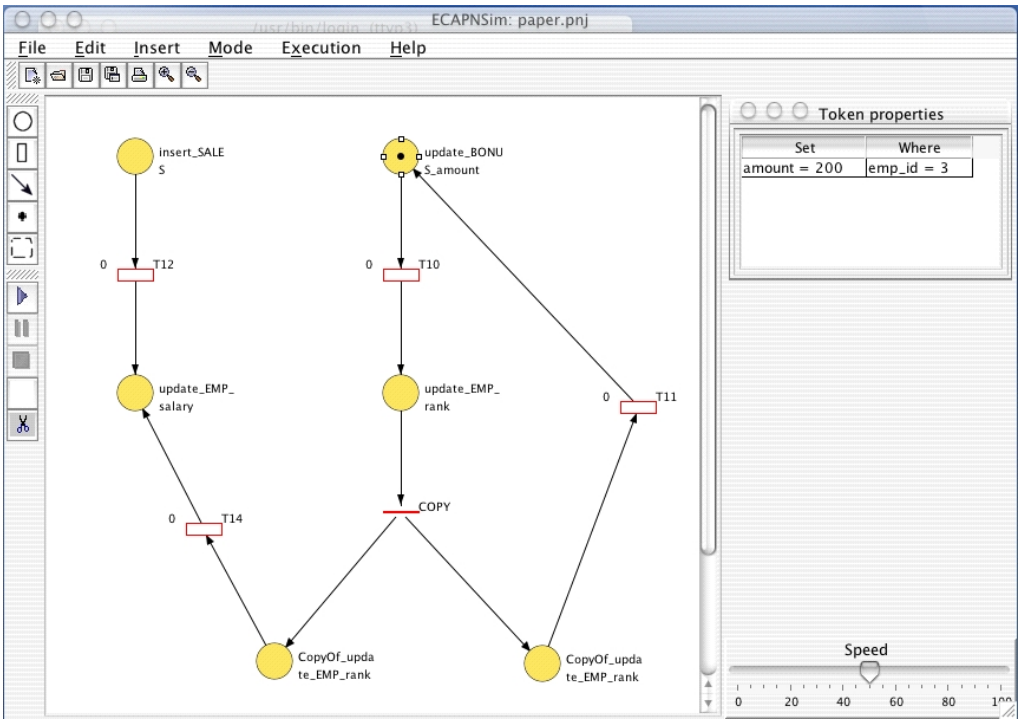


Fig. 17. A token information with color `set amount=200 where emp_id=3` is deposited to the place *update_BONUS_amount*.

is deposited to the place *update_EMP_rank* (the downstream place of transition *T10*). Meanwhile, the action is transmitted to the database. If you check the database status, you will find that column `rank` of the last record has been modified to `5` (see Figure 19).

Until now only the place *update_EMP_rank* in the CCPN model has token, so transition copy is enabled. According to Definition 5, it fires immediately. Then the token in the place *update_EMP_rank* is removed and two places *copy of update_EMP_rank* have one same token. So transitions *T11* and *T14* are enabled. The condition evaluation process are repeated until no transition can fire, and ECAPNSim stops.

From this example, we see that ECAPNSim is easy to connect with postgres. In fact, ECAPNSim can connect with many traditional relational database systems. We have made test successfully on Oracle, Progress, Access too, the results showed that ECAPNSim can be used as an engine of active database.

VII. Conclusion

Here we just make a simple comparison with two other relevant works, one is the CPN model in [7], the other is that in [11]. For example, composite event *Sequence*: *seq*(e_1, e_2) is represented by the CPN in [7] as shown in

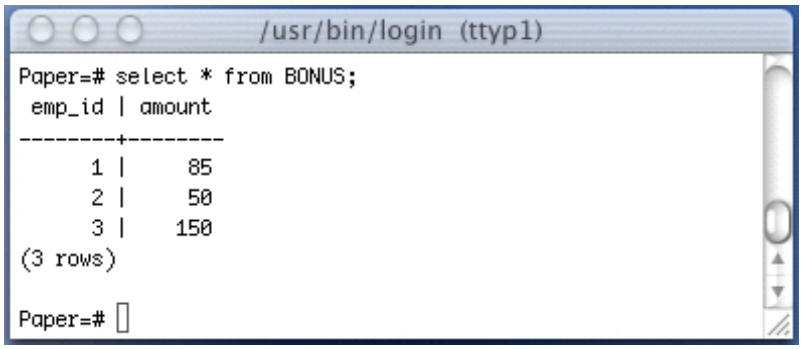


Fig. 18. The database status of table `BONUS` after an instruction from *Console* "`update BONUS set amount=200 where emp_id=3`"

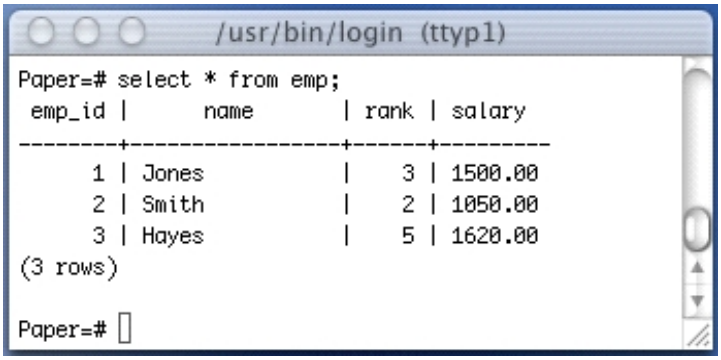


Fig. 19. The database status of table `EMP` after carrying out an action from ECAPNSim "`update EMP set rank = update.rank+1 where emp_id = update.emp_id`"

Figure 20-(a) , by the CPN in [11] as shown in Figure 20-(b), and our model is shown as Figure 5-(a). It is not difficult to see that our model has the following advantage than the CPN models in the other two relational works:

1) The model structure is simple. There are only three places and one transition in our model, but the other two models use much more places and transitions since additional places and transitions have to be used to represent the order of condition attached to the two primitive events.
2) Our model depicts the composite and event-condition-action relationship intuitively and clearly. In our model, transitions correspond to a composite event or a rule. But, these relations are hidden in net structure in the other two models.
3) It is easy to verify the correctness of our model. The mapping from rules to CCPN is direct, EVENT part are modeled as input places, ACTION part is modeled as out places, and composite conditions are attached to the corresponding composite transition. But, in the other two models, one has to use techniques and additional axillary places to represent the composite condition. However, the correctness of the obtained model cannot be guaranteed.
4) CCPN model is a timed model, i.e., time are integrated into the model, so it is suitable for real-time database too. The other two models are logic, time information were not considered.
5) CCPN is implemented easily based on object oriented programming. The other two models didn't discuss implementation issues.

Although CCPN has advantages on modeling ECA rules, it cannot be applied to any area that CPN can be applied. The disadvantages of CCPN are as following:

1) CCPN is not a normal CPN although it takes some concepts of CPN. Tokens in CCPN take information, but the arcs don't have expression for specifying permitted tokens. In CCPN, all tokens have right to enable its

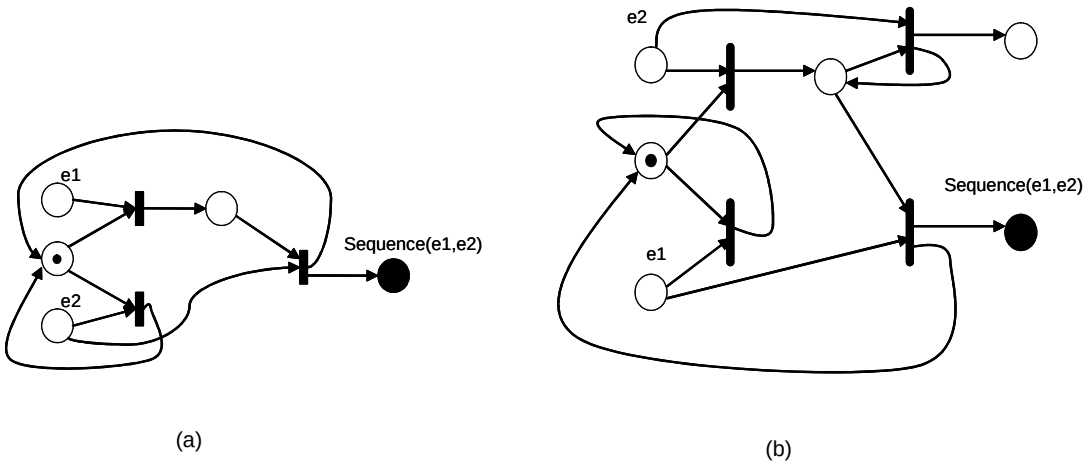


Fig. 20. CPN models of Sequence(e1,e2). (a) CPN model in [7]; (b) CPN model in [11]

output transitions. Therefore, CCPN model semantics is less than CPN´s.

2) Up to now, CCPN is developed specially for modeling ECA rules. We haven't investigate if it is suitable for other systems. So its application areas are not as widely as CPN.

As a conclusion, CCPN is a good model for modeling, analyzing and simulation of active database systems. It may be used as an active engine of database system, but not dependent on the database. We have experimented ECAPNSim successfully on ORACLE, Postgres, Progress, Access. Our future work is applying CCPN on workflow systems.

REFERENCES

[1] Xiaoou Li, Wen Yu and Felipe Lara Rosano, Dynamic knowledge inference and learning under adaptive fuzzy Petri net framework, *IEEE Transactions on System, Man, and Cybernetics, Part C*, vol.30, No. 4, 2000 , pp. 442-450

[2] Xiaoou Li, Joselito Medina Marín, and Sergio Chapa V., A Structural Model of ECA Rules in Active Database, *Mexican International Conference on Artificial Intelligence (MICAI'02), Lecture Notes in Artificial Intelligence series (volume 2313)*, Springer-Verlag, Mérida, Yucatan, México, April 22-26, 2002

[3] Joselito Medina Marín y Xiaoou Li, ECAPNSim, un simulador para reglas ECA, *el XIII Congreso Interuniversitario de Electrónica, Computación y Eléctrica (CIECE)*, Zacatepec, México, Abril, 9-11, 2003

[4] Xiaoou Li, Sergio Chapa, Joselito Medina Marín, and Jovita Martínez Cruz, An Application of Conditional Colored Petri Nets: Active Database System, *IEEE International Conference on System, Man and Cybernetics*, The Hague, Netherlands, Oct. 10-13, 2004

[5] Zimmer D., Unland R., "On the Semantics of Complex Events in Active Database Management Systems", *Proceedings of the 15th International Conference on Data Engineering*, IEEE Computer Society Press, pp. 392-399, 1999.

[6] N.W. Paton and O. Díaz, "Active database systems", *ACM Computing Surveys*, Vol. 31, No. 1, March, 1999

[7] Gatziu S., Dittrich K.R., "Events in an Active Object-Oriented Database System", *Proceedings of the 1st International Workshop on Rules in Database Systems*, Edinburgh, Scotland, 30 August - 1 September 1993. pp. 23-39.

[8] Detlef Zimmer, Axel Meckenstock, Rainer Unland: "A General Model for Event Specification in Active Database Management Systems", *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases, DOOD'97*, Montreux, Switzerland, December 8-12, 1997 LNCS 1341 pp. 419-420

[9] N. H. Gehani and H. V. Jagadish and O. Shmueli, "Composite event specification in active databases: Model & implementation", Proceedings of the 18th International Conference on Very Large Databases, 1992.

[10] C. Collet and T. Coupaye, "Primitive and Composite Events in NAOS", C. Collet and T. Coupaye. Primitive and Composite Events in NAOS. *In actes des 12e Journees Bases de Donneees Avancees*, Cassis (France), pages 331–349. 1996.

[11] Masum Z. Hasan, "The management of data, events, and information presentation for network management", *Doctoral Thesis of the University of Waterloo,* 1996.

[12] CPN introduction papers, *http://www.daimi.au.dk/CPnets/*

[13] Meo, G. Psaila, and S. Ceri. "Composite Events in Chimera". *Proceedings of the 5th International Conference on Extending Database Technology, EDBT'96, LNCS 1057*, Springer Verlag, Avignon, France, pp. 56-78, 1996

[14] I. Motakis, and C. Zaniolo, Formal semantics for composite temporal events in active database rules, *Journal of System Integration (JOSI)*, Vol. 7, No. 3-4, pp. 291-325, 1997.

Modelling Probabilistic Inference using Coloured Petri Nets and Factor Graphs

Somsak Vanit-Anunchai and Jonathan Billington

Computer Systems Engineering Centre
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
Email: vansy014@unisa.edu.au, jonathan.billington@unisa.edu.au

Abstract. Bayesian Networks are a leading technology for probabilistic reasoning in artificial intelligence communities while Petri Nets play an important role in the modelling and analysis of complex systems. The recent development of Factor Graphs and its message passing algorithm help to provide a way of translating Bayesian Networks into Coloured Petri Nets. The integration of these two powerful paradigms will help us to obtain and analyze intelligent agent's belief and preference models. This paper presents our initial work on modelling probabilistic inference based on Pearl's belief algorithm or message passing algorithm of factor graphs using Coloured Petri Nets. Occurrence graph with the equivalence class tool is used to assist validating the inference algorithm. The occurrence graph is expected to have one dead marking with no livelock.

Keywords: Multiagent Systems, Coloured Petri Nets, Bayesian Networks, Belief propagation, factor graphs, State space methods.

1 Introduction

Engineering problems generally involve complexity and uncertainty, with their levels depending on the problem domain. One of the most important theories, which has been successfully applied to solve uncertainty problems, is probability theory. However sometimes when there are too many variables, the probability problem is very complex or intractable and requires a large amount of prior knowledge or data. To mitigate the complexity in the uncertainty problems many subjects such as Statistics, Fuzzy Logic, Econometrics and Decision Theory have been studied.

During the last two decades, Bayesian Networks (BN) have played a leading role for probabilistic reasoning in the artificial intelligence community as well as several other communities. They can perform efficiently not only probabilistic inference but also learning from observed data. Although BN have simplified the representation and probabilistic inference, the size of the BN is still a matter of concern. In general probabilistic inference is NP-hard (Cooper, 1990). Even approximate inference has been shown to be NP-hard (Dagum and Luby, 1993). It is very difficult to model a complex system using BN. There are some proposals to tackle this complexity problem by grouping BNs into interacting sub-networks. Some examples of these attempts are Multiply Section Bayesian Networks (MSBN) (Xiang and Lesser, 2003), Hierarchical Bayesian Networks (HBN) (Gyftodimos and Flach, 2002) and Object Oriented Bayesian Networks (OOBN) (Pfeffer, 2000). Besides complexity reduction, these approaches have the advantages of flexibility, modularity, reusability and ease of maintenance.

On another vane, Petri Nets (PN) are well-known for modelling and analyzing complex, concurrent systems. In many cases, the behaviour of systems, which we study, can be non-deterministic or uncertain. Many different approaches have been proposed and developed to build Petri Nets which are suitable for modelling and analysing the uncertain behaviour of

complex systems, such as Stochastic Petri Nets (SPN) (German, 2000), Fuzzy Petri Nets (FPN), (Shen, 2003). Possibilistic Petri Nets (PPN) (Lee et al., 2003).

When problems are large and subtle, we find many applications are suitable to be implemented by intelligent Multi-agent Systems (MAS). There are two important features of MAS. Firstly, each agent has knowledge or belief about its problem domain (world model) which is often represented by Bayesian Networks. Secondly, MAS require inter-agent communication and cooperation protocols which are often formally modelled and analysed by Petri Nets. Another example is the command and control system in the battle field. This requires assessing the most likely outcome based on unreliable and incomplete evidence, here BN are the leading technique. Before taking the selected action, the system needs to explore and evaluate possible actions in detail. Petri Nets play a leading role in this second task.

According to the above discussion, we argue that modelling uncertain-complex systems, like MAS or command and control systems, should be performed in a common environment so that the merit and accumulated knowledge in both fields can be effectively exploited. As far as we are aware, there are only two groups attempting to merge BN with PN. Firstly, Bayesian Petri Nets were proposed by (Kruse and Lautenbach, 1998) who introduced the notions of *transition tokens* and *backward place firing*. In Pearl's belief propagation algorithm (Pearl, 1988), π messages are defined as messages sent from parent to child nodes and λ messages are defined as messages sent from child to parent nodes. According to Kruse and Lautenbach's work, place tokens represent π messages and transition tokens represent λ messages. Secondly, (Wagenhals and Levis, 1998) proposed executable models of Influence Nets (IN) using Design/CPN. Influence Nets are an extension of traditional Bayesian Networks but Influence Nets use simplifying assumptions that all parents are independent and the variables are boolean. Although IN inference is not exact, it can be efficiently applied to belief networks with loops where Pearl's belief propagation algorithm can not be applied directly (Pearl, 1988). They also extend their work to Timed Influence Nets (TINs) (Wagenhals and Levis, 1999) and TIN with logic (Lindstrøm and Haider, 2001). As indicated in (Haider and Zaidi, 2004), one disadvantage of TINs is that the influence of actions only propagates in the forward direction: from source to target nodes via intermediate nodes. The limitation occurs when the states of non-root nodes are observed but INs can not incorporate such information. To overcome this difficulty, two methods have been proposed. (Haider and Levis, 2004) have proposed an approximation for belief revision in TIN based on the constraint that "the marginal probability of a parent node will not be updated unless all of its children which need to be updated have updated marginal probabilities". The other way is to transform a TIN into a Timed Sliced Bayesian Network (TSBN) or Dynamic Bayesian Network (DBN) (Haider and Zaidi, 2004). In contrast to TINs, our work can give an exact inference, not limited to boolean variables and the observed states of non-root nodes can be incorporate in the inference.

This paper presents our initial work on modelling probabilistic inference using Coloured Petri Nets. The development of Factor Graphs (FG) (Kschischang et al., 2001) and their message passing algorithms provide a way of translating Bayesian Networks into Coloured Petri Nets. Factor graphs and their message passing algorithms can be applied to many fields depending on how the functions and variables are defined. The most popular application of factor graphs seems to be signal processing and communications such as in the area of error control coding. When applying factor graphs to Bayesian Networks, a message passing algorithm with the sum of product operation is equivalent to Pearl's belief propagation. In particular, we propose a formal model of Pearl's belief propagation algorithm using CPNs.

The occurrence graph with the equivalence class tool is used to assist validating inference algorithms. The occurrence graph is expected to have one dead markings with no livelocks.

The rest of the paper is organised as follows. We review the technical background of Bayesian Networks, inference algorithms, factor graphs and the message passing algorithm in section 2. Assuming the reader has some familiarity with CPNs, section 3 explains the description of our Coloured Petri Net (CPN) model. Section 4 discusses the idea how our model works. Section 5 discusses the simulation and analysis of the model. Section 6 explains the contribution of combining BN with CPN. Finally, conclusion and future work are presented in Section 7.

2 Technical Background

2.1 Bayesian Networks

Bayesian Networks (BN) are graphical models which represent joint probability density functions and dependency relationships among random variables. A BN is defined as a Directed Acyclic Graph (DAG) $G = (V, E)$. Random variables are represented by nodes (V) in the graph while dependency relationships are indicated by edges (E) connecting pairs of variables. Given nodes X and Y in V, if there is an edge from X to Y, Y is called a child of X and X is called a parent of Y. Each variable is conditionally independent of the others given values for its parents. Random variables Y and Z are said to be **conditionally independent** if there exists a random variable X, such that the joint probability of Y given X and Z denoted $P(Y|X,Z)$ equals the probability of Y given X ($P(Y|X,Z) = P(Y|X)$) when X is possible ($P(X) \neq 0$). After we associate a local Conditional Probability Table (CPT) with each node, a BN represents the joint probability defined in equation 1.

$$P(X_1, X_2, ..., X_N) = \prod_{k=1}^{N} P(X_k|pa(X_k)) \tag{1}$$

where $pa(X_k)$ is the set of parents of a random variable X_k and N is the number of random variables.

Figure 1 shows an example of a Bayesian Network named the lung cancer network which represents the joint probability distribution of boolean variables: smoking history (H); bronchitis (B); lung cancer (L); and chest X-ray (C). The tables in the figure are the Conditional Probability Tables (CPT) associated with each node. From an empirical study, suppose smoking causes lung cancer and bronchitis with probability $P(L = True|H = True) = 0.003$ and $P(B = True|H = True) = 0.25$ respectively so that there are directed edges from the H node to the L and B nodes. Without a smoking history, the probability of lung cancer $P(L = True|H = False) = 0.00005$ and probability of bronchitis $P(B = True|H = False) = 0.05$. Chest X-ray could reveal the cancer with probability $P(C = True|L = True) = 0.6$, however it could be a false alarm with probability $P(C = True|L = False) = 0.02$. The lung cancer network represents the joint probability

$$P(B, H, L, C) = P(B|H)P(H)P(L|H)P(C|L). \tag{2}$$

When the joint probability is known, theoretically we can compute marginal probabilities (Neapolitan, 2004) of any variable combinations. Given some observed state of the nodes, called evidence, we can calculate the marginal posterior probabilities $P(X_k|evidence)$. For

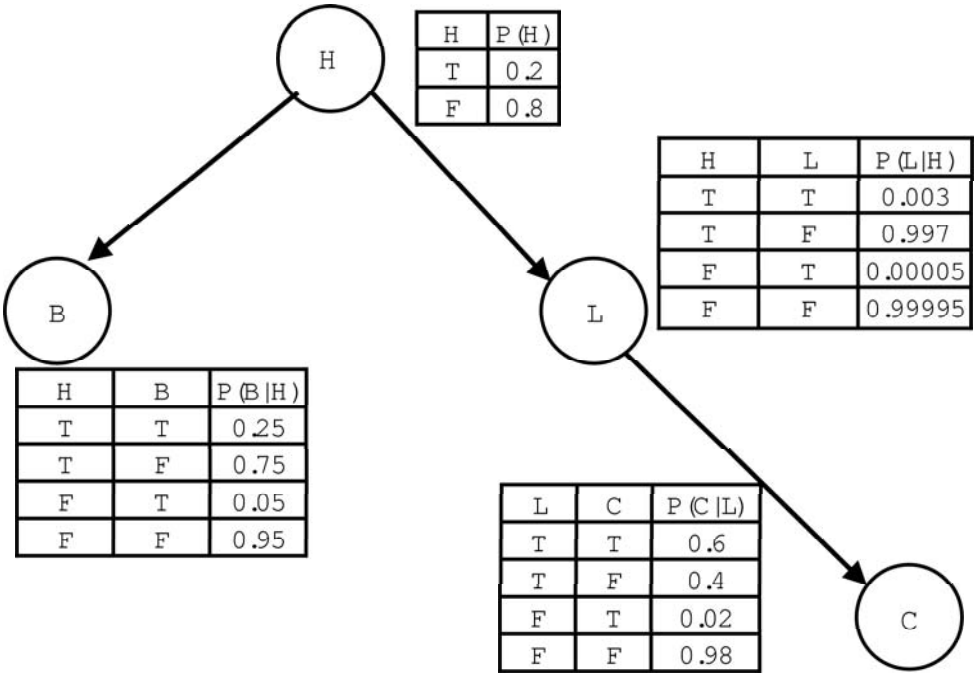


Fig. 1. The lung cancer network.

example, if there is evidence that a patient has bronchitis ($B = True$), we can calculate the chance that he could have lung cancer ($P(L = True|B = True)$). This process is called probabilistic inference or belief updating (Guo and Hsu, 2002). On the other hand, given the observed evidence, we may be interested in finding the most probable instantiation of some hypothesis variables. When all hypothesis variables are not observed, this inference process is called a most probable explanation (MPE).

2.2 Review of Bayesian Network Inference Algorithms

In the early 1980s Pearl proposed an exact and efficient message propagation inference algorithm for singly connected or polytree networks (Kim and Pearl, 1983) (Pearl, 1988). A singly connected network is a graph which has only one path between any two nodes. While general exact inference is NP-hard, Pearl's algorithm has polynomial complexity in the number of nodes. In order to apply his algorithm to a multiply connected network, (Pearl, 1988) suggests loop cutset conditioning to transform the multiply connected network into multiple polytree networks by instantiating a selected subset of nodes as illustrated in figure 2. Then each polytree network can be solved by the message propagation algorithm. The results from networks then are weighted by its prior probabilities and added together. For example, suppose all nodes in the figure 2 (Neapolitan, 2004) are boolean variables. We can compute the marginal probability $P(W)$ from

$$P(W) = P(W|X = x_1)P(X = x_1) + P(W|X = x_2)P(X = x_2) \tag{3}$$

where the prior probability is $P(X = x_1)$ and $P(X = x_2)$. $P(W|X = x_1)$ and $P(W|X = x_2)$ can be computed by applying Pearl's belief propagation to networks in figure 2 b and c respectively.

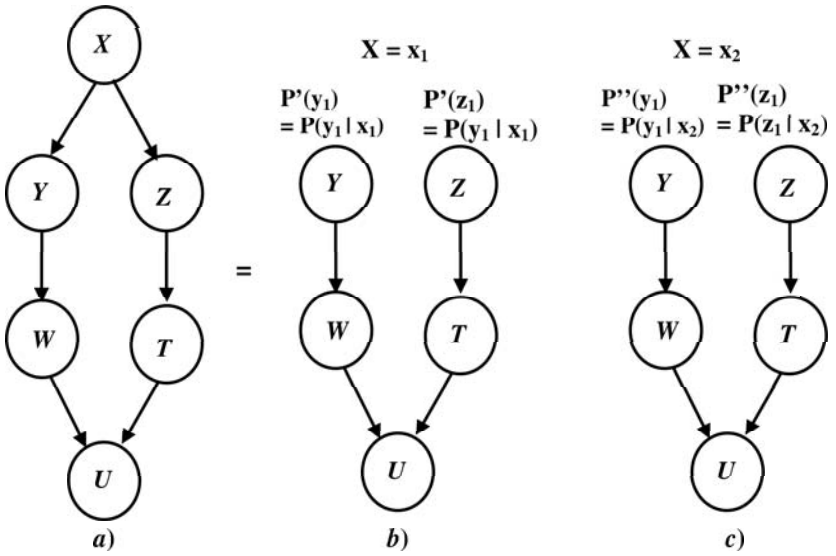


Fig. 2. A multiply connected network is transformed into two polytree networks by instantiating X with $x1$ and $x2$ (Neapolitan, 2004).

However, the complexity of the loop cutset method increases exponentially with the size of the loop cutset and the number of possible values of instantiated variables. (Pearl, 1988) also proposed a node clustering method an example of which is shown in figure 3.

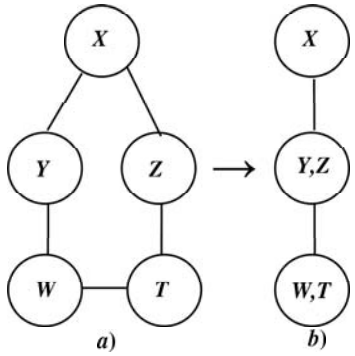


Fig. 3. a) An undirected graph with loop. b) Converting the graph with loop into a tree by the clustering method.

The most popular algorithm of BN inference is not Pearl's belief propagation but the "junction tree" method which was developed by (Jensen et al., 1990), based on the "clique-tree propagation algorithm" of (Lauritzen and Spiegelhalter, 1988). The "junction tree" algorithm transforms a multiply connected network into a clique tree[1]. Then it performs message propagation on the clique tree. However the complexity of the clique message propagation algorithm grows exponentially with the size of the largest clique of the tree.

[1] A clique tree is a graph where the nodes are cliques. A clique is a collection of vertices which are all pairwise neighbors.

Recently, the interest of researchers has returned to Pearl's belief propagation (BP) because they found that BP can efficiently perform an approximate inference on graphs with loops especially in the area of error correction coding and image processing (McEliece et al.,1998) (Murphy et al., 1999). However in some cases loopy BP may not converge or give the correct result. A new breakthrough (Yedidia et al., 2002) in understanding loopy BP occurred when a close connection was found between the BP algorithm and the Bethe free energy approximation in statistical physics. (Yedida et al., 2002) show that BP converges to a stationary point of an approximate free energy, known as the Bethe free energy in statistical physics. By using Kikuchi approximations (Kikuchi, 1951) and the cluster variation method (Morita, 1991), they propose a new generalized belief propagation (GBP) algorithm which always converges and gives much more accurate results than ordinary BP.

There are still many other exact and approximate methods which can not be described here, such as Markov Chain Monte Carlo (MCMC) sampling and symbolic probabilistic inference (Guo and Hsu, 2002).

2.3 Factor Graphs

A factor graph is a undirected graphic representation of a mathematical relation between local functions and variables. The factor graph comprises factor nodes, variable nodes and edges connecting factor nodes f_k and variable nodes x_j if and only if variable x_j is an argument of function f_k. The factor graph represents the product of local functions f_k as shown in equation 4.

$$g(x_1, x_2, ..., x_N) = \prod_{k \in K} f_k(X_k) \tag{4}$$

where X_k is a subset of $\{x_1, x_2, \ldots, x_N\}$ and $f_k(X_k)$ is a local function which has X_k as an argument. N is the number of variables and K is the number of local functions.

An example of a factor graph is shown in figure 4 which represents equation: $g(x_1, x_2, ..., x_4) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) f_D(x_3, x_4) f_E(x_3)$.

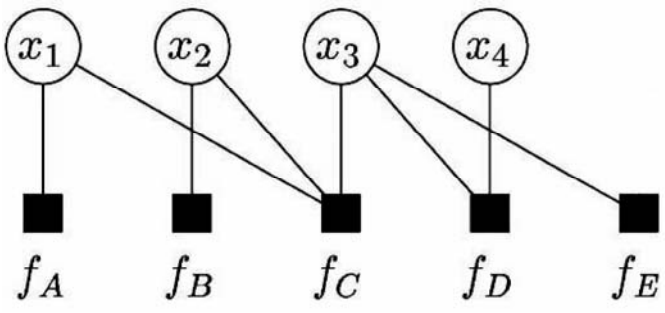


Fig. 4. An example of factor graph.

Similar to BN, the factor graph not only encodes the factorization of the global function in its structure, but also can arithmetically compute the marginal functions[2] associated with the global function $g(x_1, x_2, ..., x_N)$. There are several ways to perform the marginal operation.

[2] Associated with $g(x_1, x_2, ..., x_N)$, there are N marginal functions $g_i(x_i)$. The value of $g_i(x_i = a)$ is obtained by summing the value of $g(x_1, x_2, ..., x_N)$ over all possible configurations of the variable which have $x_i = a$.

For example one can apply the distributive law to the global function and represent the marginal function as order rooted trees (Rosen, 1999). Then the marginal operation can be performed either top-down (from root to leaves) or bottom-up (from leaves to root). However this method results in only one marginal function $g_i(x_i)$. In many cases we are interested in more than one marginal function. It is not efficient to recompute each value of $g_i(x_i)$ each time. (Kschischang et al., 2001) developed a message passing algorithm which computes all of the marginal functions $g_i(x_i), (i = 1, .., N)$ simultaneously. This algorithm involves the sum of products operation at function and variable nodes and is equivalent to belief propagation in BN.

Suppose a factor graph has a set of variable nodes $V = \{x\} \bigcup \{y_1, y_2, ..y_R\}$ and a set of function nodes $F = \{f\} \bigcup \{h_1, h_2, ..h_Q\}$ as shown in figure 5. The number of variable nodes is $R + 1$ and the number of function nodes is $Q + 1$. A set of neighbours of a function node f is $n(f)$ which is a subset of V and a set of neighbours of a variable node x is $n(x)$ which is a subset of F. Node x is a neighbour of node f. Let a function $\mu_{x \to f}(x)$ be the message sent from variable node x to function node f and a function $\mu_{f \to x}(x)$ be the message sent from function node f to variable node x.

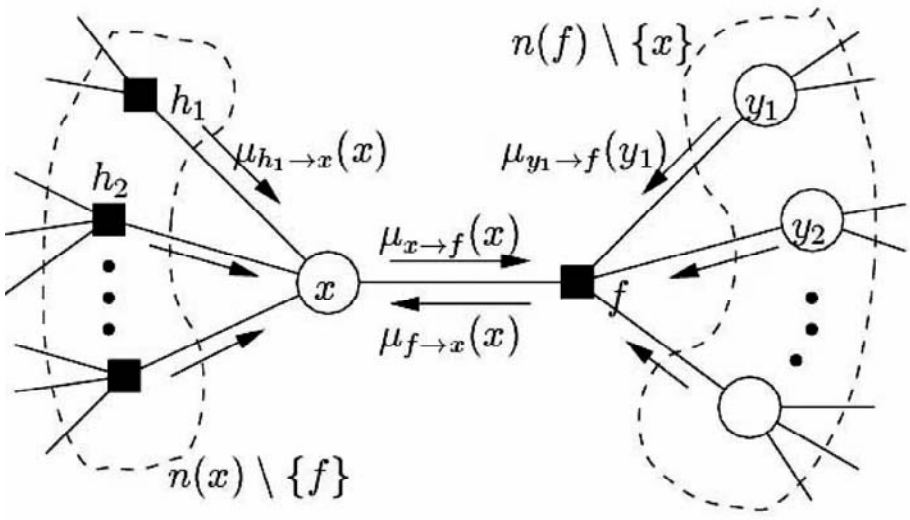


Fig. 5. Message passing algorithm in a factor graph (Kschischang et al., 2001).

According to (Kschischang et al., 2001), the message from a variable node to a function node is computed as follows:

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x) \tag{5}$$

The message from a function node to a variable node is computed as follows:

$$\mu_{f \to x}(x) = \sum_{X \setminus \{x\}} [f(X) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y)] \tag{6}$$

where $X = n(f)$ is the set of arguments of the function f.

While each node starts in the idle state, waiting for messages from its neighbours, the leaves and root initiate sending messages. If the leaf or root is a variable node, the initial message is equal to one. If the leaf or root is a function node, the initial message is equal to its local function associated with the node. A node x would be able to compute the sending message when it receives messages from its neighbours. The message sent to a neighbour f is calculated from all received messages except the message received from that neighbour f. After sending a message to the neighbour f, node x waits for a returned message from node f. When the returned message has arrived, node x can compute the message sent to other nodes (h). The algorithm stops when messages have been exchanged between two neighbours. The product of all incoming messages at a variable node x_j is the marginal function $g_j(x_j)$.

Similar to Loopy BP, while using the sum of product algorithm with a predetermined message schedule, factor graphs with cycles in general may not converge or may converge to the wrong result because they have the risk of over-counting information (Kschischang et al., 2001). However if the original factor graph contains loosely coupled loops, it usually converges and gives a very good approximate answer. Even though it is not well understood why factor graphs with cycles work, it has been practically demonstrated in many fields (McEliece et al.,1998) (Murphy et al., 1999).

2.4 Bayesian Networks represented by Factor Graphs

A Bayesian Network represents the joint probability defined as equation 1. According to (Kschischang et al., 2001), this equation can also be represented by a factor graph. The local function nodes are the conditional probabilities which have random variables X_k and $pa(X_k)$ as arguments. The function node $P(X_k|pa(X_k))$ is connected with variable nodes X_k and $pa(X_k)$. The joint probability of the lung cancer network, equation 2, is represented by a factor graph as shown in figure 6. In order to be comparable with figure 1, we rearrange figure 6 into figure 7. A node i of a Bayesian Network, corresponding to a dashed ellipse, is composed of a function node f_i and a variable node v_i. A message $\mu_{v_p \to f_c}(v_p)$ is a message sent from a variable node of a parent v_p to a function node of a child f_c and a message $\mu_{f_c \to v_p}(v_p)$ is a message sent from a function node of a child f_c to a variable node of a parent v_p. These messages are equivalent to the messages $\pi_c(p)$ and $\lambda_c(p)$ respectively in BP algorithm.

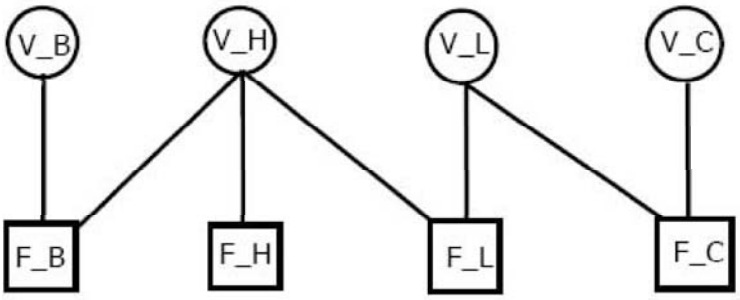


Fig. 6. A lung cancer Network represented by a factor graph.

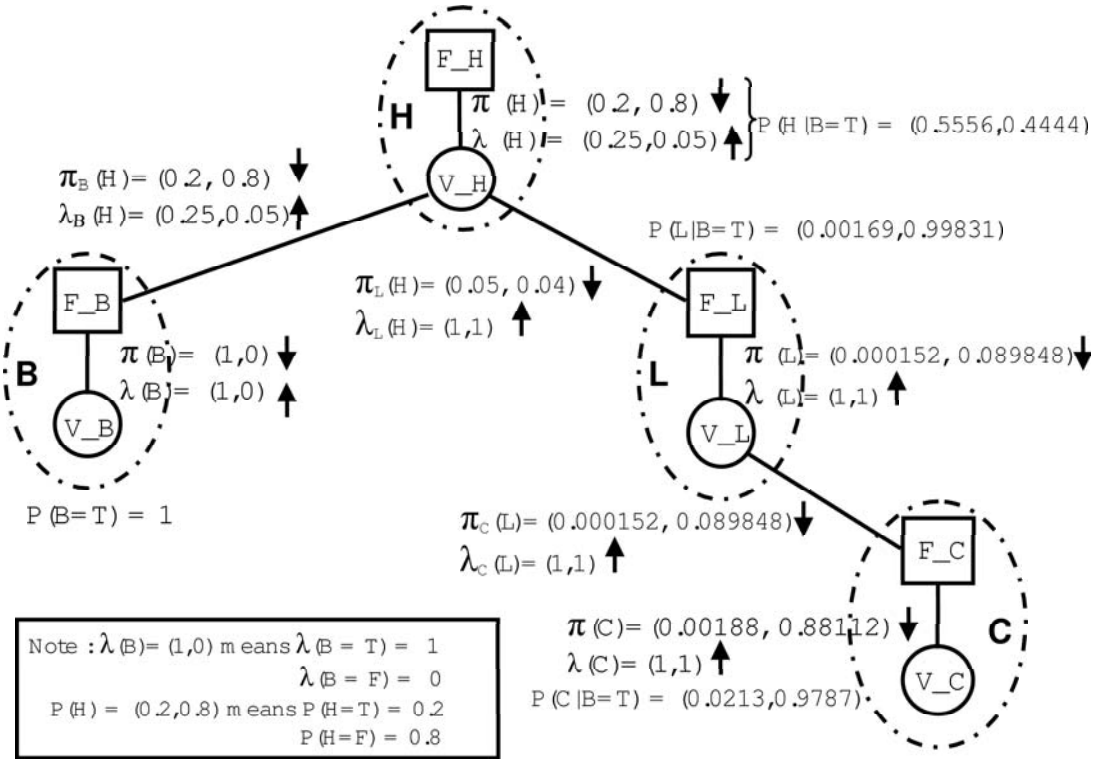


Fig. 7. A lung cancer Network represented by a factor graph and its message propagation.

The messages inside a dashed ellipse are $\mu_{v_i \to f_i}(v_i)$, sent upward from a variable node v_i to a function node f_i and $\mu_{f_i \to v_i}(v_i)$, sent downward from a function node f_i to a variable node v_i. These messages are equivalent to the messages $\lambda(v_i)$ and $\pi(v_i)$ respectively in BP algorithm.

Figure 7 also shows the message passing in the lung cancer network. The message values are computed as described in the previous section. The leaves and roots initiate sending messages. If there is evidence that a patient has bronchitis ($B = True$), the probability of ($B = true$) is equal to one. The node B initiates the message $\pi(B) = (1, 0)$ and $\lambda(B) = (1, 0)$ as shown in figure 7. The other messages is computed accordingly by equation 5 and 6. For example, we can compute $\lambda(H)$ and $\pi(L)$ by equation 7 and equation 8.

$$\mu_{V\_H \to F\_H}(H) = \lambda(H) = \lambda_B(H)\lambda_L(H) \tag{7}$$

$$\mu_{F\_L \to V\_L}(L) = \pi(L) = \sum_H (P(L|H)\pi_L(H)) \tag{8}$$

According to Pearl's algorithm (Pearl, 1988), the marginal probability is computed by

$$P(v_i) = \alpha\pi(v_i)\lambda(v_i). \tag{9}$$

The normalized factor α is used to make sure that summation of probability is equal to one. For example, marginal probability of lung cancer (L) can be computed by $P(L) = \alpha\lambda(L)\pi(L)$.

3 Description of Bayesian Network - CPN Model

We think the best way to understand our work is to explain the modelling together with an example. We model a BN representation of a lung cancer inference problem in figure 7 using Colour Petri Nets (CPNs). Our translating BNs to CPNs preserves all ability of Pearl's belief propagation. With loop cutset conditioning and node clustering methods, our work can be applied to any multiply connected Bayesian Networks with discrete variables, not limited to boolean variables, and gives an exact inference. With particular design of message scheduling, we also believe our model will help to model and analyse loopy belief propagation for approximate inference. However in this paper we limit our work to discrete variables and polytree Bayesian Networks.

Suppose we wish to model a Bayesian Network of figure 7 in order to infer the probability of lung cancer $P(L)$. An example question is if we know that a patient has bronchits, what is the chance he could have lung cancer? We expect to answer this question using CPN. Our model is organized into three hierarchical levels as shown in figure 8. The top level has one page named BN_Overview. The second level has one page named BN_NODE. The third level has two pages named FUNCTION and VARIABLE pages. The global declarations defining all the constants, and colour sets and declaring the types of the variables used in our model, are shown in figure 9.

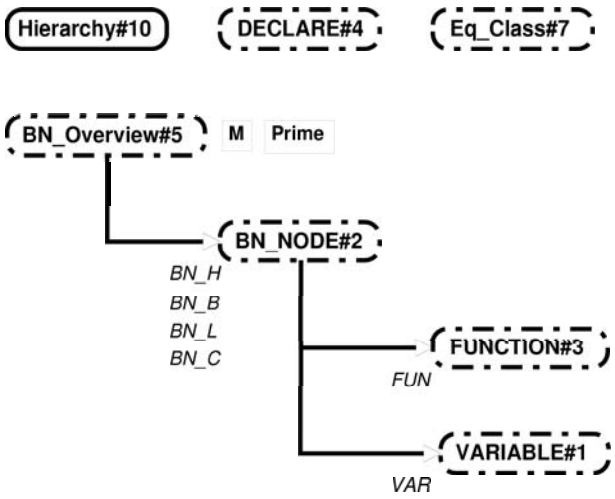


Fig. 8. Hierarchy page for the CPN model.

3.1 First Level Page

Figure 10 shows the top level page named BN_Overview. Each substitution transition models each discrete BN variable (BN_H, BN_L, BN_C and BN_B) which links to the second level page named BN_NODE page. Each BN_NODE sends a message to each other through socket places named IN_FUN and IN_VAR typed by *PACKET*. A message token typed by *PACKET*, is defined as a product comprising colour sets *FNODE*, *VNODE* and *LPAIR*. The colour sets *FNODE* and *VNODE* are used as the source and designation identifications.

```
(* --- Global Declaration Node --- *)
 val F_B = 1; val F_H = 2; val F_L = 3; val F_C = 4;
 val V_B = 1; val V_H = 2; val V_L = 3; val V_C = 4;
 val pr = 6;                (*Precision of Real Number *)
 color FNODE = int;         (*Function Node *)
 color VNODE = FNODE;       (*Variable Node  *)
 var vsrc1,vdst1:VNODE;
 var fsrc1,fdst1:FNODE;
 color LVNODE = list VNODE;  (* e.g. [V_B, V_L, V_C] *)
 var lv1,lvB:LVNODE;
 color LFNODE = list FNODE;
 var lf1,lfB:LFNODE;
 color LLFNODE = list LFNODE; (*List of (FNODE LIST)->[[F_L,F_B],[F_C,F_H]]*)
 var llf1,llv1:LLFNODE;
 color Real = IntInf;        (* define Real as Unbounded Integer  *)
 color X = int;              (* value of a variable e.g. V_L=2    *)
 color FX = Real;            (* value of a message mu(V_L=1) = 0.7*)
                             (* value of a message mu(V_L=2) = 0.3*)
 color PAIR = product X*FX;  (* from a pair of variable value and message value*)
 color LPAIR = list PAIR;
 var lpair1:LPAIR;           (*               [(1,0.7),(2,0.3)]    *)
 color PACKET  = product FNODE*VNODE*LPAIR;
                      (* mu(F_L->V_B)(V_B) = (F_L, V_B, [(1,0.7),(2,0.3)] )*)
 color VNODExLPAIR = product VNODE*LPAIR;     (* (V_B,[(1,0.7),(2,0.3)] )    *)
 color LMSG = list VNODExLPAIR;
                       (* Lists of messages which are stored in the nodes  *)
 var lmsg:LMSG;     (* [(V_B,[(1,0.7),(2,0.3)]), (V_H,[(1,0.2),(2,0.8)]) ] *)
 color VnX = product VNODE*X;        (* (V_H, 1) *)
 color LVnX = list VnX;              (* [ (V_H,1),(V_B,2),(V_L,1) ] *)
 color CPT = product LVnX*FX;   (* P(V_H=1|  V_B=2, V_L=1) = 0.2 *)
                                (*([(V_H,1),(V_B,2),(V_L,1)],0,2)*)
 color LCPT = list CPT;         (* List of 1CPT to form a conditional prob table*)
 var lcpt:LCPT;                 (* associate with a function node *)
```

Fig. 9. Global declarations of a lung cancer BN-CPN model.

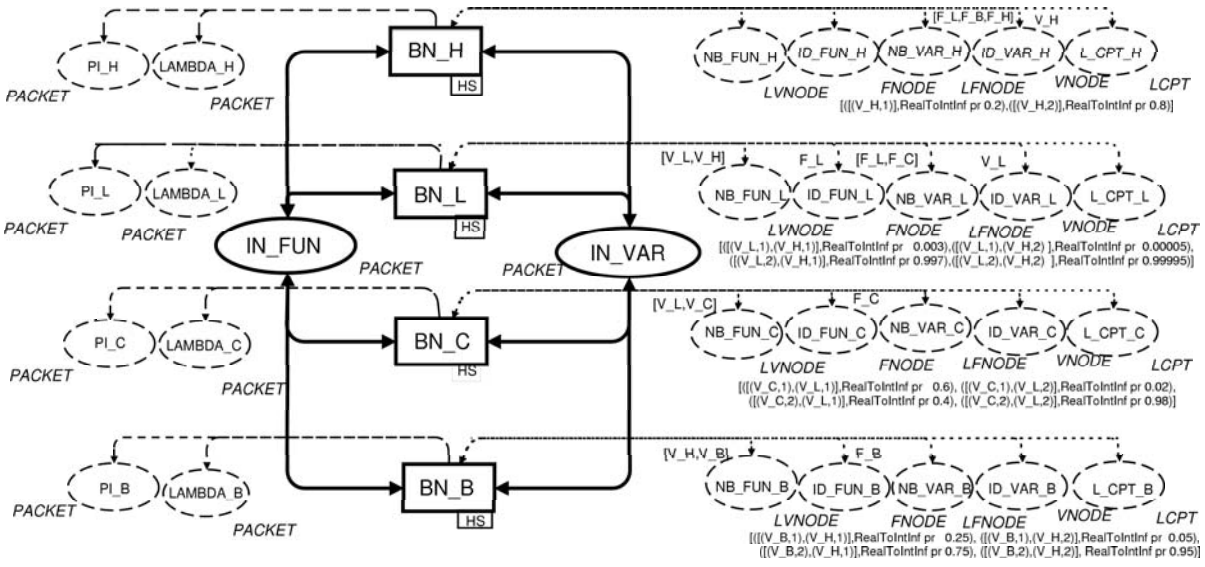


Fig. 10. The top level page: BN_Overview.

The colour set $LPAIR$, which is defined as a list of products (list of $PAIR$), models message values. One message value, typed by $PAIR$, is modelled with a product of an integer and a real number which represents a value of variable, typed by X, and a value of message, typed by FX. Although we define the colour set X as the integers to model a value of variable, the enumerated colour sets can also be used instead. We define a colour set FX as the long intergers to represent the real numbers as recommended by Design/CPN group (University of Aarhus, 2004).

3.2 Second Level Page

Figure 11 shows the second level page. The BN_NODE page models a BN node comprising two substitution transitions named FUN and VAR which model the function node and variable node respectively. A place named L_CPT stores initial markings, the Conditional Probability

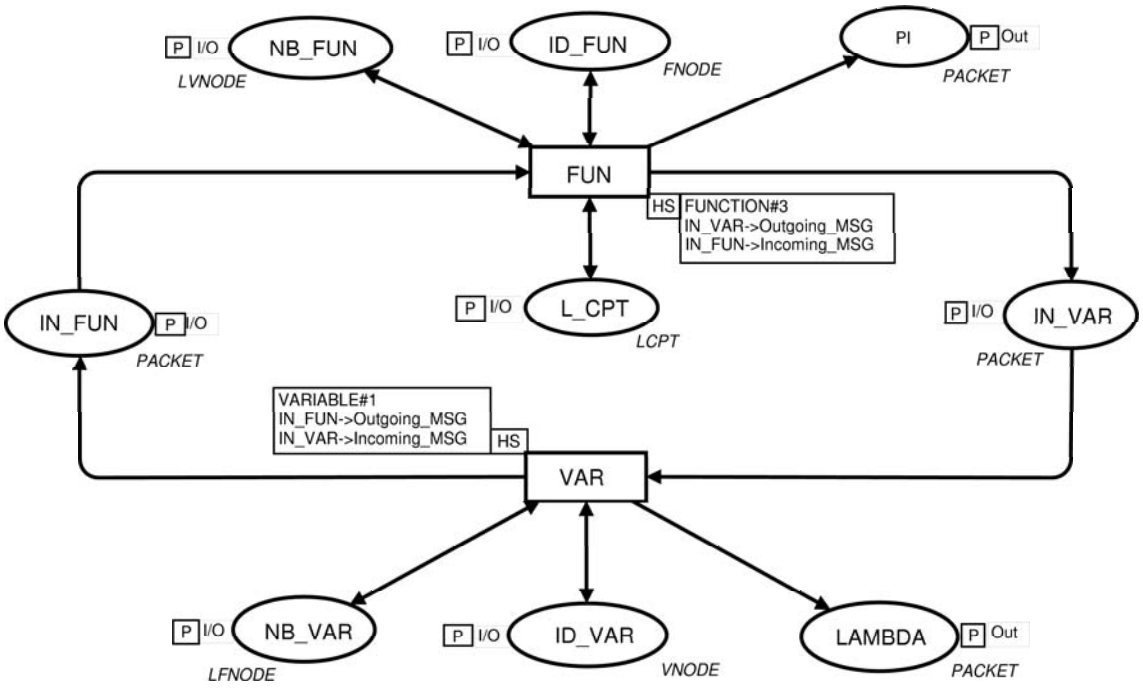


Fig. 11. The Second Level Page: BN_NODE PAGE.

Table (CPT) associated with the function node in the list format. Figure 12 shows the initial marking for the CPT which associated with function nodes in our example (figure 1). Place ID_FUN and place ID_VAR, store a node's identification (ID). For simplicity we use an integer to represent the node's ID. However the enumerated colour sets can be used to model the node's ID instead. Places NB_FUN and NB_VAR store a list of node IDs of adjacent nieghbour nodes. Figure 13 shows the initial marking in places ID_FUN and ID_VAR. The importance of this figure is that it encodes the structure of the Bayesian Network we model.

```
------------------------------------------------------------
BN_Overview'L_CPT_B 1:   1`[([(1,1),(2,1)],(ii ("250000"))),
                           ([(1,1),(2,2)],(ii  ("50000"))),
                           ([(1,2),(2,1)],(ii ("750000"))),
                           ([(1,2),(2,2)],(ii ("950000")))]
BN_Overview'L_CPT_H 1:   1`[([(2,1)],      (ii ("200000"))),
                           ([(2,2)],      (ii ("800000")))]
BN_Overview'L_CPT_L 1:   1`[([(3,1),(2,1)],(ii   ("3000"))),
                           ([(3,1),(2,2)],(ii     ("50"))),
                           ([(3,2),(2,1)],(ii ("997000"))),
                           ([(3,2),(2,2)],(ii ("999950")))]
BN_Overview'L_CPT_C 1:   1`[([(4,1),(3,1)],(ii ("600000"))),
                           ([(4,1),(3,2)],(ii  ("20000"))),
                           ([(4,2),(3,1)],(ii ("400000"))),
                           ([(4,2),(3,2)],(ii ("980000")))]
------------------------------------------------------------
```

Fig. 12. Initial marking for the CPT associated with function nodes.

```
----------------------------------------------------------------------
BN_Overview'ID_FUN_B 1: 1`1           BN_Overview'NB_FUN_B 1: 1`[2,1]
BN_Overview'ID_FUN_H 1: empty         BN_Overview'NB_FUN_H 1: empty
BN_Overview'ID_FUN_L 1: 1`3           BN_Overview'NB_FUN_L 1: 1`[3,2]
BN_Overview'ID_FUN_C 1: 1`4           BN_Overview'NB_FUN_C 1: 1`[3,4]

BN_Overview'ID_VAR_B 1: empty         BN_Overview'NB_VAR_B 1: empty
BN_Overview'ID_VAR_H 1: 1`2           BN_Overview'NB_VAR_H 1: 1`[3,1,2]
BN_Overview'ID_VAR_L 1: 1`3           BN_Overview'NB_VAR_L 1: 1`[3,4]
BN_Overview'ID_VAR_C 1: empty         BN_Overview'NB_VAR_C 1: empty
----------------------------------------------------------------------
```

Fig. 13. Initial marking represent node's ID and its neighbour.

3.3 Third Level Pages

There are two pages in the third level named FUNCTION and VARIABLE pages. Figure 14 shows the FUNCTION page which has two transitions named Load_MSG and Snd2Var as well as two places named Ptr_Buffer and MSG_Buffer. The others are port places which are linked to the upper level pages. The VARIABLE page as shown in figure 15 is similar to the FUNCTION PAGE but the FUNCTION page also has a place named L_CPT to store CPT values which are used to compute the outgoing messages by function LMsg2Var() (equation 6). The VARIABLE PAGE uses LMsg2Fun() (equation 5) instead to compute the outgoing messages. All ML functions used in these two pages are defined in the global declarations page but space limits prevent us from including the listing in figure 9. When the message is trans-

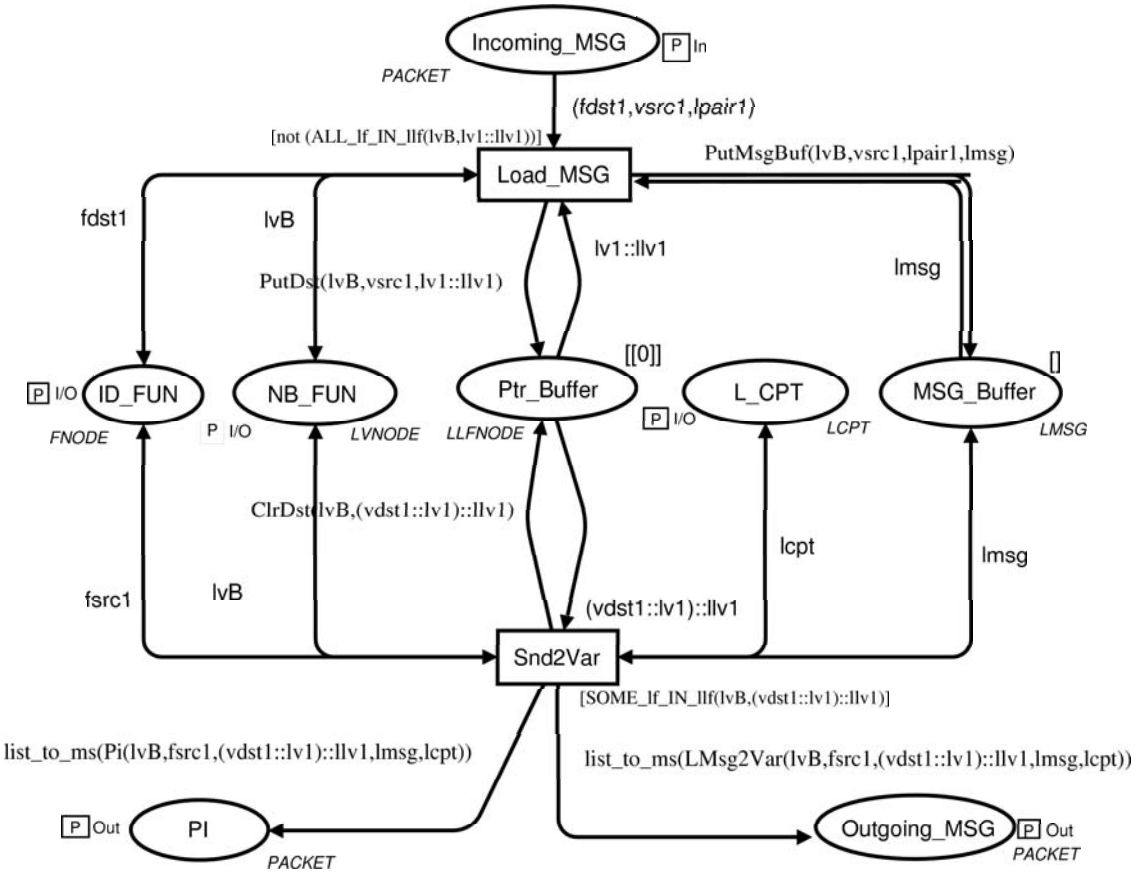


Fig. 14. The Third Level Page: FUNCTION PAGE.

ferred inside the BN node (dashed ellipse in figure 7), the message from the function to the variable node is the π value while the message from the variable to the function node is the λ value. We compute the π and λ values by functions Pi() and Lambda() and store these value in places PI and LAMBDA respectively. These two places are output port places and are assigned to socket places in second and top levels.

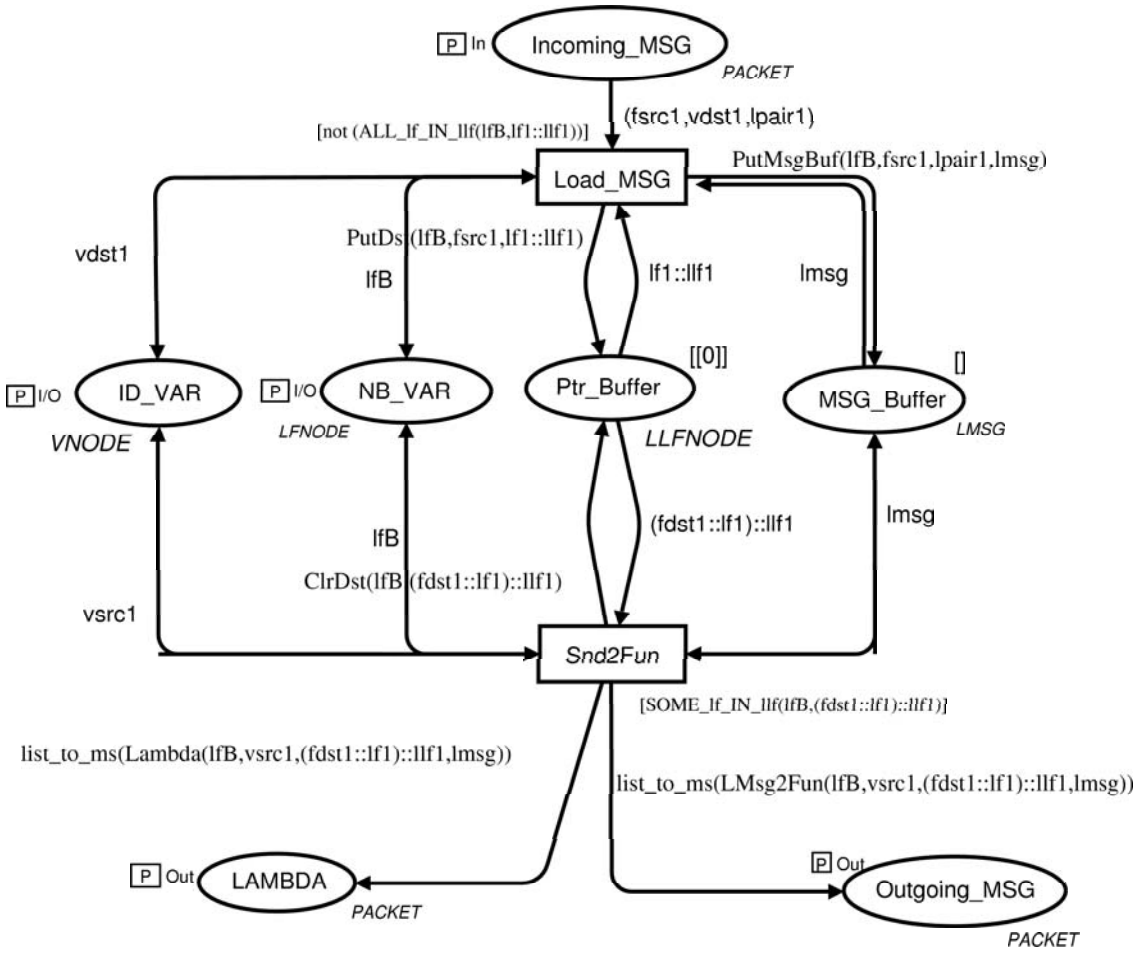


Fig. 15. The Third Level Page: VARIABLE PAGE.

4 Discussion of the 3-level CPN model

Our model structure has inspiration from packet switching hub used in the Internet. However the structure of our model (on the top level page) turns out to be different from traditional, graphical Bayesian Network models. While our CPN model loses the causal relationship representation, we believe it has more advantages. Because we encode the structure of BN into the tokens in place NB_VAR and NB_FUN, our CPN structure does not change if the BN structure changes given both models have the same variables. This will allow us to model structure learning where the structure of Bayesian Networks can be adapted.

While the first and second level pages are presented in order to give a clear view of the model, the actual operations are in the third level pages. We can imagine that there are boxes in the Ptr_Buffer place. The number of boxes is equal to the number of neighbours. Each box labels by the name of a neighbour. When the transition Load_MSG receives the PACKET, which has destination ID the same as its ID, it will put source ID into all boxes except the one which has the same label as the source ID. Also the message values with source ID are loaded into MSG_Buffer place. If the new source ID is the same as the one is already in the boxes, it will not add another source ID into the box but the old message values in MSG_Buffer place will be replaced by a new message. When all source IDs in a box plus its label v_i are equal

to the neighbour list, it means the node has all messages required to compute an outgoing message sending to the node v_i. After the transition Snd2Var computes the outgoing message according to equation 6 by using function LMsg2Var(), the corresponding box in Ptr_Buffer will be empty by function ClrDst().

If the node is not ready to take the incoming message tokens because it already has all incoming messages required, the guard function [not All_If_IN_llf()] will disable the transition Load_MSG. The transition will be enabled when at least one new message is required. Any new incoming messages will replace the old messages in the buffer. If the node does not have any outgoing messages to send because all outgoing messages have just been sent out, the guard function [SOME_If_IN_llf()]will disable the transition Snd2Var or Snd2Fun. This transition will be enabled when there is at least one message to be sent.

It is possible to fold FUNCTION page and VARIABLE page together. However the function node and variable node have different roles in factor graphs and thus two separate pages makes the model conceptually clearer, more understandable and flexible. Sometimes we also wish to do some experiments about message scheduling so that two separated pages give us more flexibility. It is also possible to fold transition Load_MSG and Snd2Fun together by which it can reduce the size of state space. But in this paper we wish to clearly demonstrate two separate operations, loading message into the buffer and computing the outgoing message according to equation 5 and 6.

5 Analysis of the BN-CPN Model

The lung cancer model is analyzed by simulation and generating occurrence graph (OG) using Design/CPN 4.0.5 (University of Aarhus, 2004) on a Pentium-III 1 GHz computer with 512 MB RAM. Various inference problems can be computed by instantiating the message values π and λ which are the initial markings (Packet) in places IN_VAR and IN_FUN (respectively) of the BN_Overview page. Figure 16 shows the initial markings comprising the messages when the question is, if we found that a patient has bronchitis, what is the chance he could have lung cancer?

```
-------------------------------------------------------------------------
BN_Overview'IN_VAR 1: 1`(2,2,[(1,(ii  ("200000"))),(2,(ii  ("800000")))])
                   ++ 1`(1,1,[(1,(ii ("1000000"))),(2,(ii       ("0")))])
BN_Overview'IN_FUN 1: 1`(1,1,[(1,(ii ("1000000"))),(2,(ii       ("0")))])
                   ++ 1`(4,4,[(1,(ii ("1000000"))),(2,(ii ("1000000")))])
-------------------------------------------------------------------------
```

Fig. 16. Initial marking comprising messages in places IN_VAR and IN_FUN.

Refer to figure 7, we set the initial message values of $\mu_{V\_C \to F\_C} = \lambda(C) = (1,1)$ because the message is initiated from the leaf which is a variable node. $\mu_{F\_H \to V\_H} = \pi(H) = P(H)$ because the message is initiated from the root which is a function node. When the evidence shows that the patient has bronchitis, we instantiate the evidence $V\_B = true$ so that these instantiation messages are $\mu_{(V\_B) \to F\_B} = \lambda(B) = (1,0)$ and $\mu_{(F\_B) \to V\_B} = \pi(B) = (1,0)$.

Using simulation, it takes 21 steps and gives the expected inference results. By generating the OG graph, there are 418 nodes and 699 arcs with 11 dead markings. Figure 17 shows the values of π and λ which are the output of the inference. These values can be verified with

the values in Figure 7. The value of $\pi(H)$, $\lambda(C)$, $\pi(B)$ and $\lambda(B)$ are the same as the initial markings in figure 16.

All dead markings give the same answer as shown in Figure 17 but the differences are the token values in the places Ptr_Buffer and MSGB_buffer. After we know the value of π and λ, we can compute the marginal probability $P(L|B = true)$ using equation 9. Because we

```
------------------------------------------------------------------------
BN_Overview'PI_H       1: empty
BN_Overview'LAMBDA_H   1: 1`(2,2,[(1,(ii  ("250000"))),(2,(ii   ("50000")))])
BN_Overview'PI_L       1: 1`(3,3,[(1,(ii     ("152"))),(2,(ii   ("89848")))])
BN_Overview'LAMBDA_L   1: 1`(3,3,[(1,(ii ("1000000"))),(2,(ii ("1000000")))])
BN_Overview'PI_C       1: 1`(4,4,[(1,(ii    ("1888"))),(2,(ii   ("88112")))])
BN_Overview'LAMBDA_C   1: empty
BN_Overview'PI_B       1: empty
BN_Overview'LAMBDA_B   1: empty
------------------------------------------------------------------------
```

Fig. 17. Output of the inference.

are aware of the state explosion problem that can occur when we apply our work to large problems, reducing the size of state space in the early stage of the work is very important. The tokens in the places Ptr_Buffer and MSGB_buffer are lists of data which does not require ordering. For instance, figure 18 shows the token values in in Ptr_Buffer and MSG_Buffer of a terminal state.

In other words, lists which have reordered elements are equivalent. Then it is possible to use occurrence graph with the equivalence class tool (OEOS) in Design/CPN to reduce the size of state space. To generate an OG with equivalences, we must define two functions; EquivMark() to detect equivalent markings and EquivBE() to detect equivalent bindings. EquivMark() performs tests on each place in the marking and performs a logical AND on the results to determine if the markings are equivalent. The EquivBE() function performs a similar test to EquivMark(), but it determines if input parameters of two binding elements are the same. If two bindings have the same transition and the values for the bindings are the same then they are equivalent. Using OEOS tools together with EquivMark() and EquivBE(), the state space is reduced to 154 nodes and 302 arcs with only one dead marking.

6 Contributions

This paper provides initial attempt to combine BNs with CPNs in order to model Agent's belief and preference in MAS. The contribution of the paper are the advantages that we gain from combining BNs with CPNs. These advantages are following;

1. Two important problems in multi-agent system design are agent design, which often uses the Artificial Intelligent (AI) techniques, and society design, which often uses the formal method and Petri Nets. This paper shows that reasoning or inference algorithms can be also modelled and analysed by Design/CPN.

2. The original belief updates and message passing algorithm are naturally parallel distributed over the network with simple control mechanisms and no timing information. Later several researchers have proposed modified versions of distributed message passing algorithms with more control mechanisms (Murphy et al, 1999), (Teh and Welling, 2001), (Yedidia et

```
------------------------------------------------------------------------
VARIABLE'MSG_Buffer 1:  1`[]
VARIABLE'MSG_Buffer 2:  1`[(3,[(1,(ii ("1000000"))),(2,(ii ("1000000")))]),
                           (1,[(1,(ii  ("250000"))),(2,(ii   ("50000")))]),
                           (2,[(1,(ii  ("200000"))),(2,(ii  ("800000")))])]
VARIABLE'MSG_Buffer 3:  1`[(4,[(1,(ii ("1000000"))),(2,(ii ("1000000")))]),
                           (3,[(1,(ii     ("152"))),(2,(ii   ("89848")))])]
VARIABLE'MSG_Buffer 4:  1`[]
------------------------------------------------------------------------
VARIABLE'Ptr_Buffer 1:  1`[[0]]
VARIABLE'Ptr_Buffer 2:  1`[[3],[1],[2]]
VARIABLE'Ptr_Buffer 3:  1`[[4],[3]]
VARIABLE'Ptr_Buffer 4:  1`[[0]]
------------------------------------------------------------------------
FUNCTION'MSG_Buffer 1:  1`[(1,[(1,(ii ("1000000"))),(2,(ii       ("0")))]),
                           (2,[(1,(ii  ("200000"))),(2,(ii  ("800000")))])]
FUNCTION'MSG_Buffer 2:  1`[]
FUNCTION'MSG_Buffer 3:  1`[(3,[(1,(ii ("1000000"))),(2,(ii ("1000000")))]),
                           (2,[(1,(ii   ("50000"))),(2,(ii   ("40000")))])]
FUNCTION'MSG_Buffer 4:  1`[(4,[(1,(ii ("1000000"))),(2,(ii ("1000000")))]),
                           (3,[(1,(ii     ("152"))),(2,(ii   ("89848")))])]
------------------------------------------------------------------------
FUNCTION'Ptr_Buffer 1:  1`[[1],[2]]
FUNCTION'Ptr_Buffer 2:  1`[[0]]
FUNCTION'Ptr_Buffer 3:  1`[[3],[2]]
FUNCTION'Ptr_Buffer 4:  1`[[4],[3]]
------------------------------------------------------------------------
```

Fig. 18. Token values in Ptr_Buffer and MSG_Buffer.

al., 2002). Because the nature of BP, applying known CPN modelling technique of concurrent systems to belief propagation algorithm is very well suited and gives flexibility when investigating alternative (new) message passing algorithms. We do not think of any other BN tools allow us modify message passing algorithms easily.

3. CPN analytic capability provides a basis for developing verification technique for probabilistic reasoning. The reachability graph also provides a clear picture how the message passing algorithm operates. Occurrence graph with the equivalence class tool can be used to assist validating inference algorithms. The occurrence graph is expected to have one dead marking with no livelock. In belief propagation with the loop cutset method, if there are other deadlocks, it means we have selected incorrect loop cutsets. The state space analysis can help us to discover this fault. Another example, if the state space has the livelock property, it means the iterative inference does not converge.

4. Bayesian networks are not only a well proven technology but also accumulated knowledge in this area is growing rapidly. Instead of inventing new types of Petri Nets, such as Possibilistic Petri Nets (Lee et al., 2003), Fuzzy Petri Nets (Shen, 2003), we propose to bring the enormous knowledge from the Bayesian Network paradigm to the Petri Net domain by translating BNs to CPNs in which inference capability of BNs is preserved. This will enhance the capabilities of CPNs, in terms of modelling probabilistic reasoning problems.

5. Other important abilities of BNs are parameter learning and structure learning from empirical data. At this stage, we do not expect to implement learning algorithm by ML

language. However we can use other tools such as MATLAB toolbox to learn parameter and structure of BNs from empirical data and then automatically translate them to CPN.

7 Conclusion and Future Work

In this paper, we have presented a CPN model of a polytree Bayesian Network with discrete variables. Our initial work models a BN inference algorithm, Pearl's belief propagation. To explain our model, we use a lung cancer Bayesian Network as an example. We were able to use equivalence class tools to further reduce the size of state space. We intend to proceed to model Agent's preference and decision models using CPNs and Influence Diagrams (IDs) which are an extension of Bayesian Networks. Due to the performance limitations of learning and inference in real time, agent belief and preference models need to have a small number of variable nodes (less than 8 nodes). Because of its lower complexity, we consider that Pearl's belief propagation and the loop cutset method are enough to model an agent's belief, especially when the root nodes are instantiated by evidence ($P(X = x_1) = 1$). The loop cutset method, which creates multiple graphs with the same structure but having different instantiated messages (figure 2), is very suitable for CPN modelling using substitute transition and folding.

In conclusion, we feel that the approach presented here of using Bayesian networks, Factor Graphs and Coloured Petri Nets will lead to further insights and techniques for designing multiagent decision support systems.

References

1. Cooper, G.F. (1990), *The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks.* Artificial Intelligent, vol. 42, pp. 393-405.
2. Dagum, P. and Luby, M. (1993), *Approximating Probabilistic Inference in Bayesian Belief Networks is NP-hard.* Artificial Intelligent, vol. 60, pp. 141-153.
3. German, R.(2000), *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets.* John Wiley & Sons, Ltd., West Sussex.
4. Guo, H. and Hsu, W. (2002), *A Survey of Algorithms for Real-Time Bayesian Network Inference.* AAAI/KDD/UAI-2002 Joint Workshop on Real-time Decision Support and Diagnosis Systems, Edmonton, Alberta, Canada.
5. Gyftodimos, E. and Flach, P. (2002), *Hierarchical Bayesian Networks: A Probabilistic Reasoning Model for Structured Domains.* In: Proceedings of the ICML-2002 Workshop on Development of Representations, Edwin de Jong and Tim Oates, editors, pp 23-30, University of New South Wales.
6. Haider, S. and Levis, A.H., (2004), *An Approximation Technique for Belief Revision in Timed Influence Nets.* In 2004 Command and Control Research and Technology Symposium, Loews Coronado Bay Resort San Diego, California.
7. Haider, S. and Zaidi, A. K., (2004), *Transforming Timed Influence Nets into Time Sliced Bayesian Networks.* In 2004 Command and Control Research and Technology Symposium, Loews Coronado Bay Resort San Diego, California.
8. Heckerman, D. (1995), *A Tutorial on Learning With Bayesian Networks.* Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washiton. ftp://ftp.research.microsoft.com/pub/dtg/david/tutorial.ps.
9. Jensen, F.V., Lauritzen, S.L., and Olesen, K.G. (1990), *Bayesian Updating in Causal Probabilistic Network by Local Computation.* Computational Statistical Quarterly, vol. 4, pp. 269-282.
10. Jensen, K. (1997), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use.* Volumes 1-3, Monographs in Theoretical Computer Science, Springer-Verlag, Berlin.
11. Kikuchi R. (1951), *A Theory of Cooperative Phenomena.* Phys. Rev., vol. 81, no. 6, pp. 988-1003.
12. Kim J.H. and Pearl, J.D. (1983), *A Computational Model for Causal and Diagnostic Reasoning in Inference Engines.* In Proceeding of the 8th International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, pp. 190-193.

13. Kschischang, F.R., Frey, B.J., and Loeliger, H. (2001), *Factor Graphs and the Sum-Product Algorithm.* IEEE Transction on Information Theory, vol. 47, no. 2, pp. 498-519.
14. Kschischang, F.R. (2003), *Codes Defined on Graph.* IEEE Communications Magazine, vol. 41, no. 8, pp. 118-125.
15. Kruse, R.J. and Lautenbach, K. (1998), *Bayessche Petri Netze.* Forschungsbericht, No. 694:5, Workshop Algorithmen und Werkzeuge für Petrinetze, pp. 67-72, Universität Dortmund, Fachbereich Informatik.
16. Lauritzen, S.L., and Spiegelhalter, D.J. (1988), *Local Computations with Probabilities on Graphical Structures and Their Applications to Expert Systems.* Proceedings of the Royal Statistical Society, Series B., vol 50, pp. 154-224.
17. Lee, J., Liu, K.F.R., and Chiang, W. (1999), *Modeling Uncertainty Reasoning with Possibilistic Petri Nets.* IEEE Trans. Systems, Man and Cybernetics-Part B:Cybernetics, vol. 33, no. 2, pp. 214-224.
18. Lindstrøm, B. and Haider, S. (2001), *Equivalent Coloured Petri Nets Models of a Class of Timed Influence with Logic.* In Proc. of Workshop and Tutorial on CPNs and CPN tools, pp 35-54. DAIMI PB-554, Aarhus University, Demnark.
19. Loeliger, H. (2004), *An Introduction to Factor Graphs.* IEEE Signal Processing Magazine, vol.21, no. 1, pp 28-41.
20. McEliece, R.J.,Mackay, D.J.C.,and Cheng, J. F. (1998), *Turbo Decoding as an Instance of Pearl's 'Belief Propagation' Algorithm.* IEEE J. Select. Areas Commun., vol. 16, no.2 , pp. 140-152.
21. Morita, T. (1991), *Cluster Variation Method for Non-uniform Ising and Heisenberg Models and Spin-pair Correlation Function.* Prog.Theor.Phys. vol.85, no. 2, pp. 243-255.
22. Murphy, K., Weiss, Y.,and Jordan, M. (1999), *Loopy-belief Propagation for Approximation Inference: An Emprical Study.* In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, pp. 465-475.
23. Neapolitan, R. E. (2004), *Learning Bayesian Networks.* Prentice Hall, Upper Saddle River, New Jersey.
24. Pearl, J.D. (1988), *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, California.
25. Pfeffer, A.J. (2000), *Probabilistic Reasoning for Complex Systems.* PhD Thesis, Stanford University.
26. Rosen, K.H. (1999), *Discrete Mathematics and its Applications.* fourth edition, McGraw-Hill, New York.
27. Shen, V.R. (2003), *Reinforcement Learning for High-Level Fuzzy Petri Nets.* IEEE Trans. Systems, Man and Cybernetics-Part B:Cybernetics, vol. 33, no. 2, pp. 214-224.
28. Teh, Y.W. and Welling, M. (2003), *Passing and Bouncing Messages for Generalized Inference.* GCNU TR 2001-001, Gatsby Computational Neuroscience Unit, University College London.
29. Wagenhals, L.W., Shin, I. and Levis, A.H. (1998), *Creating Executable Models of Influence Nets with Colored Petri Nets.* International Journal on Software Tools for Technology Transfer, vol. 2, no. 2, pp. 168-181.
30. Wagenhals, L.W. and Levis, A.H. (1999), *Converting Influence Nets with Timing Information to A Discrete Event System Model, A Colored Petri Net.* In Proc. of 2nd Workshop on Practical Uses of Colored Petri Nets and Design/CPN, DAIMI PB-532, Aarhus University, Denmark.
31. University of Aarhus (2004),
Design/CPN Online. http://www.daimi.au.dk/designCPN/.
32. Xiang, Y. and Lesser, V. (2003), *On the Role of Multiply Sectioned Bayesian Networks to Coorpeative Multiagent Systems.* IEEE Trans. Systems, Man, and Cybernetics-Part A, vol. 33, no.4, pp. 489-501.
33. Yedidia, J., Freeman, W.T. and Weiss, Y. (2002), *Understanding, Belief Propagation and its Generalizations.* Technical Report TR-2001-22, Mitsubishi Electric Research Laboratories, Inc., Massachusetts.

An Efficient Algorithm for the Enabling Test of Colored Petri Nets

Sami Evangelista and Jean-Francois Pradat-Peyre

CEDRIC - CNAM Paris
292, rue St Martin, 75003 Paris
{evangeli, peyre}@cnam.fr

Abstract. Model checking and simulation tools based on the colored Petri nets formalism spend a significant amount of time in performing enabling tests. This consists in taking into account the color mappings of the net to determine valid transitions variables assignments at a given marking. This work proposes an algorithm for the enabling test problem. It implements the relations of conflict and causality between transitions to efficiently maintain a set of enabled transitions. This set is updated during the search algorithm according to the transitions fired (or unfired). However, in most cases this approach is not sufficient to compute the set of enabled transition bindings, and has to be followed by a unification algorithm. This is the objective of the second part of this work.

1 Introduction

Colored nets provide model designers with the ability to express complex synchronizations patterns. The price to pay is that the analysis and simulation of such nets can be a difficult task. The possibility to unfold the net to directly analyze the unfolded ordinary net still exists but such an unfolding is not always possible because of large or even infinite color domains.
The firing rule of colored nets is made difficult by the management of colors of the net. When checking whether a transition is firable or not at a given marking, one has to find an assignment for the variables instantiated by the transition which respects the firing rule. This process is known under the term of enabling test.
A trivial solution is to check all the possible assignments and keep only the valid ones. This is only efficient if a large number of transitions instances is enabled at a marking. However, in most cases, this solution is not satisfactory.
We propose in this work an algorithm to deal with the enabling test problem. It includes two main features. Firstly, it implements the relations of conflict and causality (introduced by Haddad and Dutheillet in [1–3]) to manage a set of enabled transitions. This set is updated according to the fired (or unfired) transitions by the solving of some constraints systems. Secondly, as this approach is not always sufficient, we present in the second part of this work a unification algorithm inspired from the work of Mäkelä [14].

This work is organized as follows. Section 2 recalls some basic definitions on Petri nets and Colored Petri nets. Section 3 makes an informal presentation of the algorithm we propose. Section 4 recalls the concepts of conflict and causality. In Section 5 we detail an implementation of the conflict and causality relations. A unification algorithm is proposed in Section 6. A set of experimental results are presented in Section 7 to show the efficiency of our algorithm. At last Section 8 concludes our work.

2 Basic definitions

We recall here some basic definitions on ordinary Petri nets and Colored Petri nets. We also present the sub class of colored nets studied in this work.

Petri nets

Definition 1 (Petri net). *A Petri net is a tuple* $\langle P, T, W^-, W^+, m_0 \rangle$ *where* P *is a finite set of places;* T *is a finite set of transitions such that* $P \cap T = \emptyset$*;* W^- *and* W^+*, the backward and forward incidence matrixes are mappings from* $P \times T$ *to* $\mathbb{N}$*; and* m_0 *is a mapping from* P *to* $\mathbb{N}$*. The incidence matrix* W *is defined by* $W = W^+ - W^-$*.*

The definition of extended conflict and causality relations is based on the existence of some particular flows.

Definition 2 (Flow). *Let* $\langle P, T, W^-, W^+, m_0 \rangle$ *be a Petri net. A vector* $f \in \mathbb{Z}^P$ *is a flow if it fulfills* ${}^t f.W = 0$*; a positive flow if it fulfills* $\forall p \in P, f(p) \geq 0$*; a binary positive flow if it fulfills* $\forall p \in P, \Sigma_{p \in P}(f(p).m_0(p)) = 1$*.*

In the remainder, if f is a flow, we note $||f|| = \{p \in P \mid f(p) \neq 0\}$.

Colored Petri nets Analysis of colored Petri nets is based on multi-sets handling. Intuitively, a multi-set is a set which can contain several occurrences of the same item.

Definition 3 (Multi-set). *Let* s *be a set. A multi-set over* s *is a function from* s *to* $\mathbb{N}$*.* $Bag(s)$ *is the set of multi-sets over* s*.*

Multi-sets are usually noted as a linear combination of items. For instance, the multi-set $m = 4.a + 2.b + c$ over the set $\{a, b, c, d\}$ is the multi-set containing 4 occurrences of element a, 2 occurrences of element b, and 1 occurrence of element c. It is defined by $m(a) = 4$, $m(b) = 2$, $m(c) = 1$ and $m(d) = 0$. $m(a)$ is called the multiplicity of item a. 0 is the empty multi-set, i.e., $\forall e \in E, 0(e) = 0$.
The following operations may be defined for multi-sets :

Definition 4. *If* S *is a set and* a *and* b *are two elements of* $Bag(S)$ *then :*
$a + b = \sum_{x \in S}(a(x) + b(x)).x$
$a - b = \sum_{x \in S} \max(0, a(x) - b(x)).x$
$a \cap b = \sum_{x \in S} \min(a(x), b(x)).x$
$a \geq b \Leftrightarrow \forall x \in S, a(x) \geq b(x)$
$a > b \Leftrightarrow a \geq b \wedge \exists x \in S \mid a(x) > b(x)$

In colored Petri nets [10], tokens contained in places have colors, and places and transitions have color domains. Markings of colored nets associate to each place of the net a multi-set over its color domain.

Definition 5 (Colored Petri net). *A colored Petri net is a tuple* $\langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ *where* P *is a finite set of places;* T *is a finite set of transitions such that* $P \cap T = \emptyset$*;* $C : (P \cup T) \rightarrow \omega$ *is a color domain mapping;* ω *is a finite set of finite and non empty sets;* W^- *and* W^+ *associate to each couple* $(p, t) \in P \times T$ *a mapping from* $C(t)$ *to* $Bag(C(p))$*;* m_0 *associates to each* $p \in P$ *an element of* $Bag(C(p))$*; and* ϕ *associates to each* $t \in T$ *a guard, i.e., a mapping from* $C(t)$ *to* $\mathbb{B}$*, the set of booleans.*

The set of input (resp. output) places of a transition t is the set ${}^\bullet t = \{p \in P \mid W^-(p,t) > 0\}$ (resp. $t^\bullet = \{p \in P \mid W^+(p,t) > 0\}$). Similarly, the set of input (resp. output) transitions of a place p is the set ${}^\bullet p = \{t \in T \mid W^+(p,t) > 0\}$ (resp. $p^\bullet = \{t \in T \mid W^-(p,t) > 0\}$). These definitions are extended to set of places and transitions, i.e., for $P' \subseteq P, {}^\bullet P' = \cup_{p \in P'} {}^\bullet p$. In the remainder a place (p, c) with $p \in P$ and $c \in C(p)$ is called an instance of p. This also holds for transitions though the term binding is also used for transitions. The set of markings of a colored net N noted $\mathbb{M}_N$ is the set of mappings from P which map each place p to an element of $Bag(C(p))$.

Definition 6 (Colored firing rule). *Let* N *be a colored net and* $m \in \mathbb{M}_N$*. A transition* $t \in T$ *is firable at* m *with color* c_t *(denoted by* $m[(t, c_t)\rangle$*) if and only if :* $\phi(t)(c_t) \wedge \forall p \in P, m(p) \geq W^-(p,t)(c_t)$*. The marking* m' *obtained is defined by :* $\forall p \in P, m'(p) = m(p) + W(p,t)(c_t)$*. In this case, we note* $m[(t, c_t)\rangle m'$*.*

The set of transitions enabled at a marking m is the set $En(m)$. $Reach(N) \subseteq \mathbb{M}_N$, the set of reachable markings of N is recursively defined as $\{m_0\} \cup \{m \in \mathbb{M}_N \mid \exists m' \in Reach(N), t \in T, c_t \in C(t) \mid m'[(t, c_t)\rangle m'\}$.

The definition of some operations on color mappings may be useful. We first extend mappings from C to $Bag(C')$ to mappings from $Bag(C)$ to $Bag(C')$ by the following rules :

- $f(\lambda.c) = \lambda.f(c)$
- $f(c_1 + c_2) = f(c_1) + f(c_2)$

Definition 7. *If* f *is a mapping from* $Bag(C'')$ *to* $Bag(C')$*, and* g *is a mapping from* $Bag(C)$ *to* $Bag(C'')$ *then* $f \circ g$ *is a mapping from* $Bag(C)$ *to* $Bag(C')$ *defined by* $\forall c \in C, c' \in C', (f \circ g)(c)(c') = \sum_{c'' \in C''} f(c'')(c').g(c)(c'')$*.*

Definition 8. *If* f *is a mapping from* $Bag(C)$ *to* $Bag(C')$*, then* ${}^t f$ *is a mapping from* $Bag(C')$ *to* $Bag(C)$ *defined by* $\forall c \in C, c' \in C', {}^t f(c')(c) = f(c)(c')$*.*

Colored positive flows are defined below.

Definition 9 (Colored positive flow). *Let $\langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net. A positive flow f with color domain $C(f)$ is a vector over P, noted as the formal sum $f = \Sigma_{p \in P} f_p.p$ such that $\forall p \in P, f_p$ is a mapping from $Bag(C(p))$ to $Bag(C(f))$ and such that $\forall t \in T, \Sigma_{p \in P} f(p) \circ W(p,t) = 0$. A colored positive flow is a colored binary positive flow if*
$\forall c_f \in C(f), \Sigma_{p \in P} f_p(m_0(p))(c_f) = 1$.

In the next sections we will often refer to the unfolded net of the colored Petri net. It is defined below.

Definition 10 (Unfolded net). *Let $N = \langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net : $N_u = \langle P_u, T_u, {W^-}_u, {W^+}_u, {m_0}_u \rangle$, the unfolded net of N is the Petri net defined by :*

- $P_u = \{(p, c_p) \mid p \in P, c_p \in C(p)\}$
- $T_u = \{(t, c_t) \mid t \in T, c_t \in C(t) \wedge \phi(t)(c_t)\}$
- $\forall (p, c_p) \in P_u, (t, c_t) \in T_u, {W^-}_u((p, c_p), (t, c_t)) = W^-(p, t)(c_t)(c_p)$
- $\forall (p, c_p) \in P_u, (t, c_t) \in T_u, {W^+}_u((p, c_p), (t, c_t)) = W^+(p, t)(c_t)(c_p)$
- $\forall (p, c_p) \in P_u, {m_0}_u((p, c_p)) = m_0(p)(c_p)$

A sub class of Colored Petri nets In the sub class under study, color domains are cartesian products of basic finite sets called color classes. Color mappings are linear combinations of simpler mappings called tuples. Tuples are cartesian products of simple expressions. Three kinds of expressions are allowed : a variable X of the corresponding transition, a constant, and any user defined mapping f which parameters are valid expressions. At last, a guard can be any boolean expression.

Example 1. $X, 3, f(X)$ are valid expressions.
$\langle X, 3, f(X, Y) \rangle$ is a tuple.
$2.\langle X, 3, f(X, Y) \rangle + 3.\langle 5, X, g(Y) \rangle$ is a color mapping.

In our sense, this class of colored nets can cover a large range of practical models. However, it could be considered in future works to enrich it with the following features :

- broadcast mapping of well formed nets
- guarded tuples, e.g., $[X > Y]\langle X, Y \rangle$
- variable multiplicity tuples, e.g., $X.\langle X, Y, 0 \rangle$

The definition of an enabling test algorithm highly depends on the class of colored nets considered. Numerous works exploit the good structuring of color mappings of well formed nets to define specific optimizations techniques : [5], [8], [18]. For instance, Illié and Rojas use the reversibility of well formed color mappings to find transitions instances linked to a marked place. The type of nets we consider, though inspired from well formed nets, has a major difference with these ones. Indeed, the possibility to allow any user defined mapping in arc expressions disable the reversiblity of color mappings. Thus, our class seems closer to the algebraic nets used by Mäkelä [14], even if it does not include all its features such as variable multiplicity arcs.

3 Informal presentation of the algorithm

Let us consider a colored net. At a given marking m the set of enabled bindings is $En(m)$. The firing of a transition (t, c) of $En(m)$ leads to a new marking m'. If we compare $En(m)$ with $En(m')$ we can make the following observations :

- Only the instances in conflict with (t, c), i.e. which need tokens consumed by this one are in $En(m)$ but not in $En(m')$. These are instances of transitions which belong to the set $({}^\bullet t)^\bullet$.
- Instances (t', c') which are in $En(m')$ but not in $En(m)$ are such that $t' \in (t^\bullet)^\bullet$. Indeed, the newly enabled transitions needed tokens produced by (t, c).

On the basis of this locality principle, we can reasonably think that these two sets are quite close. The algorithm we propose is based on this simple observation. Basically, instead of recomputing at each encountered marking the set of enabled bindings, we maintain a set of enabled bindings which is updated according to the transitions fired (or unfired) during the search algorithm. Figure 1 presents a basic depth first search algorithm based on this principle. This one operates on a global set of reached markings *reached* initialized to the empty set, and the set of enabled transitions *enabled* initialized to $En(m_0)$. Procedure $fire$ (respectively $unfire$) updates m by firing (unfiring) (t, c) and updates the set *enabled*. To achieve this, it proceeds in two steps. Firstly, it removes the instances disabled by (t, c). Seconly, it adds the instances which have been enabled by inspecting the tokens produced by the firing.
To manage these two steps we use the relations of conflict and causality. These concepts are recalled in the next section.

Note. Unfiring an instance (t, c) raises no difficulty. It is equivalent to firing an instance (t', c) such that $C(t') = C(t)$, $\forall p \in P, W^-(p, t') = W^+(p, t)$ and $W^+(p, t') = W^-(p, t)$.

```
DFS (in out marking m)
1   if m ∉ reached then
2      reached ← reached ∪ {m}
3      for (t, c) ∈ enabled do
4         FIRE((t, c), m)
5         DFS(m)
6         UNFIRE((t, c), m)
7      endfor
8   endif
FIRE (in transition binding (t, c), in out marking m)
1   REMOVE_DISABLED(m, (t, c), enabled)
2   ADD_ENABLED(m, (t, c), enabled)
3   m ← m[(t, c)⟩
```

Fig. 1. A depth first search algorithm

4 Conflict and causality relations

Causality and conflict relations can be used to update the set of enabled transitions. The first one is helpful to determine instances disabled by a firing whereas the second one is useful to check instances enabled by a firing. We also propose to refine these relations by taking into account the flows of the net. For each relation we first give an "ordinary version" of it, i.e., defined on ordinary Petri nets after generalizing it to colored Petri nets. To achieve this, we first have to give some definitions and notations on powersets.

Definition 11. *If S is a set, then $\mathcal{P}(S)$, the powerset of S is the set of sub sets of S, i.e., $s \in \mathcal{P}(S) \Leftrightarrow s \subseteq S$.*

Definition 12. *Let $f \in S \to Bag(S')$. $\overline{f} \in S \to \mathcal{P}(S')$ is defined by $\forall s \in S, \overline{f}(s) = \{s' \in S' \mid f(s)(s') > 0\}$.*

A function from S to $\mathcal{P}(S')$ can be extended to a function from $\mathcal{P}(S)$ to $\mathcal{P}(S')$ by the following rule : $f(\{c_1, c_2\}) = f(c_1) \cup f(c_2)$. Transpositions and composition of functions on powersets can now be defined.

Definition 13. *Let $f \in \mathcal{P}(S) \to \mathcal{P}(S')$. ${}^t f \in \mathcal{P}(S') \to \mathcal{P}(S)$ is defined by : $s \in {}^t f(s') \Leftrightarrow s' \in f(s)$*

Definition 14. *Let $f \in \mathcal{P}(S'') \to \mathcal{P}(S')$ and $g \in \mathcal{P}(S) \to \mathcal{P}(S'')$. $f \circ g \in \mathcal{P}(S) \to \mathcal{P}(S')$ is defined by : $(f \circ g)(s) = f(g(x))(s)$*

4.1 Conflict relation

Two transitions t and t' are in conflict if there is a reachable marking in which both are firable and the firing of t disables the firing of t'. This is given by the relation CO.

Definition 15 (Conflict relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary conflict relation $CO \subseteq T \times T$ is defined by*

$$(t, t') \in CO \Leftrightarrow \exists m \in Reach(N) \mid m[t\rangle m' \wedge m[t'\rangle \wedge \neg m'[t'\rangle$$

As this definition requires the generation of the reachability set of N it has no practical interest. To approximate this relation we define the relation of ordinary structural conflict which only relies on the structure of the net. A transition t may disable another transition t' only if they share a same input place p such that t decreases the marking of p. We can easily prove that $CO \subseteq SCO$.

Definition 16 (Structural conflict relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary structural conflict relation $SCO \subseteq T \times T$ is defined by*

$$(t, t') \in SCO \Leftrightarrow \exists p \in {}^\bullet t' \mid W(p, t) < 0$$

Thus, it holds that the only transitions t' disabled by the firing of a transition t are such that $(t,t') \in SCO$. We can go a step further and give a more accurate approximation of CO if we have some knowledge of the flows of the net. Indeed, if a binary positive flow covers both p and q respectively inputs of t and t', we can state that t cannot disable t' since t and t' cannot be concurrently enabled. This is given by the relation $ESCO$ defined below. Once again, it trivially holds that $CO \subseteq ESCO \subseteq SCO$.

Definition 17 (Extended structural conflict relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary extended structural conflict relation $ESCO \subseteq T \times T$ is defined by*

$$(t,t') \in ESCO \Leftrightarrow (t,t') \in SCO \wedge$$

$$\nexists f \in F, (p,q) \in ({}^\bullet t) \times ({}^\bullet t') \mid p \neq q \wedge \{p,q\} \subseteq ||f||$$

where F is the set of binary positive flows of N.

We recall now the colored version of the structural conflict relation which has been given by Dutheillet and Haddad in [3].

Definition 18 (Colored structural conflict relation). *Let $N = \langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net and $t, t' \in T$. The colored structural conflict relation $CSCO(t,t') \subseteq C(t) \times C(t')$ is defined by*

$$(c,c') \in CSCO(t,t') \Leftrightarrow c' \in \bigcup_{p \in P} \overline{{}^t W^-(p,t')} \circ \overline{(W^-(p,t) - W^+(p,t))}(c)$$

The mapping $\overline{(W^-(p,t) - W^+(p,t))}$ gives the instances (p,c_p) which marking is decreased by the firing of (t,c). By composing this mapping with $\overline{{}^t W^-(p,t')}$ we obtain the instances (t',c') which need these tokens to be firable. Thus the "unfolding" of relation $CSCO(t,t')$ produces the set of couples (c,c') such that $((t,c),(t',c'))$ belong to the relation SCO in the unfolded net.

Generalizing the extended conflict relation does not raise any difficulty. The problem is that it is currently admitted that the computation of flows of colored Petri net is a difficult task. However, we believe that the model designer can provide a model checking tool with some basic flows of the net which correspond to the modeled entities (e.g., process, critical sections). The colored extended structural conflict relation $CESCO$ is given below.

Definition 19 (Colored extended structural conflict relation). *Let $N = \langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net and $t, t' \in T$. The colored extended structural conflict relation $CESCO(t,t') \subseteq C(t) \times C(t')$ is defined by*

$$(c,c') \in CESCO(t,t') \Leftrightarrow (c,c') \in CSCO(t,t') \wedge$$

$$\nexists f \in F, (p,q) \in ({}^\bullet t) \times ({}^\bullet t') \mid p \neq q \wedge c' \in \overline{{}^t W^-(q,t')} \circ \overline{{}^t f(q)} \circ \overline{f(p)} \circ \overline{W^-(p,t)}(c)$$

where F is the set of colored binary positive flows of N.

By composing $\overline{W^-(p,t)}$ with $\overline{f(p)}$ we obtain the instances (f, c_f) of flow f which cover (p, c_p). A third composition with $\overline{{}^t f(q)}$ gives us the instances (q, c_q) also covered by (f, c_f). Finally, the composition of this mapping with $\overline{{}^t W^-(q,t')}$ produces the instances (t', c') that are in mutual exclusion with (t, c).

Example 2. An example of extended structural conflict is given on figure 2. We easily see that (t, c) is in conflict with (t', c') by place (r, c_r). Let us suppose now that a binary positive flow covers both (p, c_p) and (q, c_q). With the help of this flow we can state that (t', c') will not be disabled by (t, c) as these two transitions cannot be concurrently enabled.

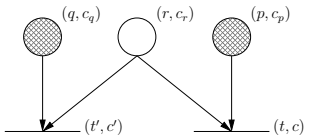


Fig. 2. An illustration of the extended structural conflict relation

4.2 Causality relation

Two transitions t and t' are causally connected if there is a reachable marking m in which t is enabled, t' is disabled and the firing of t enables t'.

Definition 20 (Causality relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary causality relation $CA \subseteq T \times T$ is defined by*

$$(t, t') \in CA \Leftrightarrow \exists m \in Reach(N) \mid m[t\rangle m' \wedge \neg m[t'\rangle \wedge m'[t'\rangle$$

As for the conflict relation, the structure of the net can help us to approximate this relation by the structural causality relation SCA. Trivially, we have $CA \subseteq SCA$.

Definition 21 (Structural causality relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary causality relation $CA \subseteq T \times T$ is defined by*

$$(t, t') \in SCA \Leftrightarrow \exists p \in {}^\bullet t' \mid W(p, t) > 0$$

Once again we can make use of place flows to give a more accurate approximation of the causality relation. If a binary positive flow covers a place p output of t and a place q input of t' then we can statically determine that t' is not causally connected to t. We use the term of extended causality to define this relation.

Definition 22 (Extended structural causality relation). *Let $N = \langle P, T, W^-, W^+, m_0 \rangle$ be a Petri net. The ordinary extended structural causality relation $ESCA \subseteq T \times T$ is defined by*

$$(t, t') \in ESCA \Leftrightarrow (t, t') \in SCA \wedge$$

$$\nexists f \in F, (p, q) \in (t^\bullet) \times ({}^\bullet t') \mid p \neq q \wedge \{p, q\} \subseteq ||f||$$

where F is the set of binary positive flows of N.

We recall now the colored version of [3] of structural causality relation, and we define the colored extended causality relation.

Definition 23 (Colored structural causality relation). *Let $N = \langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net and $t, t' \in T$. The colored structural causality relation $CSCA(t, t') \subseteq C(t) \times C(t')$ is defined by*

$$(c, c') \in CSCA(t, t') \Leftrightarrow c' \in \bigcup_{p \in P} \overline{{}^t W^-(p, t')} \circ (\overline{W^+(p, t) - W^-(p, t))}(c)$$

Definition 24 (Colored extended structural causality relation). *Let $N = \langle P, T, C, W^-, W^+, \phi, m_0 \rangle$ be a colored Petri net and $t, t' \in T$. The colored extended structural causality relation $CESCA(t, t') \subseteq C(t) \times C(t')$ is defined by*

$$(c, c') \in CESCA(t, t') \Leftrightarrow (c, c') \in CSCA(t, t') \wedge$$

$$\nexists f \in F, (p, q) \in (t^\bullet) \times ({}^\bullet t') \mid p \neq q \wedge c' \in \overline{{}^t W^-(q, t')} \circ \overline{{}^t f(q)} \circ \overline{f(p)} \circ \overline{W^+(p, t)}(c)$$

where F is the set of colored binary positive flows of N.

Example 3. Figure 3 illustrates the notions of causality and extended causality. By place (r, c_r), (t', c') is causally connected to (t, c), i.e. the firing of transition (t, c) can potentially enable (t', c'). Suppose now that a binary positive flow covers both (p, c_p) and (q, c_q). With the help of this flow we can statically state that (t', c') cannot be enabled by (t, c) as this one puts a token in (p, c_p) (and so (q, c_q) cannot contain a token).

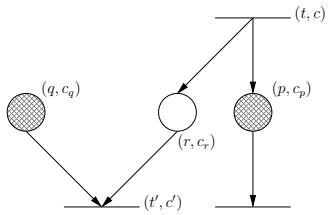


Fig. 3. An illustration of the extended causality conflict relation

5 Implementing conflict and causality relations

The idea of exploiting the locality principle of Petri nets in order to efficiently manage a set of enabled transitions is not new and has been investigated by Gaeta [18], Mortensen [12], and Haagh and Hansen [21] to get an efficient simulation engine. Gaeta's algorithm also relies on the relations of conflict and causality. However, his implementation proceeds transitions instance by instance which is equivalent to unfold the net. In our sense, this solution is not satisfactory for large color domains. In this section we focus on a way to detect such instances without enumerating them.
In a previous work [19] we have presented a general framework based on an initial work of Brgan and Poitrenaud [17]. The basic idea is to study the dependencies between places and transitions of the net at a symbolic level instead of unfolding the net. For that we translate color mappings into equivalent constraints systems which can be built and reduced before the state space generation and repeatedly solved during the search (or simulation) algorithm to identify disabled and enabled transitions.

5.1 Translating color mappings to constraints systems

If f is a color mapping (as it is defined in section 2) from C to $Bag(C')$, then the mapping $\overline{f}$ can be translated into an equivalent constraints system. For instance if we consider the mapping $f = 2.\langle X, Y, 0, X+1\rangle + 3.\langle Y, X, g(X), 2\rangle$, the items $\langle a, b, c, d\rangle$ which belong to the image of $\langle x, y\rangle$ by $\overline{f}$ satisfy the following constraint

$$[a = x] \ \wedge \ [b = y] \ \wedge \ [c = 0] \ \wedge \ [d = x + 1]$$

$$\vee$$

$$[a = y] \ \wedge \ [b = x] \ \wedge \ [c = g(x)] \ \wedge \ [d = 2]$$

By this way, each basic mapping operation such as composition, or transposition can be translated into a constraints system. For space constraints, we do not give here these systems, but we invite the reader to refer to [19].

5.2 Implementing conflict relation

To implement conflict relation between transitions, we build (and simplify) for each couple of transitions (t, t') such that $t' \in ({}^\bullet t)^\bullet$ the constraints system corresponding to $CSCO(t, t')$ (or $CESCO(t, t')$ if the flows of the net are known). During the search algorithm, each time (t, c) is fired we remove from the set of enabled transitions the set of instances (t', c') such that c and c' satisfy the constraints system.

Let us see with an example (figure 4) how we proceed. Suppose we want to check if a transition $(t', \langle X_{t'}, Y_{t'}\rangle)$ is disabled by the firing of $(t, \langle X_t, Y_t\rangle)$. We

first suppose that no binary flow can help us in this task. The constraints system corresponding to $CSCO(t, t')$ is :

$$(X_r = X_t \wedge Y_r = Y_t) \ \wedge \ \neg(X_r = X_t \wedge Y_r = f(Y_t)) \ \wedge \ (X_{t'} = X_r \wedge Y_{t'} = Y_r)$$

where X_r and Y_r respectively denote the first and second components of the color domain of r. We simplify this system in

$$X_{t'} = X_t \wedge Y_{t'} = Y_t \wedge Y_{t'} \neq f(Y_t)$$

Thus, each time a transition $(t, \langle X_t, Y_t \rangle)$ is fired, the only instances $(t', \langle X_{t'}, Y_{t'} \rangle)$ which can potentially be removed from the set of enabled transitions are those which satisfy this system. If the net is safe, we can erase these transitions from the set of enabled transitions without looking at the current marking. If it is not the case, we have to check that these are still firable, as there may still be enough tokens in the input places.

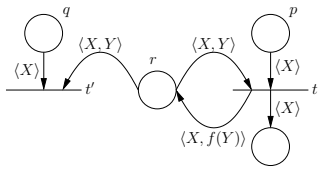


Fig. 4. An illustration of the resolution of conflicts

Suppose now that we have a binary flow f covering p and q and such that $f = \langle X \rangle.p + \cdots + \langle X \rangle.q$, i.e., the same token cannot be at the same time in places p and q. The system corresponding to $CESCO(t, t')$ is :

$$(X_{t'} = X_t \wedge Y_{t'} = Y_t \wedge Y_{t'} \neq f(Y_t)) \wedge \neg(X_t = X_p \wedge X_p = X_f \wedge X_f = X_q \wedge X_{t'} = X_q)$$

The first part of the conjunction corresponds to $CSCO(t, t')$. The second part is provided by f and is reduced to $X_t = X_{t'}$. We easily detect an inconsistency in the system$(X_t = X_{t'} \wedge X_t \neq X_{t'})$ and we can conclude that an instance (t', c') cannot be disabled by the firing of a (t, c). Thus, each time a transition (t, c) is fired, we do not have to look in the set of enabled transitions for a disabled instance of t'.
Note that building $CESCO$ is only useful when the system constructed is inconsistent. In other cases, it may introduce useless computations, as building $CSCO$ is sufficient to check the disabled transitions.

5.3 Implementing causality relation

As for the conflict relation, we construct for each transitions couple (t, t') such that $t' \in (t^\bullet)^\bullet$ the constraints system corresponding to $CESCA(t, t')$. Each time

an instance (t, c) is fired the resolution of the system gives us the set of instances (t', c') which are enabled by (t, c). However, in most cases, this is not sufficient to find a complete assignment of the variables of t'. For example let us take net depicted on figure 5. The firing of transition t with assignment $[X = x, Y = y]$ could potentially enable the transition t' with assignment $[X = x, Y = y, W = ?, Z = ?]$, but we still have to check that a token is present in place q to bind variables W and Z. So in most cases, the causality relation only help us to find a partial assignment for the transitions causally connected to the fired transition. A unification algorithm [14] is a possible way to complete these assignments.

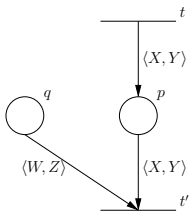


Fig. 5. An illustration of the causality

6 A unification algorithm

6.1 Principle

We present the basic idea of the unification algorithm we propose. For sake of simplicity, we assume that every transition variable appears in its input arcs.
The algorithm starts the unification process for transition t with an empty assignment $(X_1 = *, \ldots, X_n = *)$ $(X_1, \ldots, X_n$ being the variables of $t)$ which mean that no variable is assigned a value, i.e., no variable is unified. The algorithm iterates on all the tuples which label the input arcs of t. For each tuple it inspects all the tokens present in the corresponding place to find the ones which match the tuple. A token $m_c.\langle c_1, \ldots, c_n\rangle$ matches the tuple $m_t.\langle ex_1, \ldots, ex_n\rangle$ if the following conditions are met :

1. $m_c \geq m_t$
2. for each $i \in \{1..n\}$
   (a) if ex_i is a constant c then $c_i = c$
   (b) if ex_i is a variable X_j such that $X_j = c$ (and $c \neq *$) then $c_i = c$
   (c) if ex_i is a mapping $f(p_1, \ldots, p_m)$ then $c_i = f(p_1, \ldots, p_m)$

Note that if a p_i is an expression which include ununified variables then the order of analysis of the input tuples is not valid since we cannot reverse such a mapping.
When such a token is found, we decrement its multiplicity by the multiplicity

of the tuple, unify each variable X_j which appear in the tuple at position i by setting $X_j = c_i$ and pursue the unification algorithm by the analysis of the next tuple. If it is the last tuple to be analyzed, then a valid assignment has been found which can be added to the set of enabled transitions.
Concerning the guard evaluation, this one can be done as soon as all the variables of the transition which appear in it are unified. If it is evaluated to *true*, then the algorithm can pursue normally, else the unification process is stopped and the assignment is discarded.

Let us see with an example (figure 6) how we proceed. We assume the tuples are scheduled as follow : $2.\langle X, Y\rangle$, $\langle Y, f(Y)\rangle$, $\langle W, 0\rangle$, and $2.\langle W, X\rangle$.
Initially, no variable is unified : $(X = *, Y = *, W = *)$.
step 1 : analysis of $2.\langle X, Y\rangle$
We loop on each token of p. As no variable is unified, all these tokens except $\langle 7, 9\rangle$ (one more token is needed) match the tuple. This gives use the following assignments : $(X = 8, Y = 8, W = *)$, $(X = 3, Y = 4, W = *)$, $(X = 2, Y = 4, W = *)$, and $(X = 4, Y = 7, W = *)$.
step 2 : analysis of $\langle Y, f(Y)\rangle$
For the four assignments previously computed we check that a token matches the tuple $\langle Y, f(Y)\rangle$. As $2.\langle 8, 8\rangle$ has been consumed by the first tuple, this token is no more present in p and the assignment $(X = 8, Y = 8, W = *)$ is discarded. Since $f(7) = 7$ no token matches the tuple for the assignment $(X = 4, Y = 7, W = *)$. We discard this one too.
step 3 : analysis of $\langle W, 0\rangle$
$\langle 3, 0\rangle$ and $\langle 2, 0\rangle$ both match the tuple. This could potentially give the four following assignments : $(X = 2, Y = 4, W = 3)$, $(X = 3, Y = 4, W = 3)$, $(X = 2, Y = 4, W = 2)$ and $(X = 3, Y = 4, W = 2)$. However, we notice that all the variables which appear in the guard are now unified. As the guard expression does not hold for the last assignment, it is discarded.
step 4 : analysis of $2.\langle W, X\rangle$
All the variables are now unified. Since a token misses in r for the assignment $(X = 3, Y = 4, W = 3)$, this one is discarded.

Finally the set of possible assignments is :
$\{(X = 2, Y = 4, W = 3), (X = 2, Y = 4, W = 2)\}$.

A crucial point for the performance of this algorithm is the order in which tuples are treated. For instance, it is preferable to treat tuples in which constant expressions appear as early as possible since the "probability" that a token matches this tuple is low. For instance, a single token matches the tuple $\langle 5\rangle$: it is the token $\langle 5\rangle$. On the other hand, if the variable X is not unified, then all the tokens match $\langle X\rangle$. Thus, prioritizing such tuples should reduce the size of the search tree.
In the remainder, we focus on a way to find an efficient static, i.e. done before the search, scheduling of the tuples on input arcs of the transition.

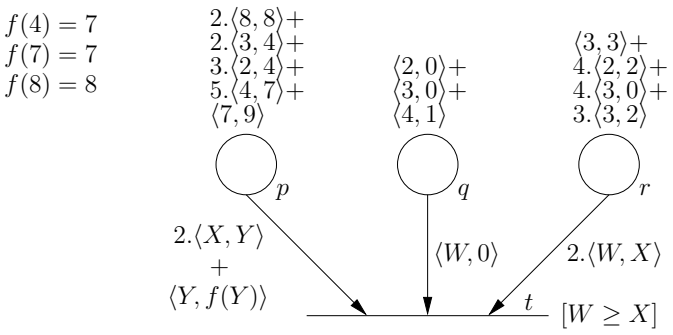


Fig. 6. Illustration of the unification process

6.2 Scheduling of the input tuples of a transition

Let t be a transition, $tup_1, \ldots, tup_n$ be the set of tuples on the input arcs of the transition scheduled in this manner and $p_1, \ldots, p_n$ be their respective corresponding places. We assume that the average number of tokens in the input places of t is k. If all the tokens match the tuples during the successive steps of the algorithm given in the previous section then all the tokens present in the input places of t are visited at each step. Thus each token t_1^i of place p_1 is visited a single time, tokens t_2^i of place p_2 are visited k times, and tokens t_j^i of place p_j are visited k^{j-1} times. This is illustrated by figure 7. We note g_1^1 the unique group of tokens of place p_1 visited at the first step, i.e., for the first tuple, of the algorithm. The visit of a group g_i^j (with $i < n$) involves the visit of groups $g_{i+1}^{j*k-k+1}, \ldots, g_{i+1}^{j*k}$.

Our aim is now to find an efficient way to schedule the input tuples of a transition. To achieve this, we use the method presented by Mäkelä in [14], that is, to define a cost function which gives an accurate idea of the complexity involved by a scheduling on the unification process. However, unlike Mäkelä's cost function, our preoccupations are double. Firstly, this scheduling must naturally minimize the size of the search tree. Secondly, checking whether or not a token matches a tuple can be expensive, as tuples may contain user defined mappings which repetitive applications may substantially slow the unification algorithm. Thus, the cost function must also try to minimize these applications. In addition, our cost function takes into account additional considerations such as transition guards, or data structures used to store local place markings.

The definition of our cost function is based on the five following principles :

1. Tuples in which constant expressions appear should be considered as early as possible since the proportion of tokens which match this tuple is low. Prioritizing such tuples should minimize the size of the generated sub trees.
2. If a variable has already been unified in a previous step, then this one should be considered as a constant since its value is set.

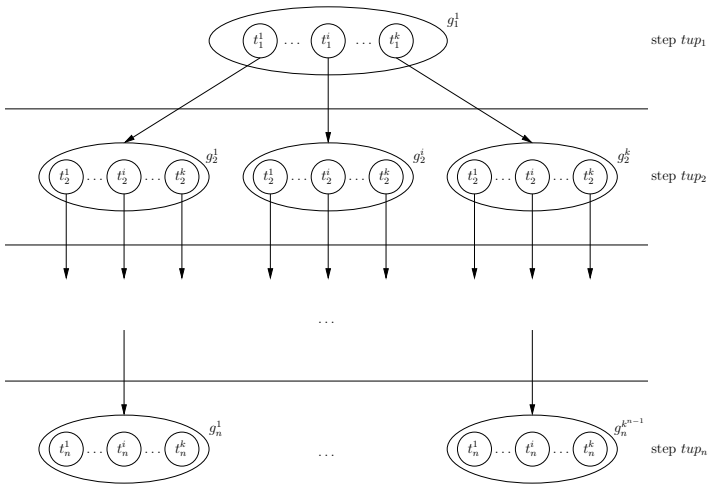


Fig. 7. Search tree of the unification algorithm

3. If all the variables appearing in a tuple have been unified, then this tuple should be considered next since the search for tokens which match this tuple can be reduced to the search of a specific element in a set. If we use balanced trees to store the local markings of places, this search has a logarithmic complexity.
4. The sooner the guard of the transition can be evaluated, the better it is. Indeed, this could allow to prune a wide set of instantiations. Let us consider for instance the transition on figure 8. Two different schedulings are possible : $\langle X\rangle, \langle Y\rangle$ and $\langle Y\rangle, \langle X\rangle$. In the worst case, i.e., all the items of C are in p and q, the first one will generate a search tree with $|C|^2$ leaves (all possible combinations will be checked), whereas the search tree of the second scheduling will have $|C \times \{c \in C | c > 10\}|$ leaves : only the couples $\langle x, y\rangle$ such that $y > 10$ will be checked. Thus the search tree will be minimal.

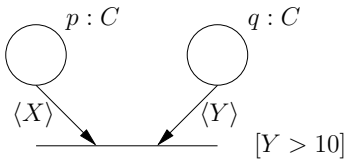


Fig. 8. Considering guards in the unification process

5. As stated before, the evaluation of a user defined function can substantially slow the unification algorithm. However, if all the variables which appear in its parameters are already unified, the function result can be computed a single time for the whole group. As an example, let us get back to the net

of figure 6. If the tuple $\langle Y, f(Y)\rangle$ is analyzed first, we have to call function f for each token present in place p since Y is not unified. If we suppose now that the tuple $2.\langle X, Y\rangle$ is analyzed before the tuple $\langle Y, f(Y)\rangle$, Y is already unified when $\langle Y, f(Y)\rangle$ is analyzed, so we only need to compute $f(Y)$ a single time for all the tokens in place p, i.e., for the whole group.

On the basis of these five principles we propose now a cost function. The following notations are used.

- $unified_i$ is the set of variables unified at level i and new_i is the set of variables unified by tup_i. It is the variables which appear in tup_i and which are not in $unified_{i-1}$. The following holds : $unified_i = unified_{i-1} \cup new_i$ and $unified_0 = \emptyset$.
- c_f is the cost of the call to function f, i.e., the complexity of f. This parameter can be provided by the user or given an arbitrary value. A cost of 0 indicates that a call to function f has a negligible impact. In this case, the cost function only gives an estimation on the size of the search tree.
- gf_i is the set of functions which appear in tup_i and such that no uninstantiated variable appears in its parameter. These functions calls can be evaluated a single time for the whole group (as stated in point 5).
- gc_i is the cost of group i, i.e., the cost to evaluate all the functions of gf_i. We define gc_i as : $gc_i = \sum_{f \in gf_i} c_f$
- tf_i is the set of function which parameters include ununified variables. These functions have to be called for each visited token (again see point 5).
- tc_i is the cost to check if a token of level i matches tup_i. We define tc_i as : $tc_i = 1 + \sum_{f \in tf_i} c_f$.
- s_i is the size of tup_i
- t_i is an estimation of the number of tokens which match tup_i. It mainly depends on the unified variables and their presence in tup_i. We use the following formula to compute t_i : $t_i = \max(1, k \times \left(\frac{|new_i|}{s_i}\right)^2)$. By this way we prioritize tuples in which constant and unified variables appear according to principles 1 and 2.
- n_i is an estimation of the number of times a token at level i is visited by the algorithm. It both depends on the number of times a token of the precedent level is visited and the number of tokens which match tup_{i-1} : we have $n_1 = 1$, and $n_i = n_{i-1} \times t_{i-1}$.
- $var(guard)$ is the set of variables which appear in the guard of t.
- cg is a cost associated to t's guard. Strong constraining guards (e.g., $X = Y$) should be assigned a higher cost than less ones (e.g., $X \neq Y$) since the set of instances disabled by the first ones is theoretically larger. A *true* guard should be assigned a value of 0. Once again, this cost parameter could be set by the model designer.

The cost function for level i is given below. It reflects point 3 : if all the variables are already unified, looking for a specific item in the local place marking has a logarithmic complexity.

$$CF_i = \begin{cases} n_i \times (gc_i + 1 + \log_2(k)) & \text{if } new_i = \emptyset \\ n_i \times (gc_i + k \times tc_i) & \text{else} \end{cases}$$

At last the total cost of a scheduling $tup_1, \ldots, tup_n$ is :

$$CF = \sum_{i=1}^{n} CF_i + cg^{i_g - 1}$$

where i_g is the smallest index in $[1..n]$ such that $var(guard) \subseteq unified_{i_g}$. In other words, after the evaluation of tup_j, the guard can directly be evaluated. This is guided by principle 4.

The idea is then to select for each transition the scheduling which minimize this cost function before the search algorithm. During the state space generation, tuples are always processed in this order. The drawback of this method is that it does not exploit some dynamic informations on the marking processed which could be useful such as the number of tokens in input places. A possible optimization would be to use some caching techniques as it is done in [8]. Indeed, we can reasonably think that the unification algorithm will perform a large number of redundant computations.

7 Experimental results

The algorithm proposed in this work has been implemented in Quasar [20], a tool for the analysis of concurrent Ada programs based on Colored Petri nets. The class of nets used in Quasar is the one described in section 2.
Quasar uses a well known technique which consists in generating a code which correspond to the actual reachability analyzer. Tools that use this technique include Prod [11], Spin [6], and Maria [15]. It has been shown in [14] that this technique greatly reduces the execution time even for small models for which we may think that the compilation of the generated code is a too severe overhead (which does not exist if the net is "interpreted").

We give here some measures performed on two examples : the slotted ring protocol taken from [16] and the distributed database management system presented by Jensen in [9]. We performed simulation experiments consisting of 10^6 event occurrences and recorded the execution times for different values of the parameters of the system. All the experiments have been made on a Pentium 4, 2.5 Ghz.

The slotted ring protocol The purpose of this example is to show how the unification process can be sped up by using balanced trees to store local place

markings. Figure 9 reports the number of firings per second observed for two different types of storage : linked lists and balanced trees. As, in this example, the unification process is quite simple and often reduced to the search of a specific token in a place, the use of balanced trees is particularly effective and we obtain very good results with this kind of storage even for small values of the number of process.

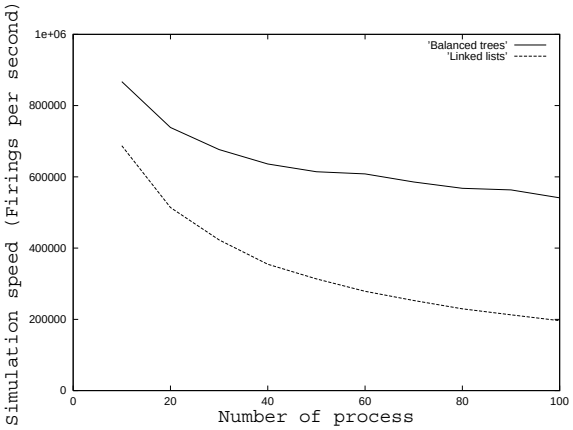


Fig. 9. Measures for the slotted ring protocol

The distributed database management system We show by this example that a static scheduling of the input tuples of a transition can reduce the enabling test complexity in a significant way. Results obtained with the optimal

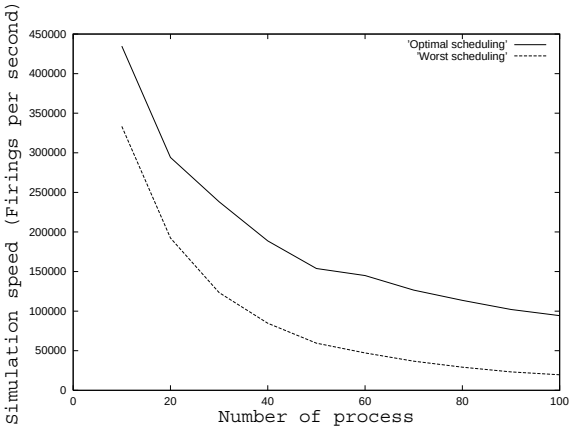


Fig. 10. Measures for the distributed database management system

scheduling, and the worst scheduling according to the cost function given in the previous section have been reported on figure 10. For 100 process, the firing rate for the optimal scheduling is 5 times greater than the one for the worst scheduling. In addition, it appears that the optimal scheduling is for this example the one which gives the best results.

8 Conclusion

We have presented in this work an algorithm for the enabling test problem of colored Petri nets. This one is based on an implementation of the conflict and causality relations to efficiently maintain a set of enabled transitions which is updated according to the transitions fired (or unfired) during the state space generation. To implement these relations we translate the color mappings of the net into equivalent constraints systems before the state space generation. These systems are repeatedly solved during the search algorithm to identify the transitions disabled or enabled by the fired transition.

As this approach is not sufficient we have also given a unification algorithm. This one processes input arcs in a predefined static order computed statically according to some heuristics which take into account numerous considerations. The major drawback of this approach is that it does not consider some dynamic informations such as the number of tokens in places. Combining our algorithm with some dynamic policies, e.g. less different tokens first policy of Gaeta [18] could be considered in future works.

References

1. Dutheillet C. and Haddad S. An efficient computation of structural relations in unary regular nets. In *Seventh International Symposium on Computer and Information Sciences (ISCIS VII)*, pages 73–79, 1992.
2. Dutheillet C. and Haddad S. Structural analysis of coloured nets. application to the detection of confusion. Technical report, Rapport IBP/MASI, 1992.
3. Dutheillet C. and Haddad S. Conflict sets in colored petri nets. In *5th International Workshop on Petri Nets and Performance Models, Toulouse (F) 19.-22. October 1993*, pages 76–85, 1993.
4. Chiola G. *A Simulation Framework for Timed and Stochastic Petri Nets.* Number 90–50. Universite Paris, Institut Blaise Pascal Rapport MASI, 1990.
5. Chiola G., Franceschinis G., and Gaeta R. A symbolic simulation mechanism for well-formed coloured petri nets. In *Proceedings of the 25th annual symposium on Simulation*, pages 192–201. IEEE Computer Society Press, 1992.
6. Holzmann G.J. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.
7. Sanders M. J. Efficient computation of enabled transition bindings in high-level petri nets. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'2000), 8-11 October 2000, Nashville, TN*, volume 4, pages 3153–3158, 2000.

8. Ilié J-M. and Rojas O. On well-formed nets and optimizations in enabling tests. In M Ajmone Marsan, editor, *14th International Conference on Application and Theory of Petri Nets*, number 691 in LNCS, pages 300–318, Chicago, USA, 1993. Springer.
9. Jensen K. Coloured petri nets and the invariant method. *Theor. Comp. Science 14*, pages 317–336, 1981.
10. Jensen K. Coloured petri nets: A high level language for system design and analysis. 483:342–416, 1991. NewsletterInfo: 39.
11. Varpaaniemi K. PROD 3.4.00 — an advanced tool for efficient reachability analysis. Laboratory for Theoretical Computer Science, Helsinki University of Technology, Espoo, Finland, June 2004. Software.
12. Mortensen K.H. Efficient data-structures and algorithms for a coloured petri nets simulator. In *3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'01) / Kurt Jensen (Ed.)*, pages 57–74. DAIMI PB-554, Aarhus University, August 2001.
13. Mäkelä M. Applying compiler techniques to reachability analysis of high-level models. In Hans-Dieter Burkhard, Ludwik Czaja, Andrzej Skowron, and Peter Starke, editors, *Workshop on Concurrency, Specification & Programming 2000*, number 140 in Informatik-Bericht, pages 129–142. Humboldt-Universität zu Berlin, Germany, October 2000.
14. Mäkelä M. Optimising enabling tests and unfoldings of algebraic system nets. In José-Manuel Colom and Maciej Koutny, editors, *Application and Theory of Petri Nets 2001 : International Conference, ICATPN 2001*, number 2075 in LNCS, pages 283–302, Newcastle upon Tyne, UK, June 2001. Springer.
15. Mäkelä M. Maria: modular reachability analyser for algebraic system nets. In Esparza J. and Lakos C., editors, *Application and Theory of Petri Nets 2002 : International Conference, ICATPN 2002*, number 2360 in LNCS, pages 434–444, Adelaide, Australia, June 2002 2002. Springer-Verlag, Berlin, Germany.
16. D. Poitrenaud and J.F. Pradat-Peyre. Pre and post-agglomerations for *LTL* model checking. In M. Nielsen and D Simpson, editors, *High-level Petri Nets, Theory and Application*, number 1825 in LNCS, pages 387–408. Springer, 2000.
17. Brgan R. and Poitrenaud D. An efficient algorithm for the computation of stubborn sets of well formed petri nets. In *Proceeding of the 16th International Conference on Application and Theory of Petri Nets, Turin, June 1995.*, pages 121–140, 1995.
18. Gaeta R. Efficient discrete-event simulation of colored petri nets. *IEEE Transaction on Software Engineering, Vol.22, No. 9*, pages 692–639, 1996.
19. Evangelista S. Syntactical rules for colored petri nets manipulation. Technical report, Rapport CNAM / Cedric, http://cedric.cnam.fr, 2004.
20. Evangelista S., Kaiser C., Pradat-Peyre J. F., and Rousseau P. Quasar: a new tool for analysing concurrent programs. In *Reliable Software Technologies - Ada-Europe 2003*, volume 2655 of *LNCS*. Springer-Verlag, 2003.
21. Haagh T.B. and Hansen T.R. *Optimising a Coloured Petri Net Simulator.* Master thesis, Univ. of Aarhus, 1994.

An Evaluation of Network Response Time using a Coloured Petri Net Model of Switched LAN

Dmitry A. Zaitsev

Odessa National Telecommunication Academy,
Kuznechnaya, 1, Odessa, 65029, Ukraine
Web: http://www.geocities.com/zsoftua

Abstract

The enterprise class model of switched LAN in the form of a coloured Petri net is represented. The components of the model are switches, servers and workstations. For the evaluation of network response time a special measuring workstation model is proposed. It counts response times for each request and calculates the average response time. For the simulation of network behaviour and accumulation of statistical information, CPN Tools was applied. Hierarchical nets usage allows the convenient representation of an arbitrary given structure of LAN.

Keywords: LAN; Switch; Response time; Colored Petri net; Evaluation

1. Introduction

The technology of switching [4] is prospective for bandwidth increase in local and global computer networks. But it is hard enough to create an adequate analytical model of a switched network [2]. Petri net models [5] contain facilities for precise description of network architecture and traffic peculiarities and allow the representation of interaction within the client-server systems.

Early represented model [9] has been refined up to enterprise quality. CSMA (Carrier Sense Multiple Access) procedures are implemented. Complete full-duplex mode is simulated with separate input and output frame buffers. The model of switch was arranged for technological convenience with fusion places allowing an easy description of an arbitrary number of ports. Moreover, the general model was supplied with special measuring workstation model that calculates network response time.

Notice that the model is represented with hierarchical coloured [3] timed [7,10] Petri nets. For automated composition of model and accumulation of statistical information during network behaviour simulation, CPN Tools [1] was used.

In the Section 2 we consider the peculiarities of switched Ethernet LAN construction. Model of LAN is described in Section 3, whereas Sections 4, 5, 6 are devoted to sub models of: switch, server, workstation, measuring workstation. Evaluation technique is represented in Section 7 and Section 8 contains the discussion of models parameters.

Results obtained may be used in real-time applications sensitive to delays, as well as at communication equipment, for instance, switches, development.

2. Switched LAN

Recently the Ethernet has become the most widespread LAN. With gigabit technology it started a new stage of popularity. And this is not the limit yet. Hubs are dumb passive equipment aimed only at the connection of devices as wires. The base element of the Local Area Network (LAN) Ethernet (IEEE 802.x) is a switch of frames. Logically a switch is constituted of a set of ports [6]. LAN segment (for example, made up via hub) or terminal equipment such as workstation or server may be attached to each port. The task of a switch is the forwarding of incoming frame to the port that the target device is connected to. The usage of a switch allows for a decrease in quantity of

collisions so each frame is transmitted only to the target port and results in an increased bandwidth. Moreover the quality of information protection rises with a reduction of ability to overhear traffic. The scheme of sample switched network is presented in Fig. 1.

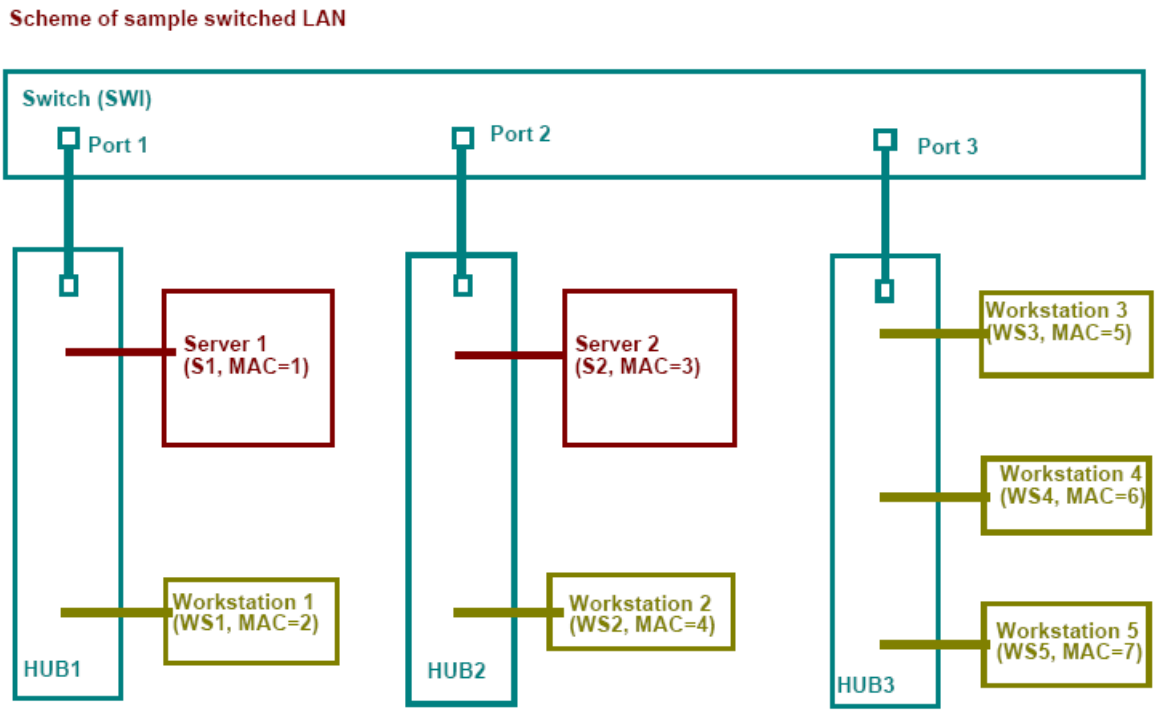


Fig. 1. Scheme of sample switched LAN

As a rule, the Ethernet works in a full-duplex mode now, which allows simultaneous transmission in both directions. To determine the target port number for the incoming frame a static or dynamic switching table is used. This table contains the port number for each known Media Access Control (MAC) address. Only static switching tables will be modelled in the present paper.

3. Model of LAN

A model of sample LAN with topology, shown in Fig. 1, is represented in Fig. 2. Let us describe the model constructed. Notice that the model is represented with coloured Petri net [3] and consists of places, drawn as circles (ellipses), transitions, drawn as bars, and arcs. Dynamic elements of the model, represented by tokens, are situated in places and move as a result of the transitions' firing.

The elements of this model are sub models of: Switch (**SWI**), Server (**S**), Workstation (**WS**) and Measuring Workstation (**MWS**). Workstations **WS1-WS4** are the same type exactly **WS**, whereas workstation **WS5** is the type **MWS**. It implements the measuring of network response time. Servers **S1** and **S2** are the same type exactly **S**. Hubs are a passive equipment and have not an independent model representation. The function of hubs is modelled by common use of the corresponding places **p*in** and **p*out** by all the attached devices. The model does not represent the collisions. Problems of the Collision Detection (CD) were studied in [11].

Each server and workstation has it's own MAC address represented in places **aS***, **aWS***. A switch has separate places for input (**p*in**) and output (**p*out**) frames for each port. It represents the full-duplex mode of work. Bidirected arcs are used to model the carrier detection procedures. One of the arcs checks the state of the channel, while another implements the transmission.

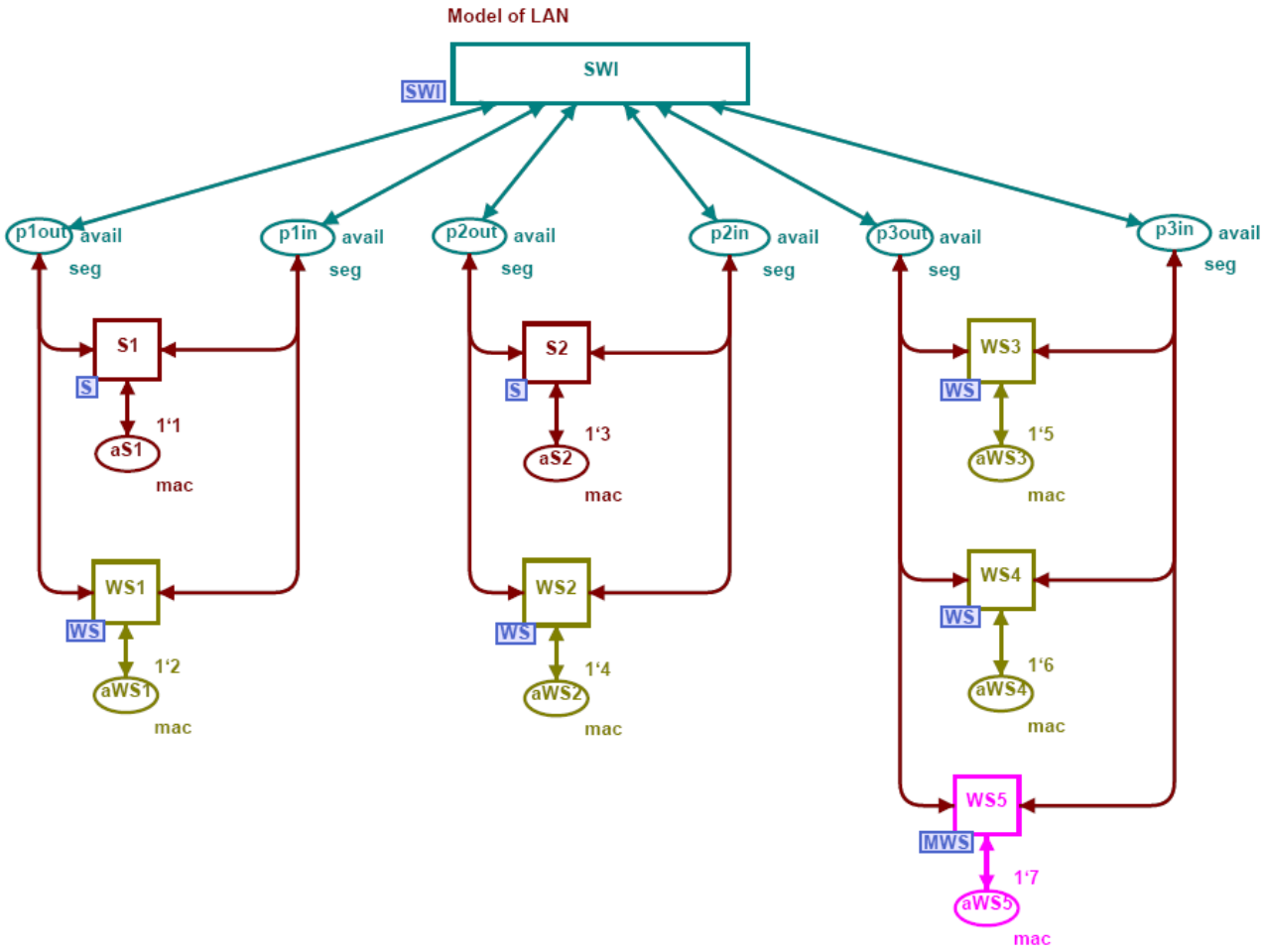


Fig. 2. Model of sample LAN

```
color mac = INT timed;
color portnum = INT;
color nfrm = INT;
color sfrm = product nfrm * INT timed;
color frm = product mac * mac * nfrm timed;
color seg = union f:frm + avail timed;
color swi = product mac * portnum;
color swf = product mac * mac * nfrm * portnum timed;
color remsv = product mac * nfrm timed;
var src, dst, target: mac;
var port: portnum;
var nf, rnf: nfrm;
var t1, t2, s, q, r: INT;
color Delta = int with 1000..2000;
fun Delay() = Delta.ran();
color dex = int with 100..200;
fun Dexec() = dex.ran();
color dse = int with 10..20;
fun Dsend() = dse.ran();
color nse = int with 10..20;
fun Nsend() = nse.ran();
fun cT()=IntInf.toInt(!CPN'Time.model_time)
```

Fig. 3. Declarations

All the declarations of colours (**color**), variables (**var**) and functions (**fun**) used in the model are represented in Fig. 3. The Ethernet MAC address is modelled with integer number (colour **mac**). The frame is represented by a triple **frm**, which contains source (**src**) and destination (**dst**)

addresses, and also a special field **nfrm** to enumerate the frames for the calculation of response time. We abstract of other fields of frame stipulated by standard of Ethernet. The colour **seg** represents unidirectional channel and may be either available for transmission (**avail**), or busy with transmission of a frame (**f.frm**). It is represented with a **union** type of colour. Notice that the descriptor **timed** is used for tokens, which take part in timed operations such as delays or timestamps.

The marking of places is represented with multisets in CPN Tools. Each element belongs to a multiset with defined multiplicity, in other words – in a few copies. For instance, the initial marking of the place **aWS2** is **1`4**. It means that place **aWS2** contains 1 token with a value of 4. The union of tokens is represented by a double plus sign (++). Tokens of timed colour have the form **x @ t** which means that token **x** may be involved only after a moment of time **t**. So, notation **@+d** is used to represent the delay with the interval **d**.

4. Model of Switch

Let us construct a model for a given static switching table. We consider the separate input and output buffers of frames for each port and common buffer of the switched frames. The model of switch (**SWI**) is presented in Fig. 4. The hosts' disposition according to Fig. 1 was used for the initial marking of a switching table.

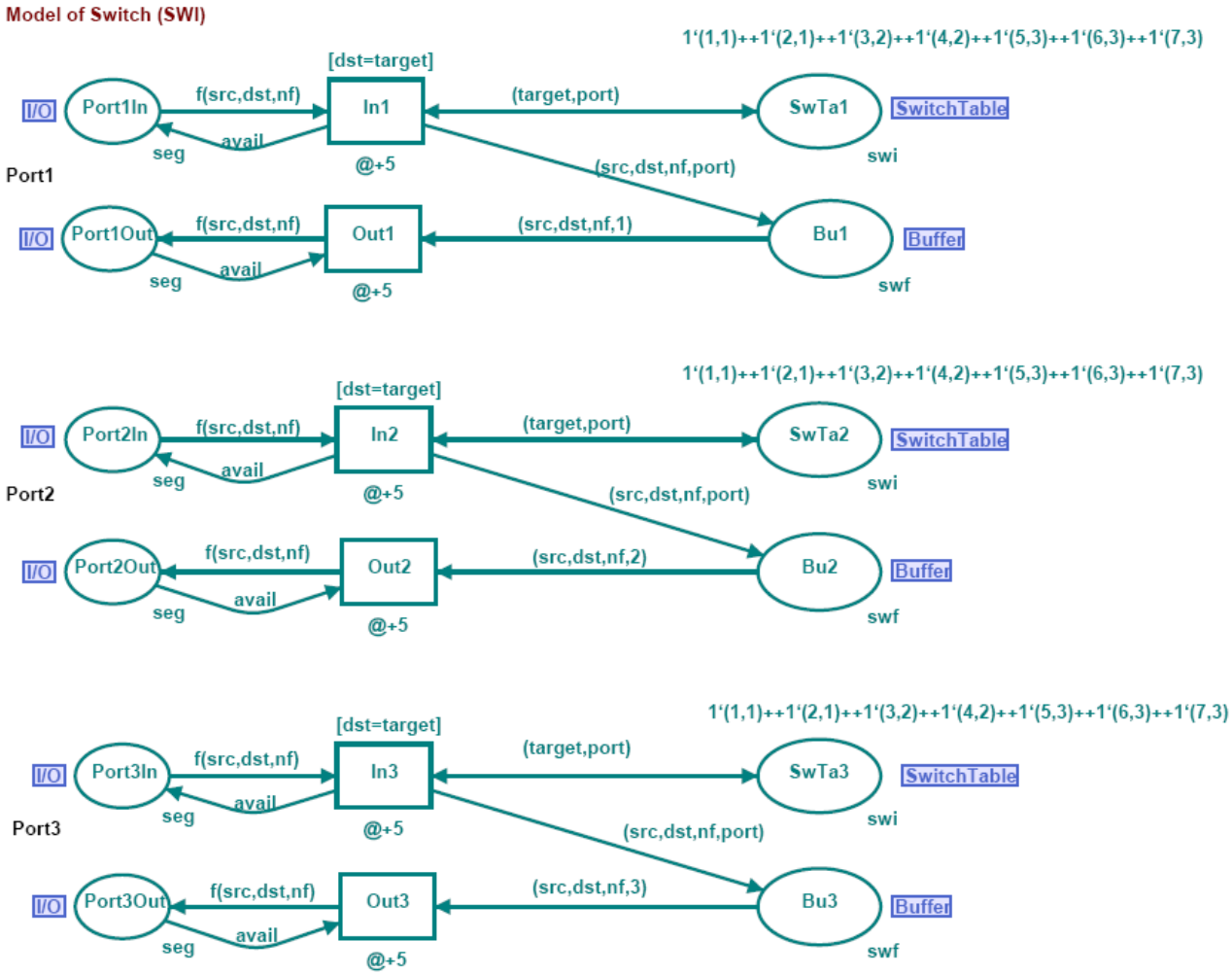


Fig. 4. Model of switch

The colour **swi** represents records of switching table. It maps each known MAC address (**mac**) to the number of port (**nport**). The colour **swf** describes the switched frames, waiting for output buffer

allocation. The field **portnum** stores the number of the target port. The places **Port*In** and **Port*Out** represent input and output buffers of the ports correspondingly. The fusion place **SwitchTable** models the switching table; each token in this place represents the record of the switching table. For instance, token **1`(4,2)** of the initial marking means that the host with MAC address 4 is attached to port 2. The fusion place **Buffer** corresponds to the switched frames' buffer. Notice that a fusion place (such as **SwitchTable** or **Buffer)** represents a set of places. The fusion place **SwitchTable** is represented with places **SwTa1, SwTa2, SwTa3**. The fusion place **Buffer** is represented with places **Bu1, Bu2, Bu3**. It allows the convenient modelling of switches with an arbitrary number of ports avoiding numerous cross lines.

The transitions **In*** model the processing of input frames. The frame is extracted from the input buffer only in cases where the switching table contains a record with an address that equals to the destination address of the frame (**dst=target**); during the frame displacement the target port number (**port**) is stored in the buffer. The transitions **Out*** model the displacement of switched frames to the output ports' buffers. The inscriptions of input arcs check the number of the port. The fixed time delays (@+5) are assigned to the operations of the switching and the writing of the frame to the output buffer.

It is necessary to explain the CSMA procedures of LAN access in more detail. When a frame is extracted from the input buffer by transition **In*,** it is replaced with the label **avail**. The label **avail** indicates that the channel is free and available for transmission. Before the transition **Out*** sends a frame into a port, it analyses if the channel is available by checking the token **avail**.

Notice that places **Port*In** and **Port*Out** are contact ones. They are pointed out with an **I/O** label. Contact places are used for the construction of hierarchical nets with substitution of transition. For example, the transition **SWI** in the top-level page of model (Fig. 2) is substituted by a whole net **SWI** represented in Fig. 3. Places **Port*In** and **Port*Out** are mapped into places **p*in** and **p*out** correspondingly.

5. Models of Workstation and Server

To investigate the frames' flow transmitting through LAN and to estimate the network response time it is necessary to construct the models of terminal devices attached to the network. Regarding the peculiarity of the traffic's form we shall separate workstations and servers. For an accepted degree of elaboration we consider periodically repeated requests of workstations to servers with random uniformly distributed delays. On reply to an accepted request a server sends a few packets to the address of the requested workstation. The number of packets sent and the time delays are uniformly distributed random values.

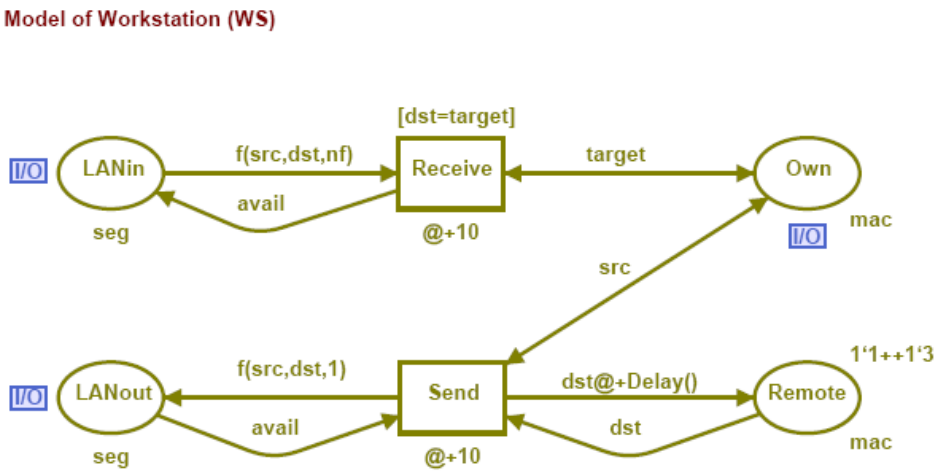


Fig. 5. Model of workstation

A model of workstation (**WS**) is represented in Fig. 5. The places **LANin** and **LANout** model the input and output channels of the local area network correspondingly. The workstation listens to the network by means of transition **Receive** that receives frames with the destination address, which is equal to the own address of the workstation (**dst=target**) saved in the place **Own**. The processing of received frames is represented by the simple absorption of them. The workstation sends periodic requests to servers by means of transition **Send**. The servers' addresses are held in the place **Remote**. After the sending of a request the usage of the server's address is locked by the random time delay given by the function **Delay()**. The sending of the frame is implemented only if the LAN segment is free. It operates by checking place **LANout** for a token **avail**. In such a manner the workstation interacts with a few servers holding their addresses in the place **Remote**.

Notice that the third field of frame, named **nfrm,** is not used by the ordinary workstation **WS**. The workstation only assigns the value of a unit to it. This field is used by a special measuring workstation **MWS**. The copies of the described model **WS** represent workstations **WS1-WS4**. To identify each workstation uniquely, the contact place **Own** is used. This place is shown also in the top-level page (Fig. 2) and contains the MAC address of host.

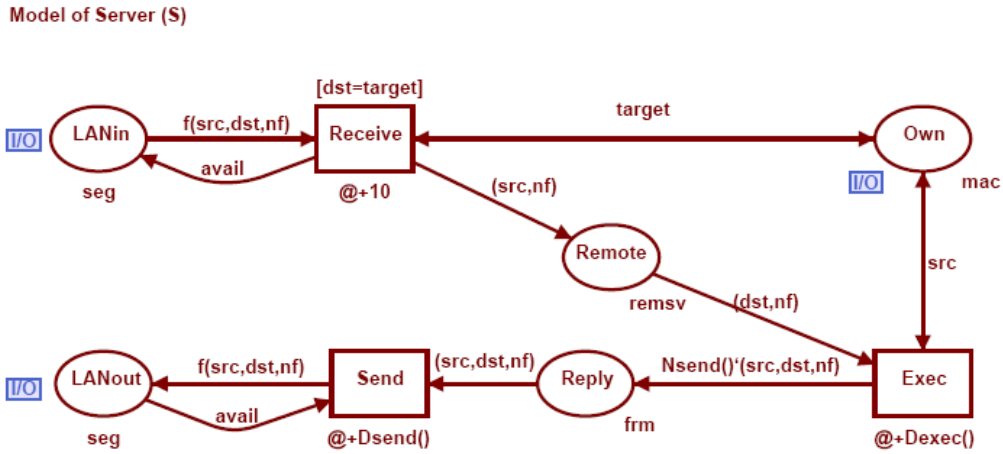


Fig. 6. Model of server

A model of server (**S**) is represented in Fig. 6. The listening of the network is similar to the model of the workstation but it is distinct in that the frame's source address is held in the place **Remote**. The transition **Exec** models the execution of the workstation's request by a server. As a result of the request execution the server generates a random number **Nsend()** of the response frames, which are held in the place **Reply**. Then these frames are transmitted into the network by the transition **Send**. Notice that the request number **nf** is stored in the place **Remote** also. It allows us to identify the response with the same number as the request.

6. Model of Measuring Workstation

A model of the measuring workstation (**MWS**) is represented in Fig. 7. In essence, it is an early considered model of workstation **WS**, supplied with the measuring elements (the measuring elements are drawn in magenta).

Let us consider the measuring elements in more detail. Each frame of a workstation's request is enumerated with a unique number contained in the place **num**. The time, when the request was sent, is stored in the place **nSnd**. The function **cT()** calculates the current value of the model's time. The place **nSnd** stores a pair: the frame's number **nf** and the time of request **cT()**.

The place **return** stores the timestamps of all the returned frames. As the network response time we consider the interval of time between the sending of the request and receiving the first frame of response. This value is stored in place **NRTs** for each responded request. The transition **IsFirst**

determines the first frame of response. The inscription of the arc, connecting the transition **IsFirst** with the place **NRTs**, calculates the response time (t2-t1).

A residuary part of the measuring elements calculates the average response time. The places **sum** and **quant** accumulate the sum of response times and the quantity of accepted responses correspondingly. The arrival of a new response is sensed by the place **new** and initiates the recalculation of average response time with the transition **Culc**. The result is stored in the place **NRTime**.

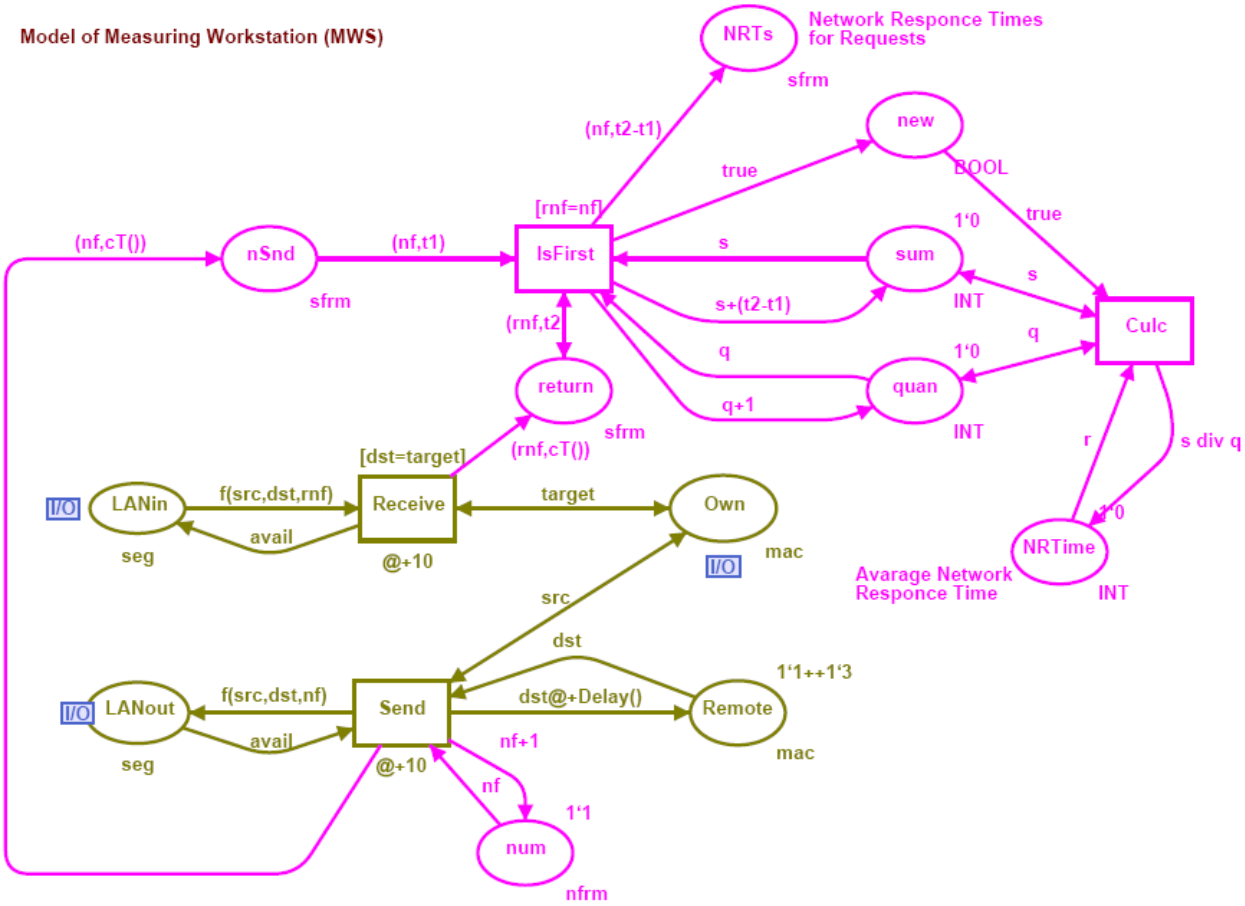


Fig. 7. Model of measuring workstation

7. Evaluation Technique

The model constructed was debugged and tested in a step-by-step mode of simulation. For these purposes the frame generated by the workstation was traced through the network to the server and back. Also we observed the behaviour of the model in the process of automated simulation with a display of net's dynamics – in the mode of the so-called game of tokens. It allows us to estimate the model with a glance at the top-level page and at sub pages during simulation.

To estimate the network response time precisely, rather huge intervals of model time are required. It is convenient for such purposes to use the simulation mode without displaying intermediate marking aimed at the accumulation of statistics.

A snapshot of the measuring workstation model is represented in Fig. 8. The rectangular labels (drawn in bright green) describe the current marking of the simulation system; the circular labels contain the number of tokens. The place **LANin** contains frame (1,5,1). The place **LANout** represents the available state of the channel **avail**. The number of the next request, according to the marking of place **num**, is 7. The place **return** indicates that 83 frames of responses have arrived. The place **NRTs** contains the response times for each of the 6 responded requests. For instance, the network response time for request 5 equals to 235. It should be calculated easily, that the average

network response time 389 in the place **NRTime** equals to 2337/6 according to the markings of the places **sum** and **quant**.

8. Parameters of Model

The right choice of time unit for model time measurement is a key question for an adequate model construction as well as the calculation of timed delays for elements of the model. It requires an accurate consideration of the real network hardware and software characteristics.

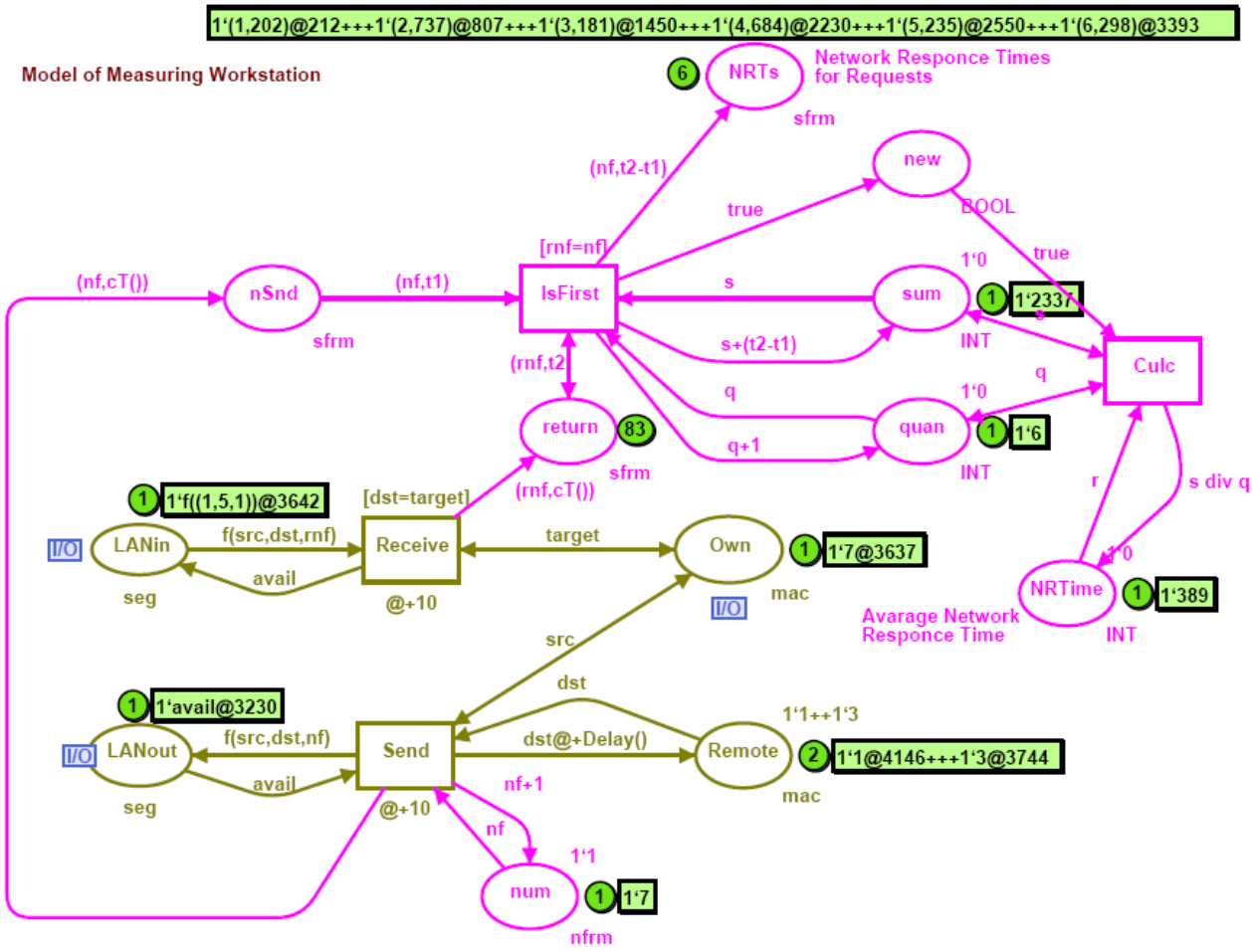


Fig. 8. Estimation of network response time

The scheme shown in Fig. 1 represents a fragment of a railway dispatch centre LAN supplied with special railway CAM software GID Ural [12]. The core of the system constitutes a pair of mirror servers **S1** and **S2**. The workstations **WS1-WS5** are situated in the workplaces of railway dispatchers.

We have to consider the performance of the concrete LAN switch and LAN adapters to calculate the timed delays of transitions **In***, **Out***, **Send**, **Receive**. Moreover, the peculiarities of client-server interaction of GID Ural software ought to be considered for the estimation of such parameters as delay between the requests **Delta** and the time of request execution **dex**. Since the unit of information transmitting through net is represented with a frame, we have to express the lengths of messages in numbers of frames. For these purposes the maximal length of an Ethernet frame equalling 1.5 Kb was chosen.

The types of LAN hardware used are represented in Table 1.

Table 1. Types of hardware

Device	Type
LAN adapter	Intel EtherExpress 10/100
LAN switch	Intel SS101TX8EU
Server	HP Brio BA600
Workstation	HP Brio BA200

In Table 2 the parameters of the model described are represented. LAN switch and adapter operations are modelled with fixed delays so they are small enough in the comparison with client-server interaction times. Moreover, in reliable Ethernet frames of maximal length are transmitted mainly, since the time of frame's processing is a fixed value. Stochastic variables are represented with uniform distribution, which corresponds to Ural GID software behaviour. The smallest timed value is the LAN switch time of read/write frame operation. But for the purposes of future representation of faster equipment we choose the unit of model time (MTU) equalling 100 ns.

Table 2. Parameters of model

Parameter	Variable/Element	Real value	Model value
LAN switch read frame delay	In*	500 ns	5
LAN switch write frame delay	Out*	500 ns	5
LAN adapter read frame delay	Receive	1 ms	10
LAN adapter write frame delay	Send	1 ms	10
Server's time of request processing	Dex	10-20 ms	100-200
Client's delay between requests	Delta	100-200 ms	1000-2000
Length of request		1.2 Kb	1
Length of response	Nse	15-30 Kb	10-20

Thus, the average network response time obtained equals 389 MTU or about 39 ms. This delay satisfies the requirements of train traffic control [12].

9. Conclusion

In the present work the technology of switched local area networks' models development was studied. The usage of coloured Petri nets allows the peculiarity of interaction within the client-server systems to be taken into account. The model reflects the major features of a real-life network. CSMA procedures, full-duplex mode and switching tables were modelled. A special measuring model of workstation was suggested and implemented to estimate the network response time.

The model developed is of enterprise class, so it allows easy and convenient adequate representation of LAN with an arbitrary given topology. The technique described is aimed at real-time applications, requiring the precise estimation of timed delays before implementation.

References

1. Beaudouin-Lafon M., Mackay W.E., Jensen M. et al. CPN Tools: A Tool for Editing and Simulating Coloured Petri Nets. LNCS 2031: Tools and Algorithms for the Construction and Analysis of Systems, 2001, 574-580.
2. Elsaadany M., Singhal T., Lui Ming. Performance study of buffering within switches in local area networks. Proc. of 4th International Conference on Computer Communications and Networks, 1995, 451-452.
3. Jensen K. Colored Petri Nets – Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag, Vol. 1-3, 1997.

4. Hunt R. Evolving Technologies for New Internet Applications. IEEE Internet Computing, 5 1999, 16-26.
5. Peterson J. Petri Net Theory and the Modelling of Systems. Prentice Hall, 1981.
6. Rahul V. LAN Switching. OHIO 2002.
7. Zaitsev D.A., Sleptsov A.I., State Equations and Equivalent Transformations of Timed Petri Nets. Cybernetics and System Analysis, 33, 1997, 659-672.
8. Zaitsev D.A. Subnets with Input and Output Places. Petri Net Newsletter, Vol. 64, April 2003, 3-6, Cover Picture Story.
9. Zaitsev D.A. Switched LAN simulation by colored Petri nets. Mathematics and Computers in Simulation, vol. 65, no. 3, 2004, 245-249.
10. Zaitsev D.A. Invariants of timed Petri nets. Cybernetics and Systems Analysis, no. 2, 2004, 92-106.
11. Zaitsev D.A. Verification of Ethernet protocols // Proceedings of Odessa National Telecommunication Academy, no. 1, 2004, p. 42-48.
12. Zyabirov H.S., Kuznetsov G.A., Shevelev F.A., Slobodenyuk N.F., Krasheninnikov S.V., Krayisvitny V.P., Vedischev A.N. Automated system for operative control of exploitation work GID Ural-VNIIZT // Railway transport, no. 2, 2003, 36-45.

A Formal Model for Information Risk Analysis Using Colored Petri Nets

Peter. R. Stephenson, CISSP, CISM, FICAF
The Center for Regional and National Security, Eastern Michigan University

---

Abstract. *The analysis of information systems (IS) risk, especially quantitative analysis, is fraught with inaccuracy and unreliability for a variety of reasons. First, the metrics required as input data to virtually all types of quantitative IS risk analysis methods are based upon data that are virtually impossible to collect accurately. These data require a thorough understanding of detailed threats and impacts on an asset-by-asset basis. In most organizations those data usually are not recorded accurately, consistently or over a long enough period of time to be reliable for quantifying, ranking and predicting risk.*

The second weakness of current IS risk analysis approaches is that they tend to be asset-based. In an environment where IS assets can run in the tens of thousands it is impractical to have a complete inventory of such assets, let alone a clear understanding of the individual asset loss impacts under varying types of threats.

Finally, vulnerability assessments and risk analyses usually are performed infrequently in large enterprises due to cost and inconvenience. This means that such assessments and analyses are, at best, snapshots that may not be repeated for months or, even, years. Certainly, any annualized loss expectancy figures, unreliable under the best of circumstances, soon become, essentially, useless. That makes effective ongoing IS risk management almost impossible in a practical sense.

In this paper we describe the FARES (Forensic Analysis of Risks in Enterprise Systems) project and propose an alternative method of IS risk analysis that supports ongoing IS risk management based upon information systems security (ISS) modeling, comparison against global (i.e., world-wide) norms (a quasi-actuarial approach) and quantification using statistical methods.

An important aspect of ongoing management of IS risk, maintaining a current risk snapshot without the burdensome cost of continual physical analysis of large information systems, is satisfied by the use of Colored Petri Nets (CPNets). CPNets allow sophisticated modeling, simulation and analysis of complex information security system behavior using proven formalisms.

Additionally, CPNets have the advantage of being graphical representation which allows risk management experts to construct, modify and present complex models without the need for advanced training in formal methods. Finally, such graphical representations are ideally suited for presentation to lay audiences such as executive and financial managers.

The FARES project is ongoing and is in early stages of evolution and field testing. The models are trivial and approaches we discuss here largely are intended to be proof-of-concept and are evolving continuously.

1.0 Background and Problem Statement

The notion of risk management and risk analysis in the information technology arena is based upon a twenty year-old approach. Current teaching is clear that information technology (IT) risk must be asset-based, in clear opposition to general risk analysis wisdom that focuses upon modeling approaches based upon actuarial historical data.

1.1 Current State of IT Risk Analysis

Typical risk analysis outside of the IT world may be generalized into three types: Point Estimates, Range Estimates and What-if Scenarios [DEC02]. The problem with these three methods, and with risk analysis in general, is that they require reliable input data to be successful. Often those data are not available.

When accurate input data are not available, the next best approach is the use of modeling methods. These methods generally make some assumptions based upon whatever data is available and attempt to infer future results. While this is more accurate than assuming data and building directly from that data to achieve a predictive result, we still are restricted to working with estimates.

The potential for accuracy, then, lies in predicting the future from as clear a picture of the past as possible. Insurance companies attempting to manage the risk of making more in benefits payouts than they take in as premium payments use an actuarial approach collecting historical data. Simply put, that means that they estimate the future from a global view of the past.

The two major areas where IS risk analysis and management techniques diverge from those of other industries lie in the choice of historical data and the use (or lack of use) of modeling techniques. This leads to some important uncertainties.

The three major approaches to risk analysis outside of IS have significant drawbacks, especially in an IS environment:

- **Range Estimates:** Calculate vague classifications for risk (best case, worst case, probable case, or high risk, low risk, medium risk). Outputs are very subjective and difficult to use as risk management metrics.

- **What-if Scenarios:** Usually use assumptions from Range Estimates to predict as many outcomes as possible. Since this approach is based upon subjective interpretation it is likely to be flawed as a predictive method.

- **Point Estimates:** Implies accurate results but, in fact, can be very misleading due to the uncertainty of input variables. Most IT quantitative risk analysis methods are, in fact, Point Estimates based upon presumed known asset values and presumed known threats, vulnerabilities and impacts. Since these data rarely are available within the organization undergoing the analysis, it is unlikely that the Point Estimates will reflect the reality of the organization's IT risk posture.

Generally IS risk analysis takes one of two possible directions: quantitative and qualitative. Qualitative risk analysis is easier to perform, admits of a lack of reliable input data but returns no useful metrics. Quantitative risk analysis offers metrics but, due to the uncertainty of the input data, those metrics are not likely to be particularly useful.

When quantitative methods actually do admit of unreliable historical data, they tend to evolve into Range Estimates, again producing little of value in the way of metrics that will aid in the management of future risks.

Other important work in the field relates more to the formal description of computer security incidents using state machine analysis [GP04]. There are significant correlations between such incidents and information system risk. Some of this approach to incident analysis may well have application in describing information system risk.

1.2 A Theoretical Solution to the Problem

The key to performing a credible IT risk analysis lies in addressing major challenges:

- Selection of credible historical data
- Application of established statistical methods for predicting future risks
- Use of modeling techniques to represent the volatility of today's IT enterprises and allow for continual reassessment of risk posture based upon both local and global changes.

We have approached these challenges as follows:

- **Historical data.** Collection of global data on IT threats vulnerabilities and impacts. Cataloging of threats, vulnerabilities and impacts into manageable taxonomies.
- **Predicting and forecasting.** Use of accepted statistical techniques such as Monte Carlo simulation, Bayesian probability, etc.
- **Ongoing Modeling.** Application of Colored Petri Nets to capture the current operating state of the enterprise and to model and simulate changes in that state over real time.

The result of this research is the FARES program. The program is extremely young and addresses an audience traditionally resistant to formal modeling. For that reason the program currently is in the very early stages of acceptance in the information assurance community.

This paper focuses upon the application of Colored Petri Nets (CPNets) to the ongoing modeling challenges but, necessarily, touches upon the remaining two issues. A significant benefit of CPNets and, particularly, CPNTools, is that there are ways to embed the formal modeling aspect on the FARES process beneath of veneer of operator-friendly interfaces.

1.3 Clarifying the Risk Management and Risk Analysis Processes

There is some danger in considering risk management solely as a business process. It is, more correctly, an iterative risk analysis process. When considering business processes, for example, we have the notions of phases, activities, tasks, and techniques, as well as issues related to human resources, technology, and the life-cycle model to be used [FD04]. In risk management we don't address most of those directly. Rather, we incorporate the risk management process into other business processes.

Jones [JKL02] defines risk management as the:

> *"Process of identifying and applying countermeasures commensurate with the value of the assets protected based upon a risk assessment."*

The definition is, of course, in the context of information security and describes an asset-

based process. While the approach reported here is intended to be a generation beyond asset-based analysis, the underlying definition of risk management holds because, whether or not the analysis is asset-based, the goal remains a balanced approach to managing risk to information assets.

Further, most business processes may be thought of in terms of work-flow management. Workflow management systems support of the definition, execution, registration and control of processes [WA98]. For the purposes of risk analysis, workflow management poses virtually no applicability.

However, the key issue, from the perspective of the FARES process, is that FARES is an analysis, not a business process. FARES supports and is a tool of risk management.

2.0 Approach

The FARES approach consists of the following specific elements:

1 Selection of global sources of actuarial historical input data to be used as a comparison baseline
2 Development of a suite of data management tools to manage historical reference data and collect local data for the enterprise under evaluation
3 Application of appropriate local data collection techniques, such as vulnerability assessment, data flow analysis, data and enterprise classification information, access control information, etc.
4 Application of appropriate statistical analysis tools to characterize local data in comparison with historical actuarial data.
5 Identification of security policy domains and their inter-domain communications channels
6 Modeling inter-domain data flow, both permitted and potentially unauthorized
7 Modeling threat, vulnerability and impact status first as a baseline, and ultimately at any time critical elements of the model change
8 Applying a combination of statistical and process analysis (from CPNet models) to determination of current and future risk profiles, risk probabilities, and current local profiles in comparison to global historical profiles

2.1 Characterizing the Enterprise Under Analysis

We characterize the enterprise under analysis in terms of security policy domains. In the definitions below we use the following simple notations:

$A \subseteq B$	A is a subset of B
$A \mid B$	A such that B
$A \bullet B$	A is subject to B
$A \Rightarrow B$	A implies B

We define security policy domains:

Definition 1 – Security Policy Domain[1].

A security policy domain consists of all of the elements of an enterprise that are subject to the same security policy.

$$P_{dom1} \subseteq E \mid \varepsilon_1 \bullet p_1$$

such that:

(i) P is the set of all security policies on a bounded enterprise network

(ii) E is the set of all elements on a bounded enterprise network

(iii) p_n is a particular policy where $p_n \in P$

(iv) ε_n is a subset of elements where $\varepsilon_n \subseteq E$

(v) P_{dom} is a security policy domain

The notion of security policy domains is critical to the FARES process for several reasons. First, in most large enterprises there are far too many elements in the IT environment to be handled individually. In a large organization the number of such elements easily ranges in the tens of thousands, and, not uncommonly, in the hundreds of thousands.

The second requirement for the use of security policy domains is that they allow the analysis of data flows on the enterprise at a manageable level. Determining data flows between every IT element in an enterprise with several thousand servers, as well as tens or hundreds of thousands of users is a near impossible task. However even a large globally-distributed enterprise may break down into fewer than 50 policy domains.

Finally, security policy domains offer an ideal platform upon which to measure threats, vulnerabilities and impacts. Once those elements of risk are measured, appropriate safeguards may be applied to all of the elements that comprise the policy domain. The high level view of these elements (from the domain perspective) allows managers a clearer picture of risks and appropriate countermeasures than does a granular, element-level view.

2.2 Defining IT Security Risk and its Elements

The elements of IT risk are threats, vulnerabilities and impacts. We take each definition in turn.

We begin with the definition of a security incident because this definition is used in later definitions and, although we do not address the issues of security incidents directly here, it is important for the role it plays in our current topic.

We follow the definition of a computer security incident (Definition 2) with the definition of a vulnerability (Definition 3). These terms are fundamental to the IT security environment and do not, to date, appear to have been defined formally in the IT security literature.

[1] Stephenson, Peter [PRS04]

Definition 2 – Computer Security Incident[1].

A computer security incident is a change of state in a bounded computer system from the desired state to an undesired state, where the state change is caused by the application of a stimulus external to the system.

$$I = \alpha_{post} \mid \exists \alpha_{pre} \cdot \beta \Rightarrow \alpha_{post} \neq \alpha_{cor}$$

such that:

(i) A is the set of all possible operating states of a bounded computer system S

(ii) α_{cor} is the desired operating state of S where $\alpha_{cor} \in$ A

(iii) α_{pre} is the pre-incident operating state of S where $\alpha_{pre} \in$ A

(iv) $\alpha_{pre} = \alpha_{cor}$

(v) α_{post} is the post-incident operating state of S where $\alpha_{post} \in$ A

(vi) B is the set of external stimuli applied to S

(vii) β is an external stimulus where $\beta \in$ B

(viii) I is a computer security incident

Definition 3 – Vulnerability

A vulnerability is a weakness or flaw, in an element of a system, that has the potential to be exploited with a damaging outcome[2].

$$v_n = \varepsilon_n \cdot \alpha_{fn} \mid \exists \varepsilon_n \cdot \beta \Rightarrow I$$

such that:

(i) E is the set of all elements in bounded system S

(ii) ε_n is a specific element where $\varepsilon_n \in E$

(iii) A is the set of all possible operating states of bounded system S

(iv) α_{fn} is a flawed or weakened operating state where $\alpha_{fn} \in A$

(v) V is the set of all possible vulnerabilities in a bounded system

(vi) v_n is a specific vulnerability where $v_n \in V$

(vii) B is the set of all possible external stimuli that could be applied to ε_n

(viii) β is an external stimulus applied to ε_n where $\beta \in$ B

(ix) I is a computer security incident (Definition 2)

The difficulty in defining vulnerabilities, threats and impacts in the context of today's view of information systems security is that they are concepts whose meaning has grown up over time without an effort to provide formal definitions. Jones [AJ02] gives informal definitions and, from those, we have derived the more formal definitions needed to apply the concepts to CPNets.

We follow Definition 3 with definitions of the other risk elements and, finally, with a formal definition of IT risk itself.

The definition of threat follows. This definition approaches threat from a very simple perspective: a threat is nothing more that an external stimulus that causes some element in an IT system to change to an undesirable state. There is a tendency among today's information security practitioners to confuse threat with vulnerability. In our definition we view a threat generally (although Jones makes a distinction between a natural threat, a threat based upon an unintentional act and a malicious threat) because at this level of abstraction we view a threat simply as

[2] Jones, Andrew [AJ02]

something that is capable of causing an element of an enterprise to change state from a correct or desired operating state to an undesired state. The definition does not address why this state change could occur (the underlying vulnerability of the element) or what the motivation for the action is. Finally, it does not address the capability of a threat agent to deliver the threat. We save these issues for a more detailed discussion involving all of the applicable elements of risk.

Definition 4 – Threat

A threat is a natural disaster, an unintentional act by an individual that causes harm or a malicious act by an individual or group of individuals[2].

$$\tau = \varepsilon_n \bullet \beta \Rightarrow \alpha_u$$

Such that:

(i) T is the set of all threats including natural disasters and malicious or unintentional acts by individuals that causes harm

(ii) τ is a specific threat where $\tau \in$ T

(iii) E is the set of all elements in bounded system S

(iv) ε_n is a specific element where $\varepsilon_n \in E$

(v) A is the set of all possible operating states of bounded system S

(vi) α_u is an undesired operating state in bounded system S state where $\alpha_u \in A$

(vii) B is the set of all possible external stimuli that could be applied to ε_n

(viii) β is an external stimulus applied to ε_n where $\beta \in$ B

The next definition is the definition of impact.

Definition 5 – Impact

An impact is an unwanted or adverse effect on a system or organization that an incident would cause[2].

$$\mu = \alpha \left| \exists \alpha \bullet \mathrm{I} \Rightarrow \alpha_u \right.$$

such that:

(i) I is a computer security incident (Definition 2)

(ii) A is the set of all possible operating states of bounded system S

(iii) α is an operating state in bounded system S operating state where $\alpha \in A$

(iv) α_u is an undesired operating state in bounded system S state where $\alpha_u \in A$

(v) M is the set of all possible impacts

(vi) μ is a specific impact where $\mu \in$ M

Finally, we assemble these definitions into a formal definition of IT risk. It is important, at this point, to recognize that there may be other definitions of IT risk and its elements. In the FARES project we have attempted to arrive at the simplest definitions of these basic building blocks so that we do not, by the fact of definition, unnecessarily limit the scope of future models.

The FARES project takes a simple approach without being simplistic. The assessment, analysis and management of IT risk is, of itself, a very complex undertaking. It is inappropriate to add artificial complexity through the use of unnecessarily complicated foundational definitions.

Definition 6 - Information Systems Risk

Information Systems Risk is the probability that a threat agent will successfully exploit a vulnerability to create an unwanted or adverse impact[3].

$$\rho = P\left(\nu \cdot \tau \Rightarrow \mu \middle| \exists \alpha_u\right)$$

such that:

(i) τ is a specific threat (Definition 4)

(ii) ν is a specific vulnerability (Definition 3)

(iii) μ is a specific impact (Definition 5)

(iv) A is the set of all possible operating states of bounded system S

(v) α_u is an undesired operating state in bounded system S where $\alpha_u \in A$

(vi) P is the set of all information systems risks

(vii) ρ is a specific information systems risk where $\rho \in$ P

Here, we have used an italic P to denote probability and an upper case Greek letter Rho (P) as the set of all information system risks.

2.3 Historical Data

A key element of the FARES approach is the selection of historical baseline data against which to profile the enterprise under analysis. We selected global historical data to represent threats, vulnerabilities and impacts. The data for building an actuarial profile of vulnerabilities was taken from the records of approximately 1 billion individual cyber attacks and probes world-wide over a period of 19 months ending in July of 2004[4] collected by the DShield network of over 500,000 attack sensors. The attacks were cross-referenced to target ports and thence to specific vulnerabilities and exposures using the Mitre Common Vulnerabilities and Exposures (CVE) catalog[5].

Data for threats and impacts is the result of interviews and surveys of organizations of all types and sizes and is ongoing.

2.3.1 Organization of Historical Reference Data

There are thousands of vulnerabilities and over one hundred specific threats cataloged by various sources. These specific threats and vulnerabilities change rapidly and are, essentially, impossible to track reliably. Additionally, there is the problem of "zero-day attacks". These attacks are based upon exploits that address newly discovered but unannounced vulnerabilities. These factors make the assessment of future risk based upon past history challenging.

In order to achieve the dual objectives of maintaining a manageable frame of reference and maintaining a relatively static basis for profiling, we reduced the CVE dictionary to a taxonomy of vulnerabilities and exposures. Likewise, using the Common Criteria (ISO 15408) we extracted a threat taxonomy and, finally, we developed an impact taxonomy based upon results of interviews and survey questionnaires. The results comprise a set of 56 categories of vulnerability, 13 categories of impact and 30 categories of threat.

2.4 Analysis Tool Set

We have built a core tool called the FARES Data Store (FDS) using Microsoft Access. The FDS is a repository for the individual

[3] Jones, Andrew. "Identification of a Method for the Calculation of Threat in an Information Environment"[AJ02] with additions and modifications by Stephenson, Peter

[4] Euclidian consulting, Quincy, MA, USA.

[5] http://www.cve.mitre.org/

taxonomies, summary statistics on the historical data and a collection, analysis and reporting point for the FARES process.
The analyst uses a suite of data collection and assessment tools to collect threat, vulnerability and impact data from the enterprise under analysis and enters that data into the FDS. The FDS allows the data to be organized and managed, and saves summaries in various external files that become input files for other analysis tools such as CPNTools, Excel spread sheets and statistical analysis tools.

The results of analysis by tools external to the FDS are saved, again in external files, and are imported back into the FDS for reporting purposes. Specifically, we use the results of certain spreadsheet analyses to create a set of values (val) imported by CPNTools to make vulnerability, threat and impact decisions within the basic CPNet risk model.
The output of a FARES analysis is a set of reports that detail the risks present in the enterprise under analysis, how the risk profile of the enterprise under analysis compares with the global actuarial data, the probability of the risks and how that probability compares with actuarial data, and the expected financial impact of each risk as a percentage of the organization's gross annual revenues.

Figure 1 shows the relationships of the tools used.

Thus, rather than being confined to developing predictive results based upon a small and unreliable local data set using trivial risk measurement techniques, FARES provides a picture of the enterprise in relation to a global profile and bases predictive processes upon accepted and tested statistical forecasting methods.

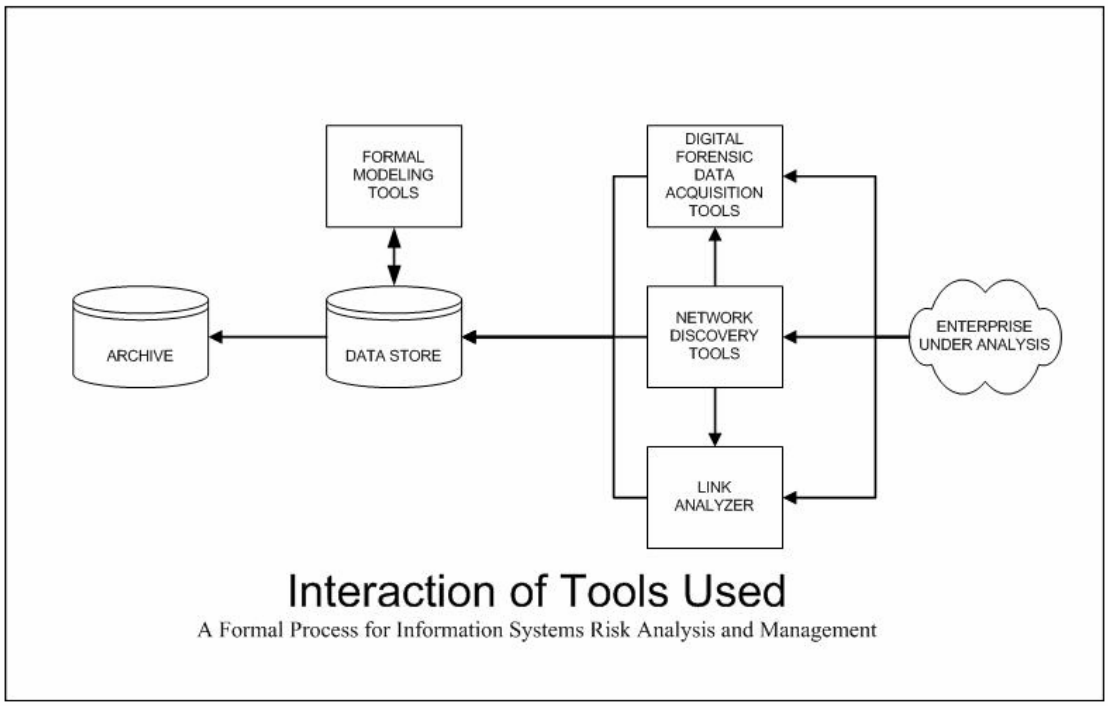


Figure 1 - The Analysis Tool Architecture Used in the FARES Project

3.0 Modeling IS Risk with CPNets

In this paper we focus upon the role of Colored Petri Nets in the FARES process. Other papers on the FARES process address issues other than the modeling of the risk process [PRS03].

3.1 Function of CPNets within the FARES Process

The CPNets used to model risk analysis are based upon the supposition that risk comprises a process and the process can be modeled. The process of risk can be decomposed into threat, vulnerability, impact and inter-domain communications processes. Taken together these processes define a clear risk including risk probability and potential countermeasures.

Interrupting the processes of the components of risk (using countermeasures) reduces the level and probability of the risk itself. This translates to managing risk by managing the components of risk, a task that, in many cases, is far more achievable than managing the entire risk as a unit. Thus, the underlying goal of risk modeling is decomposing risk into its elements, decomposing the elements further into their taxonomy families and comparing those families with historical actuarial data.

We analyze the families that comprise threats, vulnerabilities and impacts for the following relationships:

- The percent ranking of each family within the entire population of collected data (other families) within the threat, vulnerability or impact taxonomy (calculates the relative standing of the family within the overall population)
- The arithmetic mean of the percent rankings
- The standard deviation of the percent rankings population of collected atoms within a particular taxonomy
- Chi-square distribution of specific families within the actuarial data compared to the chi-square distribution of the same atom in the data collected from the enterprise under evaluation (to be added to the models in the future)
- Bayesian probability analysis (to be added in the future)

The CPNet used for proof of concepts of the primary risk model is a hierarchical Net composed of four sub-nets:

- Threats
- Vulnerabilities
- Impacts
- Inter-domain communications

The sub-pages are converged using fusion places onto the superpage where the risk itself is modeled. As the current iteration is a proof-of-concept only, some of the more advanced calculations have yet to be added to the models. The example we use in this paper is a simplified model requiring more manual input and including less sophistication in areas such as probability calculation than will be present in production versions. Additionally, we have put “place holders” in the risk model to include a “Risk Tolerance Function” that permits adjustment of results to match the organization’s tolerance for risk.

3.2 Individual CPNet Pages

The individual components of risk are modeled as sub-pages. As a starting point for describing these pages, we review the declarations for the Net as a whole in Figure 2. Note that some of these declarations would be somewhat different in a production

version that gathered much of its input data from the FDS.

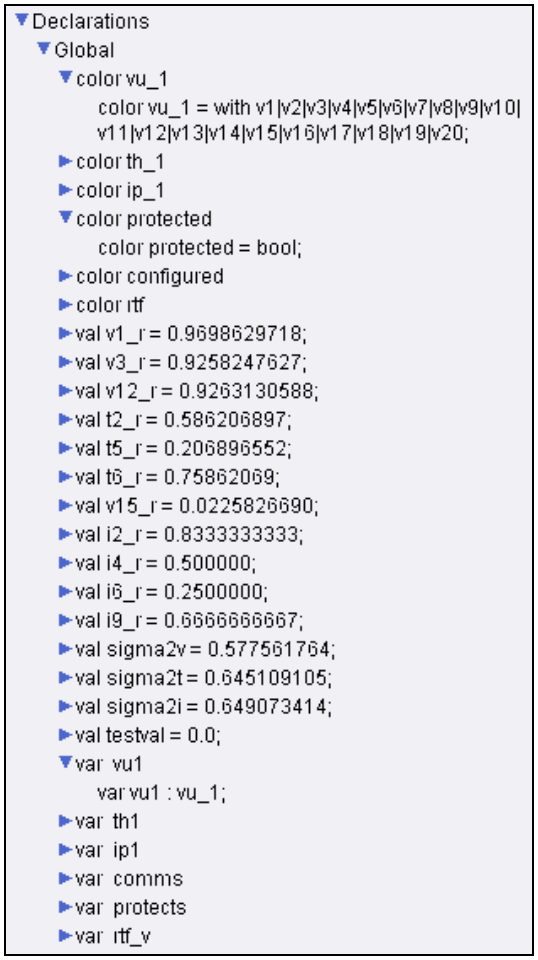


Figure 2 - Global Declarations for IS Risk Model

Color sets *vu_1, th_1* and *ip_1* are similar in their composition. Representing the vulnerability (*vu_1*), threat (*th_1*) and impact (*ip_1*) taxonomies, they contain the individual families of those taxonomies.

For example, there are 56 families in the vulnerability taxonomy (the taxonomies are cataloged by classes which are composed of families which, in turn comprise elements that are made up of atoms). The IS Risk model uses the Family level of abstraction. The vulnerability taxonomy has a tree-like structure.

The values (val) shown in the declarations in Figure 2 have been entered manually for the purposes of this proof-of-concept. However, in a production environment they would be read from an external file (*use [filename]*) created by the FDS. These values are percent rankings of the data collected from the enterprise under analysis, the 2Σ (2 standard deviations) points of the percent rankings and zero (*testval*). These values are used to determine the level of the individual family relative to the collected data.

For example, the model looks for percent rankings within the 2Σ points. These percent rankings represent individual atoms ranking in the top 95% or so of all of the discovered threats, vulnerabilities or impacts. This number has several uses including comparing with the same relative value in the actuarial data as part of risk profile matching.

For the purposes of the Risk Model, however, we use these figures to determine if an occurrence in the collected data is of interest. For average enterprises it is sufficient to keep risks outside the 2Σ points.

However, there is one difference between the model and the normal use of standard deviation. While standard deviation measures distance from the mean of a set of values, we use the standard deviation in a calculation relative to zero. We do this because we are interested in the difference between the percent ranking and the standard deviation.

An example value acceptability calculation for a vulnerability is shown below.

For taxonomy family "Interface Inconsistency" there are 7,655 occurances within the collected (not actuarial) data. This family has a percent rank within the collected dataset of .96968629718 (the rank of the value within the dataset as a percentage of the dataset).

The 2Σ point of the dataset is .577561764

Percent Rank – 2Σ > 0 = an unacceptable value

.96968629718 – .577561764 > 0

Calculation of Value Acceptability for a Taxonomy Family appearing in Collected Data

We apply this calculation approach in each of the CPNet subpages representing threat, vulnerability or impact. The outputs of those pages, to the risk model superpage, act as inputs to the final risk decision. The output of the risk model is written to an external file for use by the FDS in further calculations and reports.

Figure 3 below shows the vulnerabilities subpage.

For the purposes of this proof-of-concept we have kept the number of families to a small number and we have expanded the graphics for illustrative purposes. However, in the production version, there are place/transition pairs (Vuln_n/VT_n) for each of the 56 taxonomy families in vulnerabilities, 30 in threats and 13 in impacts. The values are extracted from an external data file populated by the FDS and the calculations appear as guards on each of the transitions as shown in Figure 3.

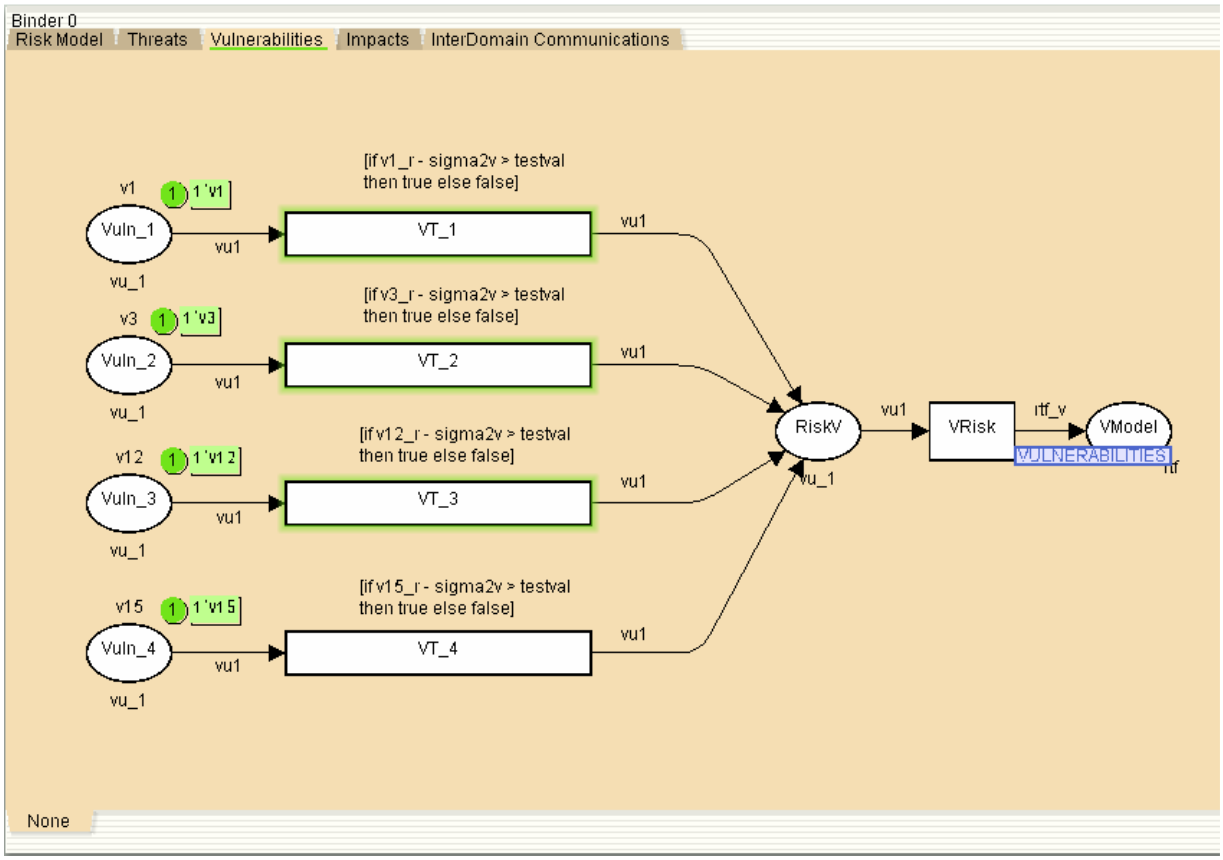


Figure 3 - Vulnerabilities Subpage

In Figure 3 we see that transitions VT_1, VT_2, and VT_3 are enabled. This is because the calculation (the guard on each transition) as shown in Figure 4 will yield a positive value. This means that the number of occurrences for that particular family is at an unacceptable high level.

Taken together, the values calculated for all of the vulnerability families will be passed to the final risk calculation on the superpage as well as to an external file for further use in the FDS. The place VModel is a fusion place connecting the superpage.

There is a similar fusion place on each of the other subpages. Transition VRisk is reserved for adding an additional risk tolerance function analysis in the future. There is a similar transition on each subpage.

The *sigma2* values in the declarations represent the 2Σ points for the collected data from the enterprise under evaluation and, as with the other values in the declarations, are passed from the FDS through an external file.

3.2.1 The Communications Page

The Vulnerabilities, Threats and Impacts subpages behave similarly. However, there is an additional element to risk that involves inter-domain communications channels.

In Figure 4 we show the inter-domain communications model subpage. This model addresses the communications channels to the security policy domain Core. In most organizations, the core comprises one of the most critical and sensitive security domains in the enterprise.

Control of access to the core, whether by users or processes, must be carefully

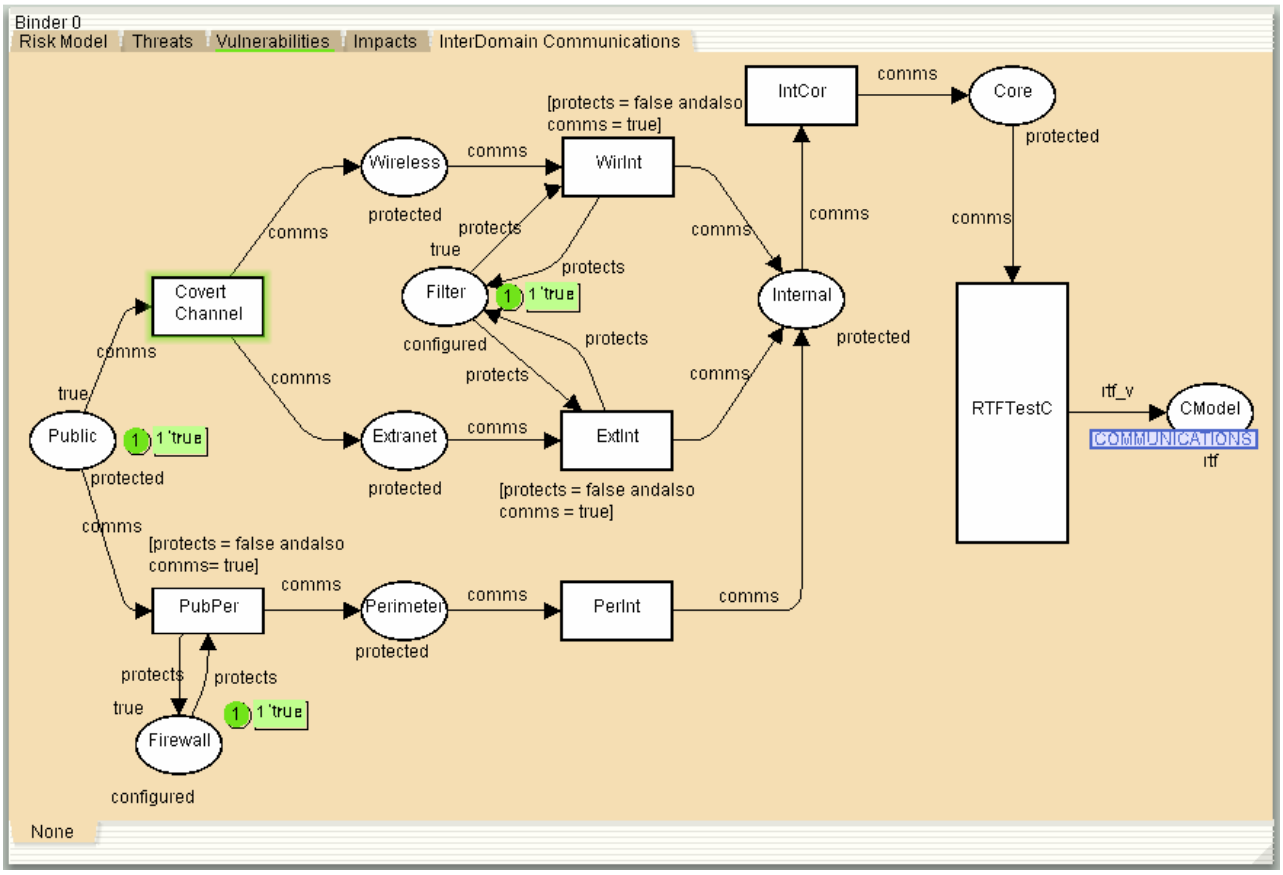


Figure 4 - The Inter-Domain Communications Model Subpage

managed. By accessing the core it is possible for intruders to gain access to other domains and rogue code such as viruses and worms can be spread throughout the rest of the enterprise using the capabilities of devices in the core. The model in Figure 4 is typical of the models that would accompany each of the inter-domain communications channels discovered in the assessment phases of the enterprise under analysis. Each of these models would appear as an additional subpage with fusion places connecting them to the risk model superpage. For this example we show only a single typical inter-domain model.

In the Figure 4 model, we see that the entry point is the Public domain. That means that we have modeled the inter-domain communications channels that connect, or could connect, the Public domain to the Core. The implication is that an attacker in the Public domain (for example, the Internet) could gain access to the Core domain if appropriate access control countermeasures are not in place.

The data that results in this model comes from the assessment phases. Some is derived from information received during interviews regarding the use and flow of various types of data within the organization. Some information, predominantly topological data, is used to map the logical domains and their physical interconnections. Finally, link analysis is used to determine all of the possible paths between domains.

At the same time, vulnerability assessment reveals countermeasures in place and general threats, impacts and vulnerabilities relative to individual domains. That information is reflected in the threat, vulnerability and impact subpages while anything affecting data flows is reflected in the inter-domain communications subpage(s).

The Figure 4 model represents inter-domain communications channels as transitions (the actions that data moving between domains take). Security policy domains (the logical state of the enterprise) are represented by places.

A quick analysis of Figure 4 shows that there is a typical channel from the public Internet to the Perimeter and thence to the Internal network and, ultimately, to the Core.

Clearly, we want to prevent unauthorized users on the public Internet from having direct access to the Core and the Internal network, so we interpose a safeguard called Firewall. Our vulnerability assessment showed that this firewall did, in fact, exist and that it was configured correctly. We reflect that in the application of the Firewall place as an inhibitor on the PubPer (Public to Perimeter) channel (transition), closing that path.

Internally, we see that there is a Covert channel from the Public domain to the Wireless and Extranet domains. We refer to this as a "covert channel" because there should be no authorized connection from the public Internet to the internal network via wireless communications of through an extranet that connects to a business partner.

To protect the Internal (and, thus, the Core) domain from unintended (and unauthorized) wireless and extranet connections, we insert another safeguard, this time a filtering router, as an inhibitor. This safeguard, represented by the Filter place, is applied to the WirInt (Wireless to Internal) and ExtInt (Extranet to Internal) channels (transitions).

Again, our vulnerability assessment indicated that the filter was in place, configured properly and performing effectively. Thus, we add it to the model. The initial marking of the Public domain is a single token with a value of *true* (indicating that the state of the place is protected). This may appear to be a bit of a philosophical stretch since most people don't view the Internet as protected.

However, for our purposes and for consistency, we have considered that the state of the Internet domain is that it is operating correctly and that the link to the organization's perimeter has not been disrupted. In other words, the public Internet does, in fact, connect to the organization's perimeter and it and the connection are working properly.

The communications models in our examples use Boolean states almost exclusively since at this level of abstraction we are concerned largely with whether the channel passes data or not. However, in production versions it may be desirable to build a somewhat more complex model that reflects additional granularity of detail by modeling the protocols in use on the individual channels.

The two safeguards show single tokens indicating that they are configured properly and ready to function. There are arcs in both directions so that the state of the safeguard may be refreshed each time a state change is required to protect its transition.

Transition RTFTestC is a place holder for future implementation of the Risk Tolerance Function as with the other subpages. Place CModel is the fusion place connecting this subpage to the superpage risk model.

3.2.2 The Risk Superpage

The purpose of the Risk page is collection and processing of the individual taxonomy data as well as the inter-domain communications channel data. The threats, vulnerabilities, impacts and potentially compromised communications channels yield data that is presented to the superpage as Boolean states. These are then passed to the *Correlation* place and, after passing the *Correlation Transfer* transition to the *Risk* place for final display.

For the purposes of this proof-of-concept, we have left the *Correlation Transfer* transition as a place holder for additional analysis capability in the future. The intent of the place holder transitions throughout the Net is to add calculations for Risk Tolerance Factor (RTF) once research on that aspect of the modeling is complete. The Risk superpage is shown in Figure 5. It is in its final state after simulating the enterprise represented on its subpages with six risks identified. The subpages may be referred to in order to ascertain those transitions that did not fire, representing adequate operational controls in place.

4.0 Using the FARES CPNet Model for Ongoing IS Risk Management

One of the strong benefits of CPNets in the IS risk analysis and management arena is the ease with which they can be updated and new simulations cast. The information technology requirements of large organizations are in a constant state of change and the costs associated with repetitive, frequent testing usually are prohibitive. Therefore, it is not uncommon for such organizations to skip security and risk testing of new implementations of hardware or software.

Additionally, new threats and vulnerabilities emerge daily. Again, most organizations are hard-pressed to stay current. A typical response to a newly announced vulnerability is the application of patches to affected information systems elements enterprise-wide. This usually is not practical due to the large number of devices and/or software that must be patched.

For this reason, organizations use automated methods to "push out" patches without testing for conflicts with existing software

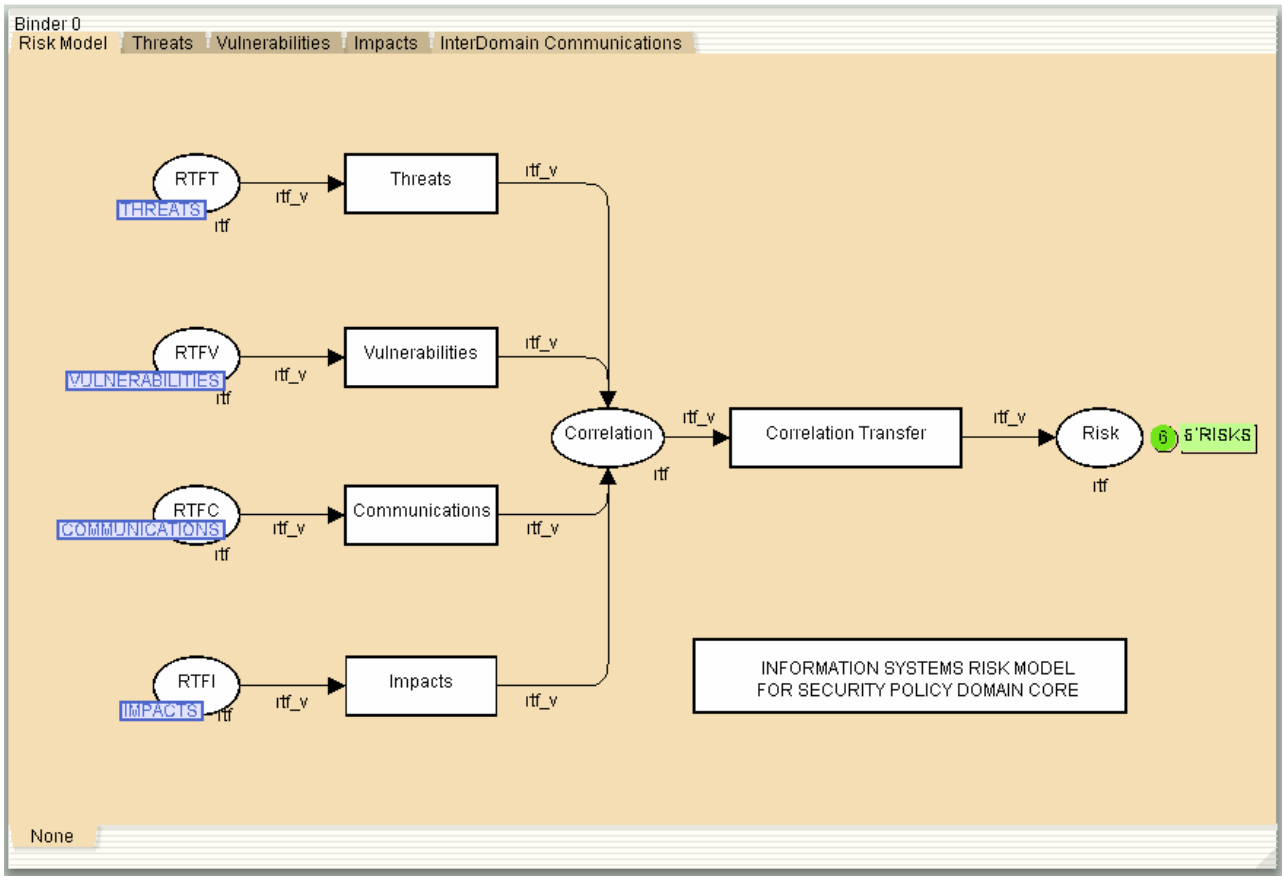


Figure 5 - Risk Superpage After Simulation

applications and hardware configurations. The ability to add to or change risk models as necessity dictates and re-run simulations is a significant cost and risk management capability.

The cooperation between CPNet models and statistical models allows IS risk managers to predict, on an ongoing basis, the potential financial impact of changes to models due to changes in the organization's IS risk profile. This greatly improves financial and operational strategic management capability.

In terms of real time response to incidents, the use of formal models and simulations allows organizations to model a complex event in progress and respond rapidly with a high probability of success.

Applied to post incident root cause analysis, the underlying causes of information security incidents can be analyzed and appropriate countermeasures applied against future impacts.

Impacts of information security incidents can be predicted and budgeted for should the organization wish to manage a risk down to acceptable levels.

5.0 Conclusions

The use of formal modeling and simulation in assessing, analyzing and managing information systems risk appears to offer strategic, tactical, operational and financial benefits. The use of CPNets as a modeling and simulation formalism is appropriate for a variety of reasons.

The formalism is well-supported by available freeware tools. The output of CPNTools is graphical, making the simulations accessible to lay audiences. Laboratory tests of the FARES risk analysis and management process have suggested that the approach is sound.

This is supported by early field trials on live information systems [PS03]. Finally, information security and risk management practitioners have received the notion of CPNets and the FARES process positively, suggesting that practical application is reasonable.

5.1 Future Work

The primary area for continued development is the integration of statistical functions, both within the CPNet model and external to it, with the modeling and simulation processes.

Currently, this analysis is performed manually using additional tools and entering the data into CPNTools manually. Additionally, full integration of all modeling, simulation and data management tasks under a single user interface is desirable.

The author is a relative novice with CPNets and it is clear that there is room for a significant increase in the sophistication of the models once the early field trails of the FARES approach are complete. At this point the project will be able to identify clearly the additional areas where CPNets can offer additional benefit to the IT risk analysis and management process.

Finally, there is an ongoing need to continue to research and populate global profiles with historical actuarial reference data. As the amount and quality of such data increases, the reliability of statistical conclusions will improve.

6.0 References

[AJ02] Jones, Andrew. "Identification of a Method for the Calculation of Threat in an Information Environment" Internal publication, QinetiQ, Inc. April 2002.

[DEC02] Decisioneering. "Traditional Spread Sheet Risk Analysis", web page last accessed19 September 2004. < http://www.decisioneering.com/risk-analysis-trad.html>

[FD04] Fernandes, Joao M, Francisco J. Duarte. "A Reference Framework for Process-Oriented Software Development Organizations". Springer-Verlag, July 2004.

[GP04] Gladyshev, P., A. Patel. "Finite State Machine Approach to Digital Event Reconstruction", Digital Investigation, Volume 1 Number 2, pp130-149, Elsevier, Ltd.

[JKL02] Jones, Andy, Gerald L. Kovacich, Perry G. Luzwick. Global Information Warfare, Auerbach Publications, 2002

[PS03] Stephenson, Peter. "Modeling of Post Incident Root Cause Analysis", International Journal of Digital Evidence, Fall 2003 issue. <http://www.ijde.org/archives_home.html> last accessed 19 September 2004.

[PRS03] Stephenson, Peter. "Getting the Whole Picture" tutorials in Computer Fraud and Security, Elsevier Science. December 2003, January 2004, February 2004, March 2004, April 2004, June 2004.

[PRS04] Stephenson, Peter. "Application of Formal Methods to Root Cause Analysis of Digital Incidents". International Journal of Digital Evidence Fall 2004 (publication ending) <http://www.ijde.org>

[WA98] van der Aalst, W. M. P. "The Application of Petri Nets to Workflow Management" Journal of Circuits, Systems and Computers, Vol. 8, No. 1 (1998) 21-66

Author Biography

Peter Stephenson, CISSP, CISM, FICAF, is the Director of Information Assurance and a research scientist at the Center for Regional and National Security, Eastern Michigan University. He has over 40 years experience with technology including over 20 years of information security experience. He holds a BSEE and recently completed PhD work in computer science at Oxford Brookes University in the UK where his thesis topic was "Structured Investigation of Digital Incidents in Complex Computing Environments".

He can be contacted by email at peter.stephenson@emich.edu.

Relating Higher Order Reference Nets and Well-Formed Nets

Lawrence Cabac and Michael Köhler

University of Hamburg, Department of Informatics
Vogt-Kölln-Str. 30, D-22527 Hamburg
Phone: +49 40 42883-2407 Fax: +49 40 42883-2246
{cabac,koehler}@informatik.uni-hamburg.de

Abstract. In this presentation we introduce the formalism of "Higher Order Reference Nets" (HORNETS). HORNETS follow the paradigm of "nets within nets", i.e. the paradigm that allows Petri nets as token objects.
Since all net tokens are objects of some net class they all share the same structure. In our contribution we therefore introduce the notion of well-formed HORNETS which can be simulated by well-formed coloured Petri nets. This allows for several analysis techniques, e.g. symbolic state space generation, which automatically takes the system symmetries into account.

1 Introduction

Object net systems (ONS) [Val98] are well suited to model systems that have a dynamic hierarchical structure, e.g. systems of mobile agents [KMR03]. Object net systems are based on the idea that the tokens of a Petri net are Petri nets again. In general, this extension increases the computational power of Petri nets [KR04]. So, for analysis purposes we are interested in special subclasses of object net systems. In this contribution we present the formalism of HORNETS that can be mapped to well-formed nets [CDFH90] – a coloured Petri net [Jen92] fulfilling additional constraints.

Taking an ONS example from [Val03]: the *Bucket-Chain*. The original bucket chain-scenario has been introduced by Carl Adam Petri [Pet79] to study the causal dependencies of distributed cooperation. The scenario serves a similar purpose as the well known *Bankers-Problem* [PS85] for deadlock-prevention in resource allocation systems (i.e. operating systems) or the *Dining Philosophers* [PS85] for the study of fairness in distributed systems.

In the bucket chain example n firemen are standing in a row, each equipped with a bucket. A water pump is available for the leftmost fireman and the fire is at the rightmost place. So the leftmost fireman fills his bucket, while the rightmost extinguishes the fire. Neighboured firemen can exchange buckets, so full buckets are handed over to right (to extinguish the fire) and empty ones to the left for refilling. The topology (i.e. each fireman can only interact with his immediate neighbour) introduces an interesting causal dependency structure[1]: The effect of exchanging buckets at a location being k steps away can be observed only when the whole system has moved k steps ahead.

[1] In the general research of Petri this causal dependency structure is closely related to Einstein's physical theory of relativity. This topic is studied in Petri's research of general net theory.

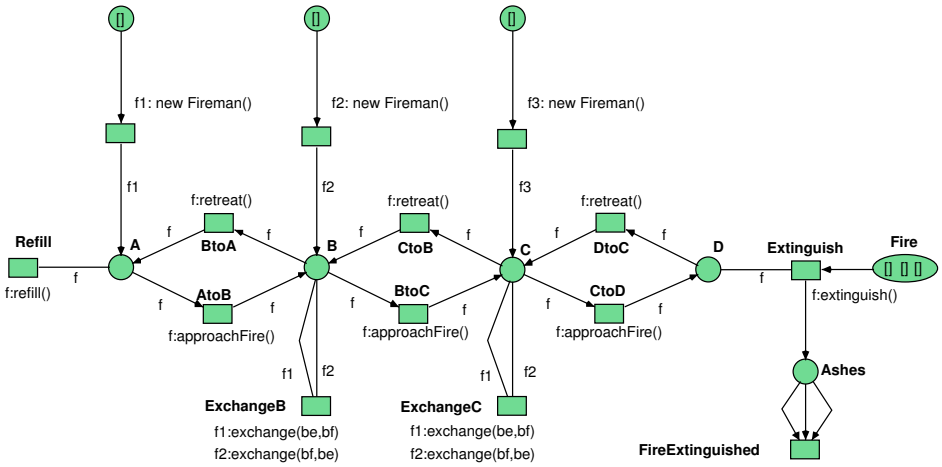


Fig. 1. The Bucket Chain

Reference nets are a powerful modelling formalism to represent the different abstraction layers of the system. The model that is presented here is from [Val03]. The bucket chain is shown in Fig. 1, the fireman in Fig. 2, and the bucket in Fig. 3.[2]

The bucket chain has four locations, named A, B, C, and D. Initially, three firemen are created on A, B, and C. The firemen can refill water in location A and extinguish the fire on D. If two firemen are located on the same place (here: B or C) they can exchange buckets. The local synchronisation of firemen is implemented using synchronous channels. In the bucket chain the calling side (the *downlink*) of the channel exchange is used twice– expressed by the transition inscription f1: exchange(be, bf) f1:exchange(bf, be). The intuitive meaning is the following: Fireman f1 must provide an empty bucket be and and f2 a full bucket bf – the first variable in the argument list. The binding mechanism is used to exchange the buckets.

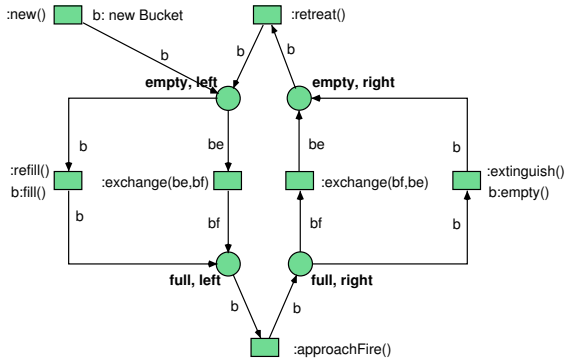


Fig. 2. The Fireman

[2] Actually the model is implemented using the Petri net tool Renew [KWD+04].

The fireman is modelled as net presented in Fig. 2. He has four different states depending on whether the bucket is full or empty and whether the fireman has already moved or not. If the bucket is empty, the firemen can either refill the bucket (only on location A), exchange the bucket with a fireman carrying a full one (location B or C), or if he has not moved with the empty bucket, the fireman can retreat from the fire. The transition inscription :exchange(be, bf) denotes the called part (the *uplink*) of the channel exchange. For the full bucket the system behaves in a similar way.

The bucket in Fig. 3 has only two states: full and empty. These states can be tested and are modified by refilling or extinguishing operations.

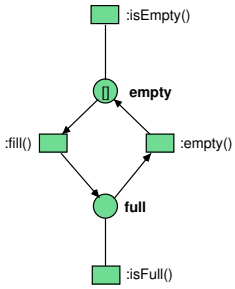


Fig. 3. The Bucket

In the Section 2 we introduce the notations used throughout the paper. In Section 3 the HORNET formalism is introduced. In Section 4 we show how the class of well-formed HORNETS has to be defined such that they can be simulated by well-formed coloured Petri nets. In Section 5 we take up our introducing example – the bucket-chain to illustrate the simulation.

2 Notations and Basic Definitions

Let $R \subseteq A \times B$ be a relation. A pair $(a, b) \in R$ will also be denoted $a\,R\,b$ in infix notation. For $a \in A$ and $b \in B$ the domain of B is defined by $(\_\,R\,b) := \{a \mid (a, b) \in R\}$ and its co-domain by $(a\,R\,\_) := \{b \mid (a, b) \in R\}$. We generalise the notion of domain and co-domain to sets $C \subseteq A$ and $D \subseteq B$ by $(C\,R\,\_) := \{b \mid \exists a \in C : (a, b) \in R\}$ and $(\_\,R\,D) := \{a \mid \exists b \in D : (a, b) \in R\}$.

Let $\mathcal{A}$ be a family of sets. Then $A^\star$ denotes the set of all Cartesian products over the elements of $\mathcal{A}$.

The definition of Petri nets relies on the notion of multisets. A multiset on the set D is a mapping $A : D \to \mathbb{N}$. Multisets are generalisations of sets in the sense that every subset of D corresponds to a multiset A with $A(x) \leq 1$ for all $x \in D$. The empty multiset 0 is defined as $0(x) = 0$ for all $x \in D$. The cardinality is $|A| := \sum_{x \in D} A(x)$. A multiset A is called *finite* iff $|A| < \infty$.

The multiset sum $A + B$ is defined as $(A + B)(x) := A(x) + B(x)$ the difference $A - B$ by $(A - B)(x) := \max(A(x) - B(x), 0)$. Equality $A = B$ is defined element-wise: $\forall x \in D : A(x) = B(x)$. Multisets are partially ordered:

$A \leq B \iff \forall x \in D : A(x) \leq B(x)$ The strict order $A < B$ holds iff $A \leq B$ and $A \neq B$. The notation is overloaded, being used for sets as well as multisets. The meaning will be apparent from its use.

The set of all finite multisets over the set D is denoted $MS(D)$. A multiset A can be considered as the formal sum $A = \sum_{x \in D} A(x) \cdot x$. Finite multisets are the freely generated commutative monoid $(MS(D), +, 0)$. If the set D is finite, then a multiset $A \in MS(D)$ can be represented equivalently as a vector $A \in \mathbb{N}^{|D|}$.

Any mapping $f : D \to D'$ can be generalised to a mapping $f : MS(D) \to MS(D')$ on multisets:

$$f\left(\sum_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} f\left(a_i\right)$$

This includes the special case $f(0) = 0$. These definitions are in accordance with the set-theoretic notation $f(A) = \{f(a) \mid a \in A\}$.

2.1 Petri Nets

$N = (P, T, F)$ is a *Petri net* iff the set of places P and the set of transitions T are disjoint, i.e. $P \cap T = \emptyset$, $F \subseteq (P \times T \cup T \times P)$ is the flow relation. Some commonly used notations for Petri nets are ${}^\bullet y := (\_\, F\, y)$ for the *preset* and $y^\bullet := (y\, F\, \_)$ for the *postset* of a net element y.

A P/T net is an extension of a Petri net, in which the arcs are inscribed by non-negative integers and places can hold more than one token.

Definition 1. *A* P/T net N *is a tuple* $N = (P, T, \partial_0, \partial_1, M_0)$*, such that:* P *is a finite set of places.* T *is a finite set of transitions, with* $P \cap T = \emptyset$*.* $\partial_0, \partial_1 : T \to MS(P)$ *are the pre- and post-condition functions, resp. The multiset* $M_0 \in MS(P)$ *is the initial marking of* N*.*

A transition $t \in T$ of a P/T net $N = (P, T, \partial_0, \partial_1, M_0)$ is enabled in the marking M iff enough tokens are present: $M \geq \partial_0(t)$. The successor marking when firing t is $M' = (M - \partial_0(t)) + \partial_1(t)$. We denote the activation of t in marking M by $M \xrightarrow[N]{t}$. Firing of t is denoted by $M \xrightarrow[N]{t} M'$. The notation extends to firing sequences $w \in T^*$.

Defining $F := \{(p, t) \mid \partial_0(t)(p) > 0\} \cup \{(t, p) \in| \ \partial_1(t)(p) > 0\}$ and $W(p, t) = \partial_0(t)(p)$ and $W(t, p) = \partial_1(t)(p)$ a P/T net $N = (P, T, \partial_0, \partial_1, M_0)$ can be denoted equivalently as $N = (P, T, F, W, M_0)$. The two notations will be treated as interchangeable throughout this paper.

2.2 Synchronisation

We are using the synchronisation mechanism provided by so-called zero-safe places introduced by Bruni and Montanari [BM97,BM00]. A subset of places $Z \subseteq P$ is used to characterise those markings that are intermediate states being visible only during a transaction. For more details cf. [KF04]. These places are called *zero-safe* since, in all "visible" markings, these places are unmarked.

Let $N = (P, Z, T, F, W, M_0)$ be a net component where (P, T, F, W, M_0) denotes a P/T net as usual and $Z \subseteq P$ is the set of zero-safe places. The places in $P \setminus Z$ are called *stable*.

Definition 2. *Let N be a net component. A firing sequence $w = t_1 \cdots t_n \in T^*$ such that*

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M_n$$

with $M_i \in MS(P)$ is a stable sequence *iff*

- *the transition sequence is enabled as a whole wrt. stable places: $\forall s \in (P \setminus Z) : \sum_{i=1}^{n} \partial_0(t_i)(s) \leq M_0(s)$ and*
- *M_0 and M_n are stable markings, i.e., $\forall z \in Z : M_0(z) = M_n(z) = 0$.*

A stable sequence w is called a transaction *iff none of the intermediate markings $M_1, \ldots, M_{n-1}$ is stable:*

$$\forall 1 \leq i \leq n-1 : \exists z \in Z : M_i(z) > 0$$

A transaction w is called closed *iff all permutations $\pi(w)$ that are activated in M_0 are transactions, too. Let $\Theta(N)$ be the set of all closed transactions of a net component N.*

Note, that the activation condition for a stable sequence requires that the transitions $t_1, \ldots, t_n$ are concurrently enabled if only resource places $s \in P \setminus Z$ but no zero-safe places $z \in Z$ are considered.

Note, that all stable sequences $M \xrightarrow{w} M'$ transform a stable marking M (i.e. a marking without tokens on zero-safe places: $M(z) = 0$ for all $z \in Z$) into a stable marking.

3 Object Net Systems and HORNETS

Object Net Systems (ONS) [Val98,KR04] are Petri nets that have Petri nets as tokens. In the following we introduce a high-level variant where net-tokens are coloured, i.e. they are instances of object nets.[3] Let $\mathcal{N} = \{N_1, \ldots, N_k\}$ be a finite set of object net sorts. N_1 is called *system net* by convention and describes the top-level of the system. In the following we use families of disjoint sets indexed by object net sorts $N \in \mathcal{N}$ denoted as A_N. Let $A = \bigcup A_N$ for an arbitrary family.

Each sort $N \in \mathcal{N}$ is also used to denote a Petri net $ON_N = (P_N, Z_N, T_N, F_N)$ where P_N is a finite set of places, $Z_N \subseteq P_N$ is a set of zero-safe places used for synchronisation, T_N is a finite set of transitions, and $F_N \subseteq (P_N \times T) \cup (T \times (P_N \cup Z))$ the flow relation. Note, that the flow allows to call channels $z \in Z$ of other nets. To simplify notations we identify ON_N with N in the following.

[3] In fact, HORNETS are a special case of reference nets of Kummer [Kum02], which allow additional data types. Here, we restrict the model to ensure well-formedness, similarly to coloured nets and well-formed nets.

Since we are interested in an embedding into well-formed nets, we assume only finitely many identifiers.[4] For each $N \in \mathcal{N}$ we assume a set $O_N = \{o_{N,1}, \ldots, o_{N,k_N}\}$ of identifiers.

Each place p is inscribed with its type $d(p) \in \mathcal{N}^\star$, i.e. a Cartesian product over $\mathcal{N}$. The map d is extended for arcs by defining $d(p,t) = d(p)$ and $d(t,p) = d(p)$. Let $X_{d(p)} := X_{N_1} \times \cdots \times X_{N_k}$ and $O_{d(p)} := O_{N_1} \times \cdots \times O_{N_k}$ whenever $d(p) = N_1 \times \cdots \times N_k$.

For each $N \in \mathcal{N}$ we assume an countable infinite set $X_N = \{x_{N,1}, x_{N,2}, \ldots\}$ of variables. Each arc $f \in F$ is inscribed by a multiset of variable-tuples of the type $d(f)$, i.e. $W(f) \in MS(X_{d(f)})$.

We use directed channels which is expressed by the fact that each arc $(t,z) \in F$ is labelled with an variable $l(t,z) \in X$ that is already used in the preset expressions.[5]

The set of *guard expressions* $Prop \in \mathbb{T}^{bool}$ consists of all propositional formulas with equality of variables as atoms and is given by the following grammar. Let $i, j \in \mathbb{N}$:

$$Prop ::= (x_{N,i} =_N x_{N,j}) \mid \neg Prop \mid (Prop \vee Prop)$$

A marking is a map $M : P \to MS(O^\star)$ such that $M(p) \subseteq MS(O_{d(p)})$.

Definition 3. *Let $\mathcal{N}$ be a finite set of net sorts. A* higher order reference net system *(HORNET) is a tuple*

$$OS = \big(ON_N, d, \{O_N\}, \{X_N\}, W, l, G, M_0\big)$$

- $ON_N = \{(P_N, Z_N, T_N, F_N)\}_{N \in \mathcal{N}}$ *is a family of object nets.*
- $\{O_N\}_{N \in \mathcal{N}}$ *is a family of net identifiers.*
- $\{X_N\}_{N \in \mathcal{N}}$ *a family of identifier variables.*
- *Let* $d : P(OS) \to \mathcal{N}^\star$ *be a type mapping of places to tuples of object nets.*
- $W : F \to MS(X)$ *is the arc inscription with* $W(f) \in MS(X_{d(f)})$.
- $l : F \cap (T \times Z) \to X$ *is the channel inscription.*
- $G : T \to Prop$ *is the guard predicate.*
- $M_0 : P \to MS(O^\star)$ *is the initial marking with* $M_0(p) \subseteq MS(O_{d(p)})$.

Define $N(p) = N_i \iff p \in P_{N_i}$. Similarly for $N(z)$, $N(t)$ etc.

The object nets (P_N, Z_N, T_N, F_N) can be regarded as classes. The net object are instances of these classes sharing the same structure but may differ in the actual state. This can be represented by colouring all net elements with object identifiers. This approach is very common: [Lak95] presents a formalism that extends the coloured Petri nets of [Jen92]. Similarly [BG91] extend algebraic Petri nets. [MM97] define an embedding of objects into coloured Petri nets.

To discriminate between instances, we use identifiers as a prefix for places, transitions, markings etc. An instance is described by an identifier $o \in O_N$

[4] We assume a finite set of identities, since it is proven in [Kum00] that Object Net Systems with an infinite set of identities have the power of Turing machines. Here, we are interested in a more restricted model.

[5] The syntax used in the tool RENEW for denoting an arc (t,z) with $W(t,z) = (x_1, \ldots, x_n)$ and $l(t,z) = x$ is $x : z(x_1, \ldots, x_n)$ as an inscription of t.

taken from a finite set O_N for each $N \in \mathcal{N}$. Define the prefix operation on places, where $o.p$ means the place p of the instance o. Similarly for $o.t$ etc. The operation extends linearly to multisets: $o.(m_1 + m_2) = o.m_1 + o.m_2$.

The initial marking M_0 is extended by:

$$M_0(o'.p)(o'') = M_0(p)(o'')$$

The notation extends to preconditions by defining:

$$\partial_0(o.t)(o'.p) = \begin{cases} \partial_0(t)(p), & \text{if } o' = o \quad \wedge\, p \in P \setminus Z \\ \partial_0(t)(p), & \text{if } o' = o \quad \wedge\, p \in Z \\ 0, & \text{otherwise} \end{cases}$$

and to postconditions by defining:

$$\partial_1(o.t)(o'.p) = \begin{cases} \partial_1(t)(p), & \text{if } o' = o \qquad \wedge\, p \in P \setminus Z \\ \partial_1(t)(p), & \text{if } o' = l(t,p) \wedge\, p \in Z \\ 0, & \text{otherwise} \end{cases}$$

Note, the special treatment of calling channels (t, p) with $p \in Z$.

A variable assignment is an indexed map $\alpha_N : X_N \to O_N$ which extends to terms the usual way. A variable assignment α is called a binding for a transition $o.t$, iff α fulfils the guard, i.e. $\alpha(G(t))$ is true. A transitions $o.t$ is *pre-activated* in M wrt. a binding α iff

$$M(o'.p)(o'') \geq \alpha\left(\partial_0(o.t)(o'.p)\right)(o'')$$

for all $o'.p$ and o''. The successor marking $M'(o'.p)(o'')$ is defined as:

$$M'(o'.p)(o'') = (M(o'.p)(o'') - \alpha\left(\partial_0(o.t)(o'.p)\right)(o'')) + \alpha\left(\partial_1(o.t)(o'.p)\right)(o'')$$

A pre-activated firing sequence $w = \alpha_1(x_1.t_1) \cdots \alpha_n(x_n.t_n)$ is *activated* iff w is a transaction, i.e. it clears all synchronisation places. Transactions of a HORNET are of the form $w = \alpha_1(o_1.t_1) \cdots \alpha_n(o_n.t_n)$.

Note, that there is no such RENEW-like construct as `x: new NetToken` for HORNETS, since as Kummer has shown the unbounded use of identifiers leads to undecidability results of almost all relevant problems related to Petri nets, since counters can be simulated (cf. [Kum00]). Therefore, for HORNETS only a bounded number of instances are allowed to be created. The maximal number of instances is determined through the initial marking. For this, a place initially holds a set of free identifiers.

4 Expressing HORNETS as Coloured Petri Nets

In this section we define a class of well-formed HORNETS and provide an embedding into well-formed coloured Petri nets.

4.1 Well-formed Nets

A well-formed net (WN) [CDFH90] is a coloured Petri net [Jen92] with additional constraints. The constraints ensure that the net is well structured such that analysis techniques can be applied efficiently.

Tokens of WN are tuples of coloured objects taken from a non-empty set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of finite *basic colour classes*. If the objects do not behave all the same way a colour class may be partitioned into disjoint subclasses: $C_i = (C_i^1 \cup \ldots \cup C_i^{k_i})$. A basic colour class C_i may be ordered. In this case the order is total and cyclic.

The basic operations on tokens are: *projections* of tuples $(x_1, \ldots, x_k)$ onto one of its components x_i, the *successor* $!x$ of an object x (if the colour class is ordered), and the *diffusion/synchronisation* denoted as $C_i.all$, denoting the multiset containing all the elements of C_i. Analogously for subclasses: $C_i^j.all$.

The colour domain $d(p)$ of a place p is a Cartesian product of basic colour classes (where e_i denotes the numbers of variables of the colour C_i):

$$cd(p) = \times_{C_i \in \mathcal{C}}(C_{j,1} \times \cdots \times C_{j,e_i})$$

The colour domain $cd(t)$ of a transition t (its firing modes) is the set of variable bindings α that is compatible with the guard predicate:

$$cd(t) = \{\alpha \mid \alpha(G(t))\}$$

Guards of a WN are defined as Boolean expressions with the atomic expression: $x = y$, $x = !y$, $d(x) = C_i^j$ and $d(x) = d(y)$, where x, y are variables and $d(x)$ denotes the (sub-) class x belongs to.

Definition 4. *A well-formed net is a tuple*

$$WN = (P, T, \partial_0, \partial_1, \mathcal{C}, cd, M_0)$$

where

1. *P and T are finite sets of places and transitions.*
2. *∂_0, ∂_1 are the arc mappings with $\partial_0(t)(p), \partial_1(t)(p) : cd(t) \to MS(cd(p))$*
3. *$\mathcal{C} = \{C_1, \ldots, C_n\}$ is the colour set of finite* basic colour classes.
4. *cd maps each place p to a Cartesian product of basic colour classes and each transition t to a set of firing modes.*
5. *$M_0 : P \to MS(\bigcup \mathcal{C})$ with $M_0(p) \in MS(cd(p))$ is the initial marking.*

4.2 Well-formed HORNETS

For a simulation of a HORNET by a well-formed net we have to ensure that the set of all synchronisations is finite. This is ensured by the following restrictions:

Definition 5. *A* HORNET *OS given as in Def. 3 is* well-formed *iff the following holds:*

1. *The net structure of (P, T, F), when considering the restriction to $Z \subseteq P$, is acyclic, i.e. $F \cap ((Z \times T) \cup (T \times Z))$ is an acyclic relation.*

2. *The synchronisations are restricted to tree-like structures:* $|{}^\bullet t \cap Z| \leq 1$ *for all* $t \in T$.
3. *All variables on outgoing arcs are already bound in the incoming arcs.*
4. *Also all inscriptions of transitions connected by zero-safe places have to be the same: For all* $t_1, t_2 \in T$ *and all* $z \in t_1{}^\bullet \cap {}^\bullet t_2$ *we have* $W(t_1, z) = W(z, t_2)$.
5. *The variables for all inscriptions of stable places – $W(s,t)$ and $W(t,s)$ are assumed to be chosen pairwise disjoint.*

The conditions have the following intuitive meaning:

1. The condition ensures that synchronisation structures cannot be iterated. Since there are only finitely many transitions the number of synchronisations is finite.
2. Due to the tree-like structure all synchronisations can be constructed by starting with channel-free transitions, i.e. transitions with ${}^\bullet t \cap Z = \emptyset$.
3. The values transferred by the channel must exist in the preset.
4. This condition ensures that the binding is equal on both sides of the channel.
5. This condition is just for convenience. Since a synchronisation cannot be iterated (due to the first condition) every transition appears at most once and the variables of arcs connected to stable places can be bound independently.

The set of all closed transactions $\Theta(OS)$ of well-formed HORNETS is finite and can easily be constructed. So, it can be used as the set of transitions of the simulating net.

4.3 Embedding of Well-formed HORNETS into Well-formed Nets

Each well-formed HORNET can be simulated directly by a well-formed net. The identifier sets $\{O_N \mid N \in \mathcal{N}\}$ are used as colours.

The places in the simulating net are the same as in the HORNET. For each place $p \in P$ of the ONS with type $d(p)$ we have a place p in the WN with the domain $cd(p) = O_{N(p)} \times O_{d(p)}$. Instead of using prefixes like $o_1.p$ marked with o_2 (where o_1 is the identity of the instance and o_2 is the instance referenced) we use pairs (o_1, o_2) as tokens of the place p.

The set of all closed transactions $\Theta(OS)$ is used as the set of transitions of the simulating net.

Definition 6. *Let a* HORNET *be given as:*

$$OS = (ON_N, d, \{O_N\}, \{X_N\}, W, G, M_0)$$

The simulating WN is defined as

$$\text{WN}(OS) = (P, \Theta(OS), \partial_0, \partial_1, \mathcal{C}, cd, M_0')$$

where

1. $\mathcal{C} = \{O_N \mid N \in \mathcal{N}\}$ *– without ordering or partition.*
2. *The colour domain is* $cd(p) = O_{N(p)} \times O_{d(p)}$ *and* $cd(t) = \{\alpha \mid \alpha(G(t))\}$.

3. ∂_0, ∂_1 *are the pre and post mappings with* $\partial_0(\theta)(p)(\alpha) = \sum_{i=1}^{n} W(p, t_i)$ *and* $\partial_1(\theta)(p)(\alpha) = \sum_{i=1}^{n} W(t_i, p)$ *for* $\theta = (x_1.t_1 \cdots x_n.t_n)$.
4. $M_0'(o.p) = M_0(p)$.

Using the bijection WN on markings defined by

$$\text{WN}(M)(p)(o', o'') = M(o'.p)(o'')$$

the well-formed net WN(OS) provides as direct simulation of a well-formed HORNET OS.

Proposition 1. *Let* OS *be an well-formed* HORNET. *A transition* $\theta \in \Theta(OS)$ *is activated in* OS *iff it is activated in* WN(OS)*:*

$$M \xrightarrow[OS]{\theta} M' \iff \text{WN}(M) \xrightarrow[\text{WN}(OS)]{\theta} \text{WN}(M')$$

Proof. (Sketch) Using this syntactical restrictions we enforce that in each closed transactions $w = \alpha_1(x_1.t_1) \cdots \alpha_n(x_n.t_n)$ each transition can appear at most once, since $F \cap ((Z \times T) \cup (T \times Z))$ is an acyclic relation. Since expressions are either disjoint or identical and all variables are bound in the preset, each transactions $w = \alpha_1(x_1.t_1) \cdots \alpha_n(x_n.t_n)$ can be denoted as one with a single variable assignment α: $w = \alpha(x_1.t_1 \cdots x_n.t_n)$. Note that the guard G of $x_1.t_1 \cdots x_n.t_n$ is the conjunction $G(t_1) \wedge \ldots \wedge G(t_n)$. So, $\Theta(OS)$ is a finite set. It is easy to see, that by definition of WN(OS) the pre- and post- conditions coincide. □

5 A Case Study: The Bucket Chain

In order to illustrate the simulation of a HORNET by a well-formed net we present the transformation of the bucket chain example. This is done step by step. First the *bucket* and the *fireman* will be fused into one net. Then the new net and the bucket chain net will be fused to form the resulting net. The result is the simulation of the earlier presented bucket chain example.

Figure 4 sketches the fusion of the bucket with the fireman. The fusion[6] of the two pairs of transitions that form the synchronous channels are highlighted in the image. The result of the first step – the fusion of the fireman net with the bucket net – is presented in Figure 5. Here the identity for the bucket net is included in the net. However, since every fireman owns exactly one bucket, only one identity is needed. Furthermore, the fusions of the synchronous channels are completed. The unused synchronous channels for status checks (*isEmpty()*, *isFull()*) are discarded for the reason of simplicity.

The complete result of the fusion of the bucket chain example, i.e. the simulating WN is shown in Fig. 6. In this net, the identities reserved for the firemen and together with these also the identities for each fireman's bucket are added. Thus, three identities for the firemen and three identities for the buckets are used in the simulation. This can be seen in the upper left corner.

[6] Note that "fusion" – in this context – means that the two transitions are actually replaced by one single transition.

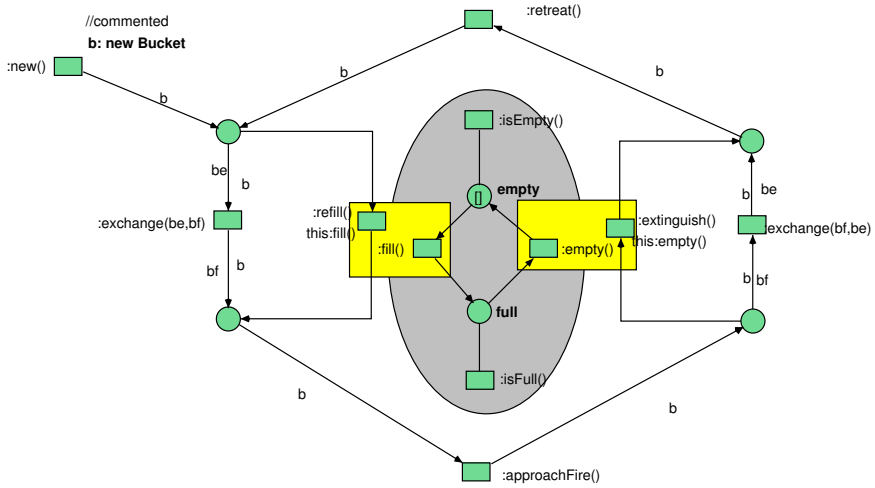


Fig. 4. Sketch of the Fusion of Fireman Net and Bucket Net

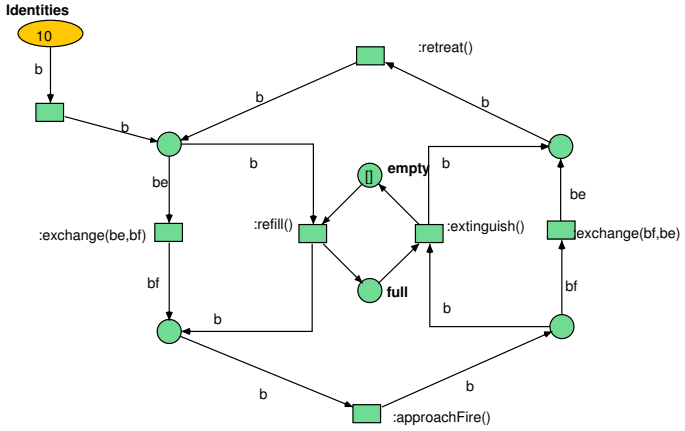


Fig. 5. Fusion of the Fireman Net and the Bucket Net

We have tried to keep the layouts of the reference nets by locating the nets next to each other (similar to the fusion of fireman and bucket) so that the layout of the old nets can still be identified in the simulating net. However, due to the fact that transitions in different areas of the net have to be fused this is not always possible. Also the duplication of transitions that results from multiple *downlinks* can not be avoided and leads to a messed up net layout. Nevertheless, the fragments (only the places) of the bucket chain can be seen at the left side of the net.

The fusion of the synchronous channels, i.e. the replacing of the original transitions by a single transition, have been applied in the location of the *uplinks*, i.e. in the location of the former fireman net. However, as mentioned before, these transitions have to be duplicated (*aproachFire()* and *retreat()*). This results in three transitions for each fused synchronous channel due to the fact that in the bucket chain three *downlinks* for each *uplink* are used.

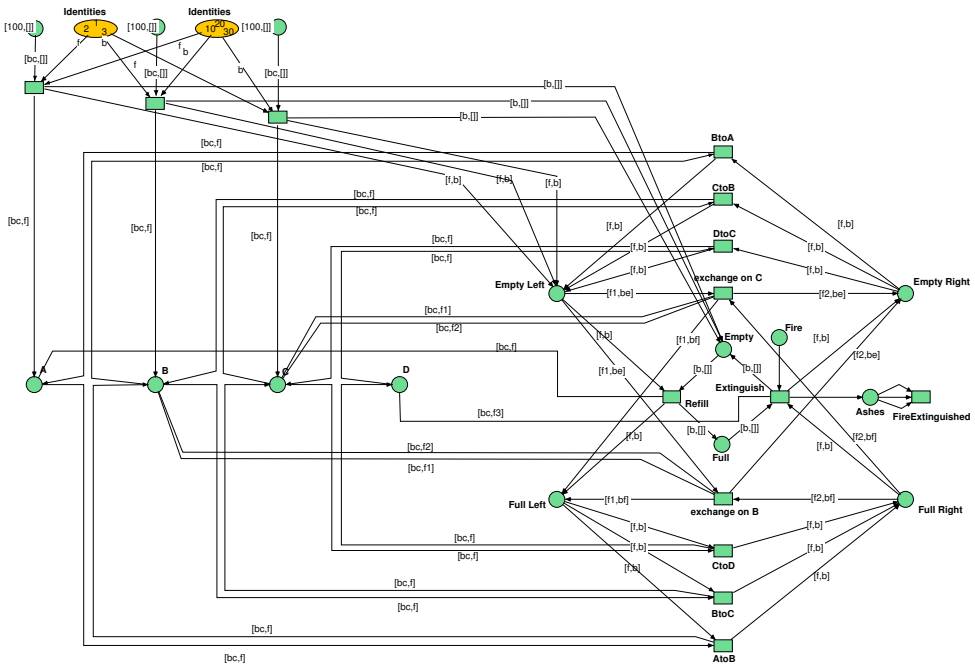


Fig. 6. The simulating WN

An important fact is that tuples of firemen and buckets (e.g. *[f,b]*) are used to identify the firemen and to imply ownership of buckets. Also the usage of the different applications of the *downlinks* is encoded into the fireman-bucket pairs.

The resulting simulating net has a rather nasty layout. However, this is not surprising, since the folding of the net into reference nets, i.e. in an object-based manner, adds to the clarity of the model by reducing the complexity. In addition, it has to be mentioned that the layout results from the approach that has been chosen, which can be performed automatically by a machine.

Nevertheless, this example shows that HORNETS can be effectively transformed into well-formed nets.

6 Conclusion

In this presentation we have presented the formalism of "Higher Order Reference Nets" (HORNETS), which is a higher order variant of the object net system of Valk, very similar to reference nets of Kummer.

From a formal point of view the reason to introduce the formalism of HORNETS is to restrict reference nets in such a way that the symmetries in the system can be exploited for analysis purposes. For the class of well-formed HORNETS these symmetries can be deduced automatically from the net structure. Here our notion of well-formedness follows the notion of well-formedness in coloured Petri nets. It has been shown that each well-formed HORNET can be simulated by a well-formed Petri net in a natural way.

From a modelling point of view HORNETS offer a very expressive modelling formalism. Especially for dynamic systems, e.g. for mobile agent systems, the different layers of the system (e.g. the bucket chain, the fireman, and the bucket) can be modelled separately leading to more compact and modular models – cf. the HORNET system in Fig. 1 to 3 compared to the simulating WN in Fig. 6.

Forthcoming work of our group is to integrate the HORNET formalism as a simulation mode into our tool RENEW and to integrate the tools for well-formed nets to analyse these nets, e.g. for calculating symbolic invariants or symbolic reachability graphs.

References

[BG91] Didier Buchs and Nicolas Guelfi. CO-OPN: A concurrent object oriented Petri net approach. In *International Conference on Application and Theory of Petri Nets*, pages 432–454. Springer-Verlag, 1991.

[BM97] Roberto Bruni and Ugo Montanari. Zero-safe nets – or transition synchronization made simple. In *Electronic Notes in Theoretical Computer Science: Proceedings of EXPRESS'97, 4th workshop on Expressiveness in Concurrency*, volume 7, pages 1–19. Elsevier Science, 1997.

[BM00] Roberto Bruni and Ugo Montanari. Executing transactions in zero-safe nets. In M. Nielsen and D. Simpson, editors, *Conference on Application and Theory of Petri Nets (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 83–102. Springer-Verlag, 2000.

[CDFH90] Giovanni Chiola, Claude Dutheillet, Guiliana Franceschinis, and Serge Haddad. On well-formed coloured nets and their symbolic reachability graph. In G. Rozenberg, editor, *Proceedings of the 11th International Conference on Application and Theory of Petri Nets*, volume 524 of *Lecture Notes in Computer Science*, pages 387–410. Springer-Verlag, 1990.

[Jen92] Kurt Jensen. *Coloured Petri nets, Basic Methods, Analysis Methods and Practical Use*, volume 1 of *EATCS monographs on theoretical computer science*. Springer-Verlag, 1992.

[KF04] Michael Köhler and Berndt Farwer. Processes of zero-safe nets. In *Proceedings of the International Workshop on Concurrency, Specification, and Programming, CS&P 2004*, 2004.

[KMR03] Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling mobility and mobile agents using nets within nets. In W. v. d. Aalst and E. Best, editors, *International*

Conference on Application and Theory of Petri Nets 2003, volume 2679 of *Lecture Notes in Computer Science*, pages 121–140. Springer-Verlag, 2003.

[KR04] Michael Köhler and Heiko Rölke. Properties of object Petri nets. In J. Cortadella and W. Reisig, editors, *International Conference on Application and Theory of Petri Nets 2004*, Lecture Notes in Computer Science, pages 278–297. Springer-Verlag, 2004.

[Kum00] Olaf Kummer. Undecidability in object-oriented Petri nets. *Petri Net Newsletter*, 59:18–23, 2000.

[Kum02] Olaf Kummer. *Referenznetze*. Logos Verlag, 2002.

[KWD+04] Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for Petri nets: Renew. In J. Cortadella and W. Reisig, editors, *International Conference on Application and Theory of Petri Nets 2004*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[Lak95] Charles Lakos. From coloured Petri Nets to Object Petri Nets. In *Proceeding of the 16th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, pages 278–297, Berlin, 1995. Springer-Verlag.

[MM97] Christoph Maier and Daniel Moldt. OCPN - a formal Technique for OO Modelling. In Berndt Farwer, Daniel Moldt, and Mark-Oliver Stehr, editors, *Petri Nets in System Engineering (PNSE'97): Modelling, Verification and Validation*, September 1997.

[Pet79] Carl Adam Petri. Introduction to general net theory. In W. Brauer, editor, *Net Theory and its applications. Proceedings of the Advanced course on general net theory of processes and systems*, volume 84 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[PS85] James L. Peterson and Abraham Silberschatz. *Operating System Concepts*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1985. Second edition.

[Val98] Rüdiger Valk. Petri nets as token objects: An introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25, 1998.

[Val03] Rüdiger Valk. Object Petri nets: using the nets-within-nets paradigm. In *Advanced Course on Petri Nets 2003*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

A Unidirectional Transition Fusion for Coloured Petri Nets and its Implementation for the CPNTools

João Paulo Barros*[1,2] and Luís Gomes[1]

[1] Universidade Nova de Lisboa / UNINOVA, Portugal
[2] Instituto Politécnico de Beja - ESTIG, Portugal
{jpb, lugo}@uninova.pt

Abstract. Petri nets have been frequently extended with object-oriented concepts. This has originated several flavours of object-oriented nets, often with no clear connection to the common Petri net semantics. This paper informally presents a unidirectional transition fusion for coloured Petri nets named *synchrony groups*, together with a filter tool, for the CPNTools, allowing the translation from hierarchical coloured Petri nets with synchrony groups to hierarchical coloured Petri nets without synchrony groups. Synchrony groups are used in a context where each class is modelled as a hierarchical coloured Petri net page allowing the modelling of synchronous requests between objects.
Keywords: Coloured Petri Nets, CPNTools, object-oriented design, synchronous requests, transition fusion, net composition.

1 Introduction

After Place/Transition nets, coloured Petri nets (CPNs) [1] are probably the best well-known Petri net class. Besides the known advantages CPNs offer, their popularity is also due to the availability of powerful tools, especially the Design-CPN tool [2], which has recently been superseded by CPNTools [3].

Coloured Petri nets are particularly interesting due to their high-level concepts, which allow the specification of compact models. The parallelism between Coloured Petri Nets versus Place/Transition nets and high-level programming languages versus assembly language is often used as an indicator of the extreme modelling convenience offered by CPNs when compared to low-level nets. Therefore, it has been quite natural to follow this analogy a step further, and try to incorporate object-oriented concepts into Petri nets, often taking CPNs as the starting point. This has given origin to numerous proposals for "object-oriented Petri nets" (see [4] for a comprehensive survey). When compared to CPNs, these proposals include additional syntax and semantics that easily lead them quite further away from CPNs and, consequentially, from well-known Petri nets syntax and semantics. Although some proposals do include translations to equivalent

* Work partially supported by a PRODEP III grant (Concurso 2/5.3/ PRODEP/2001, ref. 188.011/01).

Coloured Petri nets (e.g. [5] and [6]) they clearly emphasise the need for new Petri net classes.

In [7] we proposed a distinct attitude towards the introduction of object-oriented concepts in Petri nets: the use of two syntactically minimal abbreviations to coloured Petri nets, together with a set of design idioms. The two abbreviations were:

1. The use of fusion places for modelling asynchronous requests;
2. A unidirectional transition fusion for modelling synchronous requests named **synchrony groups**.

The first abbreviation, is also used in hierarchical coloured Petri nets (HCPNs) [1]. This paper presents a method allowing the use of the second abbreviation in the CPNTools. It also presents a filter tool, which implements the method. In particular, the filter tool allows the translations from HCPNs, extended with synchrony groups, to equivalent HCPNs without synchrony groups. As the CPNTools supports HCPNs and, consequentially, fusion places, that capability was extensively used. This allowed the generated model to stay closer to the initial model (with synchrony groups) than what would be possible if a single CPN page had to generated.

The following section presents the motivation for the use of synchrony groups. After, we informally present synchrony groups for Coloured Petri Nets. These were formally defined in [7] as part of the class of **Composable Coloured Petri nets**. Next, we present an illustrative example showing the use of simple synchrony groups and also the modelling of a polymorphic request. Finally, we present a filter tool for the transformation of HCPNs with synchrony groups into HCPNs and conclude with some pointers for future enhancements to it.

2 Motivation

The fundamental motivations for the creation of synchrony groups were already documented elsewhere [7] as a part of the Composable Coloured Petri nets class. This class of high-level Petri nets has the following objectives:

- To close the gap between CPNs and object-oriented design.
- To be strongly based on CPNs.
- To avoid modifications to the CPNs graphical notation and minimal additions to the textual notations.
- To use a composition operator, for the specification of generalisation and composition, and a different abstraction for message passing modelling.
- To be reducible to a CPN.

Note that we want of avoid significant differences to CPNs. Also, we want to stay close to common object-oriented design concepts. In particular, we want to remain close to synchronous message passing, which is typically unidirectional (e.g. a method call). Synchrony groups are the proposed way to model

synchronous message passing. Place fusion is used for asynchronous message passing.

Although the motivation was the support for object-oriented design, the syntactical and semantic proximity between method calling and synchrony groups also means they can be used outside of a object-oriented framework. In particular, they can be used as a convenient way to integrate synchronous communication in HCPN models.

3 Synchrony groups

Synchrony groups were inspired by coloured communication channels [8] but differ in the implicit direction imposed by the parameters' qualifiers. This section presents an example-based informal overview of synchrony groups followed by a specific proposal for their effective use in the CPNTools.

3.1 Syntax and Semantics

A synchrony group is specified by a request with associated *SEND* and *RECV* declarations (see Fig. 1 also found in [7]). These latter two are tuples associated to different transitions. A transition with an associated SEND declaration is called a **send transition**. Accordingly, a transition with an associated RECV declaration is called a **receive transition**. Each transition can have zero or one associated event.

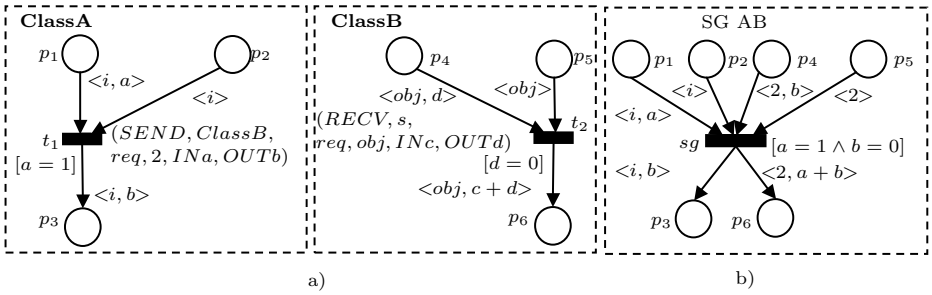


Fig. 1. a)Synchronous request with parameters, and b) The respective request transition.

When using synchrony groups, the HCPNs pages are typically seen as classes of an object system. The several objects are represented by tokens flowing around the page. Subpages (and the associated substitution transitions) continue to be seen and used as a graphical convenience allowing model modularisation.

The SEND and RECV declarations have the respective following formats:

$$(SEND, targetClass, request, [qualifier_1]parameter_1, \ldots, [qualifier_n]parameter_n)$$

$(RECV, senderClass, request, [qualifier_1]parameter_1, \ldots, [qualifier_n]parameter_n)$

Their elements are presented next:

SEND* and *RECV The *SEND* and *RECV* elements specify the request direction.

targetClass The *targetClass* element can have one of two distinct meanings: (1) the class to which a receive transition with the same request belongs; (2) a variable which can take as values, names of classes containing one receive transition for the specified request. The latter allows dynamic binding instantiation.

request The *request* element identifies the request. Formally, each synchrony group is a triple with a send transition, a recv transition, and a request.

parameter* and *qualifier The *parameters* are transition variables, and the *qualifiers* can take one the following values "*IN*", "*OUT*", or "*INOUT*". These names are given from the send transition point of view. A qualifier can be omitted. In that case, it defaults to an *IN* parameter.

senderClass The *senderClass* is the name of the send transition's page. This allows its use as a transition variable allowing the specification of a "return" to the sender class. In fact, it can be seen as a permanent *IN* parameter. This allows the modelling of a typical method call through the use of two synchrony groups: one to invoke the method; another to return to the caller.

An "*IN*" parameter must be bound by the send transition. Typically it will be used in one or more of the receive transition output arcs: the receive transition "does something" with the received input parameter.

An "*OUT*" parameter must be bound by the receive transition. Typically, it will be used in one or more of the send transition output arcs: the send transition "does something" with the received output parameter.

An "*INOUT*" parameter must be bound by the send and receive transitions. Typically, it will be used in one or more of the send transition output arcs and in one or more of the receive transition output arcs.

Parameter passing is made "by name": the receive transition parameters are textually replaced by the respective send transition parameters.

Fig. 1a exemplifies a synchronous request and the corresponding translation (a synchrony group) to a CPN (Fig. 1b).

The variables "*a*" and "*b*" replace, respectively, the variables "*c*" and "*d*" "by name": "*a*" and "*b*" are the request actual parameters, "*c*" and "*d*" are the request formal parameter. Note that the object identifier ("2" in the example) is also a request parameter: by default, it is an "*IN*" parameter . The *targetClass* identifier (*ClassB* in Fig. 1) is used to specify the net (class) where the corresponding RECV transition resides. On the RECV declaration, the *senderClass* element identifies the class where the respective SEND declaration resides ("*s*" parameter in the example).

The Renew tool [9, 10] also implements a form of synchronous channels. It differs from our proposal in two ways: (1) the Renew nets are *object nets* where

the object (or class instance) is a token; instead we rely on CPNs tokens; (2) The channel parameters in Renew have no qualifiers.

3.2 Synchrony groups and method calling

Synchrony groups fulfil the double objective of being close to Petri net semantics and allowing a unidirectional invocation with parameter passing, as usually found in object-oriented design and programming languages, most notably UML [11]. The closeness to Petri nets implies that a method call must be modelled as a pair of synchrony groups: one for the invocation, and another for the return. This is illustrated in Fig. 2 for a object method call and a class method call. In the latter, notice the absence of an object identifier parameter and the use of a place (CLA) for class level attributes. Outside of an object-oriented framework, the class method call can be seen as procedure call to a procedure in a module named *classD*.

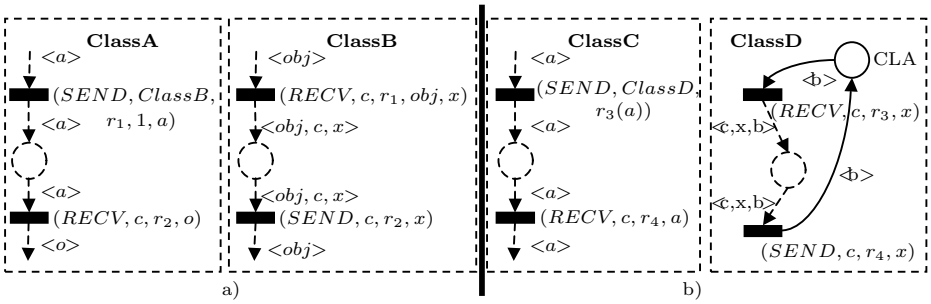


Fig. 2. a)Double synchronisation for non-atomic operation invocation. b)Double synchronisation for class level operation invocation; notice the use of the class level attributes place (CLA) accessed by two transitions with associated class level requests, without instance identifier in the request.

3.3 Synchrony groups for the CPNTools

As already exemplified, the semantics of HCPNs, with synchrony groups, in terms of the "regular" HCPNs semantics is quite straightforward. Hence, an extension to the CPNTools allowing its execution is certainly possible and probably simple for a CPNTools developer. Yet, currently the CPNTools documentation is user-oriented: there is no programmers' documentation allowing the development of tool extensions. Due to this situation, we have chosen another option: the implementation of a filter tool capable of translating HCPNs with synchrony groups to "plain" HCPNs. Basically, the tool implements the kind of transformation exemplified in Fig. 1.

The SEND and RECV specification To allow the specification of the transitions' associated SEND and RECV declarations, we decided to take advantage of an already existing transition related annotation: the code segment. The user specifies the SEND and RECV declarations as Standard ML comments inside the code segment. To distinguish this Standard ML comments from regular Standard ML comments we use a distinct "parenthesis". Instead of "(∗" and "∗)" used for Standard ML comments, the SEND and RECV declarations are written inside "(∗∗" and "∗∗)". Note that, code segments are not allowed in receive transitions. This avoids the problem of fusing code segments. while maintaining the possibility of a different code segment for each invocation: the one in the send transition.

The following section presents a slight variation of the example introduced in [7] but with all the necessary details to make it a legal CPNTools model. It specifies several SEND and RECV declarations inside the code segments.

The synchrony groups translation Synchrony groups imply the destruction of the two transitions involved (*send* and *receive*), and the creation of a new single transition. This new transition corresponds to the *synchrony group* in Fig. 1 and it is named **request transition**. It connects to all the arcs previously connected to the *send* and the *receive* transitions. The parameters in the *RECV* declaration must be textually replaced by the corresponding ones in the respective *SEND* declaration. In addition, the request transition guard is the conjunction of the send transition guard and the receive transition guard, after the parameters textual substitution. To guarantee a minimal modification to the original HCPN graphical layout, one can create a new page (a **request page**) containing the newly created transition (the request transition) and making the send transition a substitution transition for this new page. The places connected to the send transition become socket places and the corresponding port places are then created in the request page.

Additionally, the receive transition is deleted and all the places connected to it become fusion places. For each of these fusion places a copy is created in the request page. These place copies are then connected to the request transition.

Next, we exemplify this approach. Figs. 3 and 4 show two pages, each one modelling a class of an initial model.

Notice that transition *t1* in *classA* sends a request to transition *t4* in *classB*. Additionally, transition *t6* in *classB* sends a request to transition *t3* in *classA*.

The places *check1* and *check2* only serve to avoid syntax errors due to the use of transition variables in output arc expressions that are not present in input arc expressions (are not bound by the respective transition). This happens with variables that are *OUT* parameters in SEND declarations (e.g. variable *b* in *classA*) or *IN* parameters in RECV declarations (e.g. variable *y* in *classA* and variable *c* in *classB*). Due to the mandatory parameters' occurrence in the send or receive transition input arcs (listed in section 3.1), these errors cannot be present in the final model (after transformation). For this reason, the check

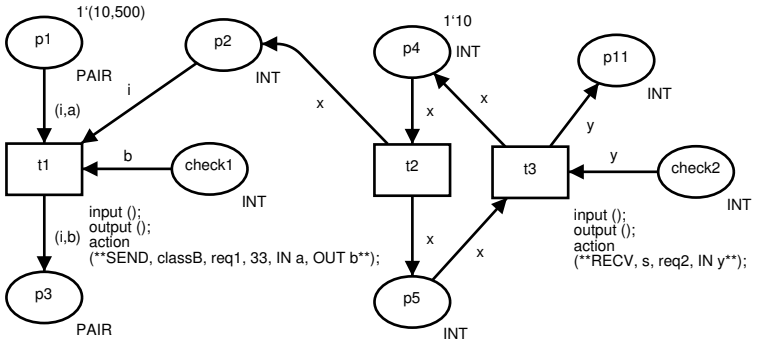


Fig. 3. ClassA page.

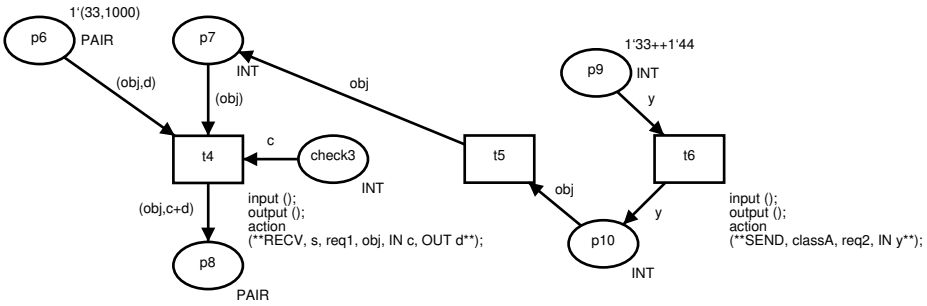


Fig. 4. ClassB page.

places are totally optional: if present, they are removed in the final model[3]. Yet, we believe they are useful in the present state of CPNTools as they allow the modeler to take advantage of the tool syntax checking: the tool immediately alerts the modeler if some parameter cannot be bound. Finally, note that these check places do not need any associated marking: they only need a colour and the associated arcs, each one with an associated variable.

Figs. 5 and 6 show the same classes after transformation. Note that the send transitions (*t1* and *t6*) remain as substitution transitions associated with the respective request classes. The receive transitions (*t3* and *t4*) were removed and their input and output places were made fusion places. Each one is fused with the respective copy in the associated request page (Figs. 7 and 8).

Unfortunately, CPNTools does not allow a port/socket place to be a fusion place. Therefore, the only option seems to be the translation of the *send transition* in exactly the same way as described for the *receive transition*: the removal of the *send transition* and the creation of places copies in the request page to-

[3] The tool (see section 5) identifies the places to be removed by looking at the respective names: the places with a name beginning with "check" are removed.

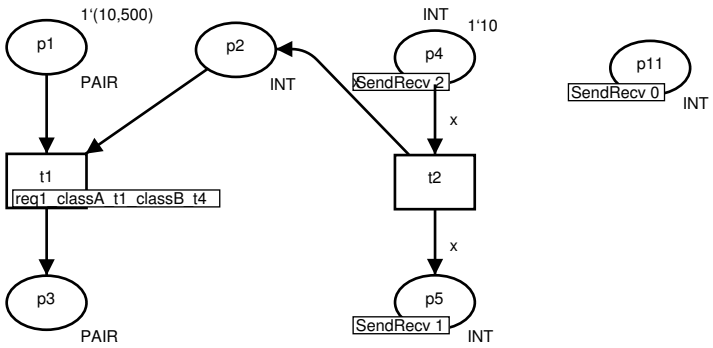


Fig. 5. ClassA page after transformation.

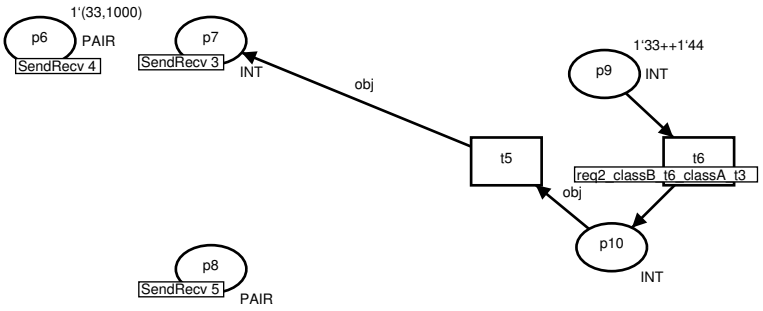


Fig. 6. ClassB page after transformation.

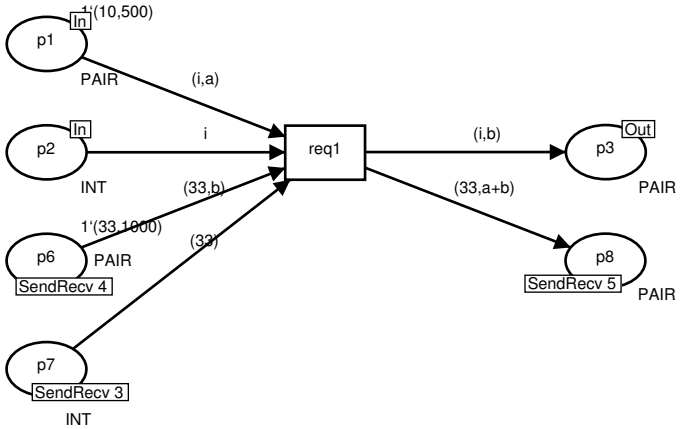


Fig. 7. Request *req1_classA_t1_classB_t4* page.

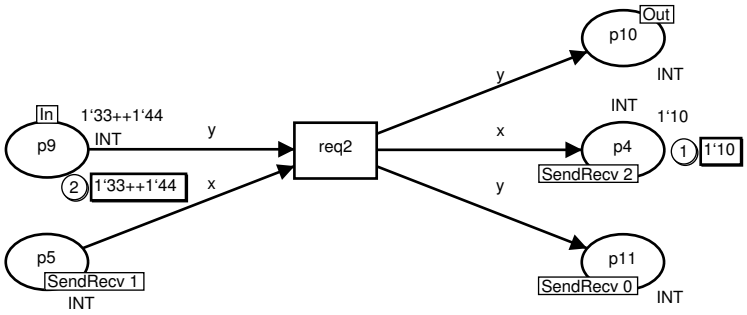


Fig. 8. Request *req2_classB_t6_classA_t3* page.

gether with their fusion with the respective places in the original *send transition*. This is the approach implemented in the filter tool presented in section 5 and reflected in the example to be presented in the following section.

Synchrony groups integration in the CPNTools The transformational approach implemented by the presented filter tool, allowing the use of synchrony groups in HCPNs models, is surely not the ideal one. There are three basic reasons for this:

1. The tool is not integrated in the CPNTools; the user has to run the tool outside the CPNTools and then reload the transformed model;
2. Most syntax checking related to the synchrony groups is done by the filter tool;
3. Notwithstanding the creation of new pages to avoid significant graphical layout modifications between the initial and the transformed model, this is still quite distinct from the initial one; this hampers model readability.

A preferable solution would be the integration of the synchrony group construction inside CPNTools. This would avoid the generation of transformed models: the simulation would be carried out with the initial model.

We end this section by proposing a list of features that should be implemented in the CPNTools to integrate and fully support synchrony groups:

- The semantics should be the same as the one exemplified by the transformed models in this paper – the actual and formal parameters are considered equal before firing the respective transitions now fused as one;
- Each transition should admit an extra inscription (separated from code segments) for the specification of a SEND or RECV request;
- SEND requests with no corresponding RECV requests should signalled as syntax errors;
- The CPNTools left tab should show the receive transitions and the several associated send transitions; one mouse click should take the modeller to the page (class) containing the respective send or receive transition;

– Non matching parameter types or qualifiers between send and receive transition pairs, should be signalled as syntax errors;
– The mandatory occurrences of parameters in input arcs should be automatically checked.
– IN parameters in SEND requests and OUT parameters in RECV requests should be considered as possible ways to bind transition variables; this would make check places useless;
– The use of a variable with class names as variables, should be supported; this allows polymorphic invocations as exemplified in the example presented in the following section.

4 An example

This section details part of the example system model in [7].

4.1 The Specification with Synchrony Groups

The model consists of the following classes:

– *Writer*
– *Producer*
– *DB*
– *Reader*
– *ConsumerA*
– *ConsumerB*

The *Writer* object (see Fig. 9) asks the *Producer* (see Fig. 10) for data(request *reqGet* in transition *get*). The *Producer* object chooses one of two types of consumer (*ConsumerA* or *ConsumerB*) and passes the data to the *Writer* object, This tries to write the data and the consumer type to the *DB* object, as long as the *DB control* place allows it.

All classes are modelled in their own page. We used a specific token to model each Writer object, although the model uses only one writer (the *writer(1)*). This approach allows a straightforward extension to more than one Writer object. On the contrary, for the Producer class, and for illustration purposes, we decided not to model the Producer objects. This attitude means the Producer class is always seen as a singleton object. In other words, our model should never have more than one Producer.

Transition *produce* in class *Producer* generates the data to be consumed. For simplification purposes, it always generates the data string *"some data"*. Obviously, it could generate any other data depending on some arbitrarily complex computation. It is also the *Producer* singleton that specifies which consumer type should, in the end, get the data. Again, for simplification purposes, the consumer type is arbitrarily chosen from the set of consumer types specified in place *consumers*.

The *Writer* class uses two *check places* (*check1* and *check2*). As already stated, their only purpose is to avoid syntax errors when modelling the class. For example, the variable *consumer* that in reality is obtained as an OUT parameter from request *reqGet* does not cause a syntax error due to the presence of place *check2* and respective arcs. As already noted, all check places and the respective arcs are removed from the final model. Finally, notice that the writer object needs, at least, a number of tokens in *control* place equal to the number of readers (*NREADERS*).

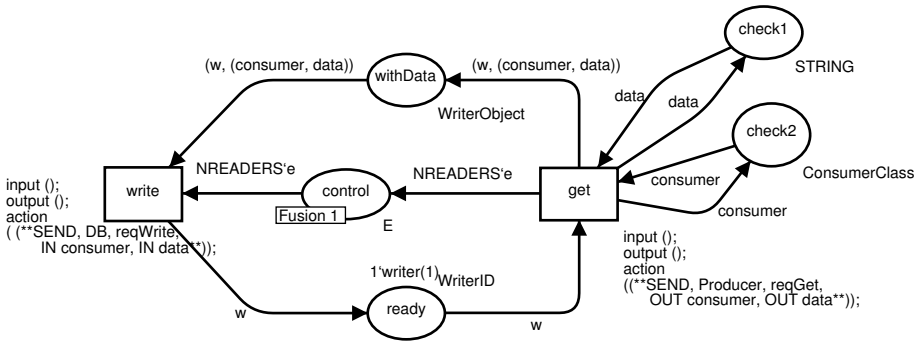


Fig. 9. *Writer* class page

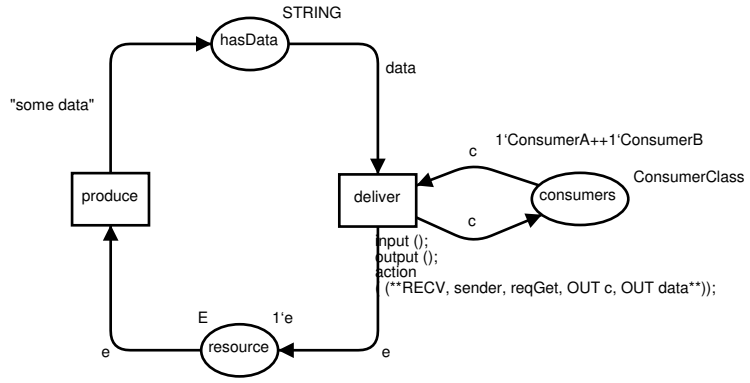


Fig. 10. *Producer* class page

The two *Reader* objects (*reader(1)* and *reader(2)* in Fig. 11) try to read data from the *DB* object, as long as the *DB control* place allows it. This is achieved by the request *reqRead* with a SEND declaration associated to transition *do_read*. The *reqRead* request has two OUT parameters: *consumer* and *data*. The latter

is passed to the consumer type specified by the former. This means either the *ConsumerA* object or the *ConsumerB* object. This is achieved by transition *send*, which has a SEND declaration for request *reqPut*. The variable *consumer* is dynamically bound. Just like the *Writer* object, the *Reader* object also needs two check places to avoid syntax errors due to variables *consumer* and *data*. Differently from the Writer object, they only need one control token in fusion place control, to be able to read from the database object *DB*. Naturally, they also need available data in the object DB (Fig. 12). That is guaranteed by the receive transition *get* in *DB*. Notice that the fusion place *control* also appears on the *Writer* (Fig. 9) and *DB* (Fig. 12) classes. Although it has no arcs in class *DB*, the *control* place "belongs" to class *DB* and it is used by the *writer* and *consumer* objects.

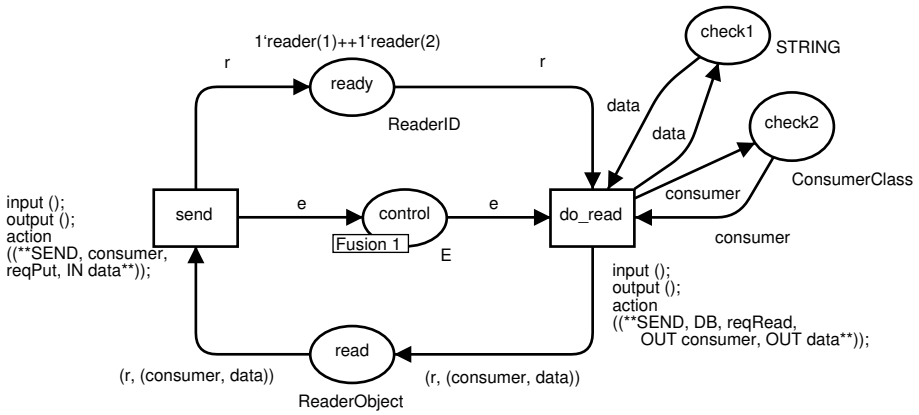


Fig. 11. *Reader* class page

The *ConsumerA* and *ConsumerB* classes are presented in Fig. 13 and Fig. 14, respectively. For simplification purposes, both were defined as singletons. Both have request *reqPut* in common. These requests are called, after dynamic instantiation of variable *consumer* in class *Reader*. More specifically, transition *send* in class *Reader* "calls" either transition *retrieveA*, in class *ConsumerA*, or transition *retrieveB*, in class *ConsumerB*, depending on the value of the *consumer* variable being *ConsumerA* or *ConsumerB*, respectively.

4.2 The Equivalent HCPN Specification

Here we present the equivalent HCPN model. It is semantically equivalent to the initial specification (with synchrony groups) just presented, but with one important difference: it is a legal CPNTools model automatically generated from the initial specification.

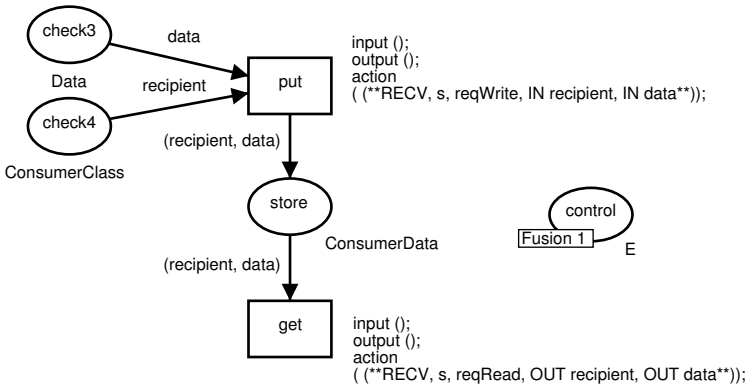


Fig. 12. *DB* class page

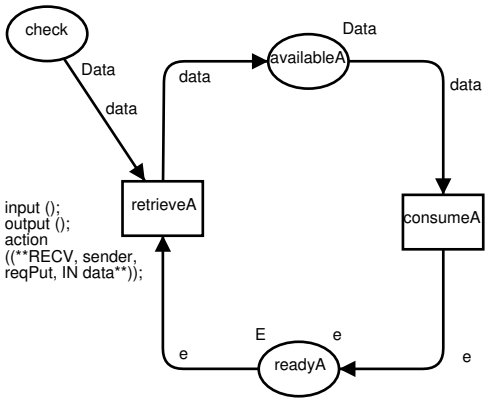


Fig. 13. *ConsumerA* class page

First, we present the initial classes. After, we present the transformed initial classes and the newly created request pages.

Initial pages Regarding the initial pages, the send and receive transitions are removed, together with all the respective arcs. With the exception of the places previously connected to the removed transitions (which become fusion places), the remaining elements are not changed, not even their layout. This allows an easy recognition when comparing with the initial specification containing the send and receive transitions. To increase this recognition, the filter tool replaces each send and receive transition with a dotted box containing the name of the respective removed transition. This makes the resulting model graphically closer to the initial one. Unfortunately, as the CPNTools does not allow it, we could

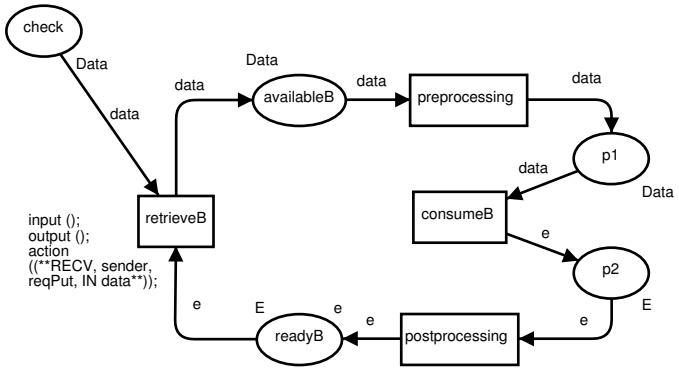


Fig. 14. *ConsumerB* class page

not draw fake arcs, which would increase the graphical similarity to the initial model. Figs. 15, 16, 17, 18, 19, and 20 show the transformed initial classes.

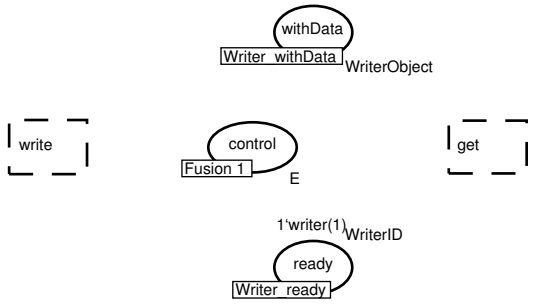


Fig. 15. Final *Writer* class page. Compare with the initial specification in Fig. 9.

Request Pages For each synchrony group, one new transition is created. This new transition is made part of a new page. As already stated, these new pages are named request pages. Figs. 21, 22, 23, 24, and 25 present all the request pages automatically generated from the initial classes.

Each request page gets a copy of each place connected to the send or the receive transitions that gave origin to the request transition. Being a transition fusion, these places are connected to the request transition in exactly the same way as the original places were connected to the respective send and receive transitions.

To avoid graphical superpositions between the places and arcs from the send and receive transitions, they cannot keep the same graphical layout in the request page. Hence, we decided to distribute the places connected to the request

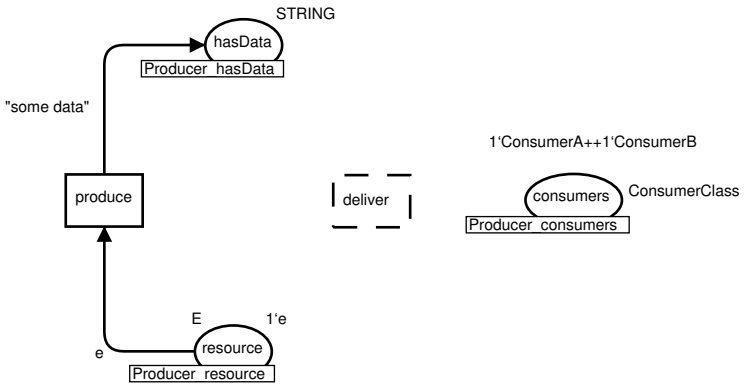


Fig. 16. Final *Producer* class page. Compare with the initial specification in Fig. 10.

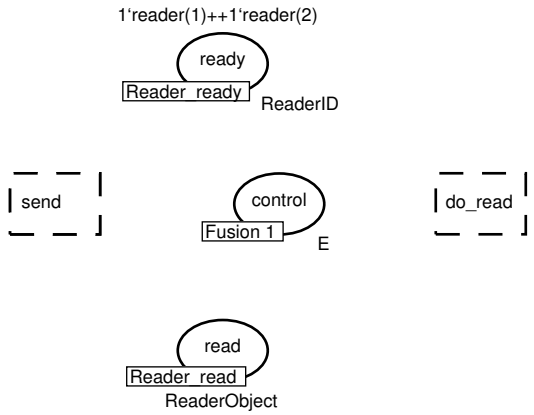


Fig. 17. Final *Reader* page class. Compare with the initial specification in Fig. 11.

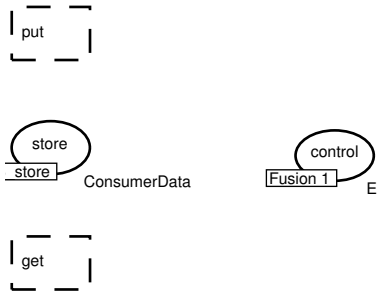


Fig. 18. Final *DB* page class. Compare with the initial specification in Fig. 12.

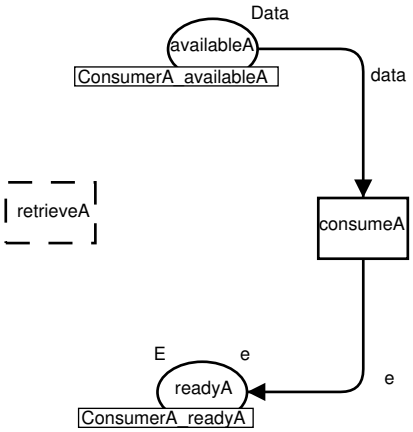


Fig. 19. Final *ConsumerA* class page. Compare with the initial specification in Fig. 13.

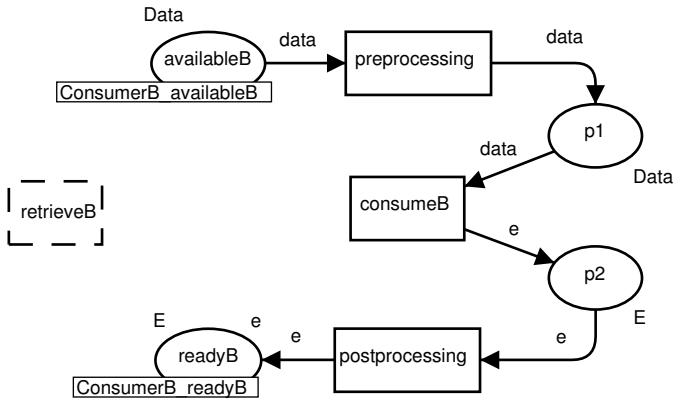


Fig. 20. Final *ConsumerB* class page. Compare with the initial specification in Fig. 14.

transition along a vertical line on the left (for transitions' input places) and a vertical line on the right (for transitions' output places). An input-output place is not duplicated (by place fusion) but instead put in one of the vertical lines. This happens with place *Producer_consumers* in the *ReqGet* request page (Fig. 23)[4].

The polymorphic invocation specified by transition *send* in class *Reader* (see Fig. 11) gives origin to one request page for each possible value of the dynami-

[4] The two arcs are superposed and have the same annotation (*consumer*). Future versions of the filter tool should avoid this superposition.

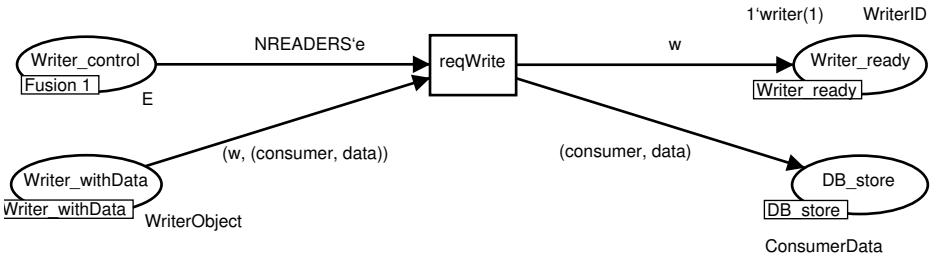


Fig. 21. ReqWrite page

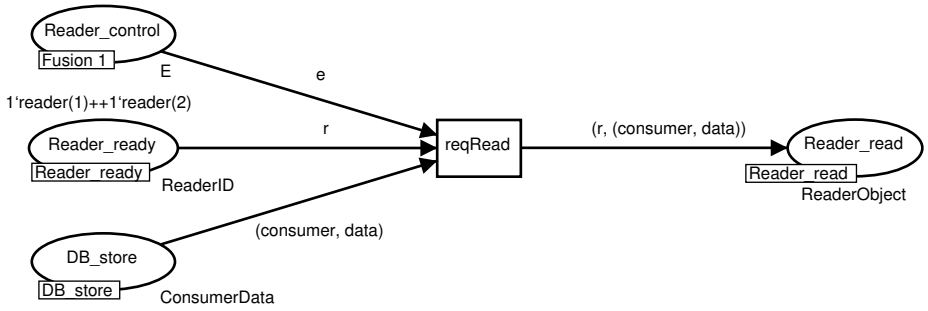


Fig. 22. ReqRead page

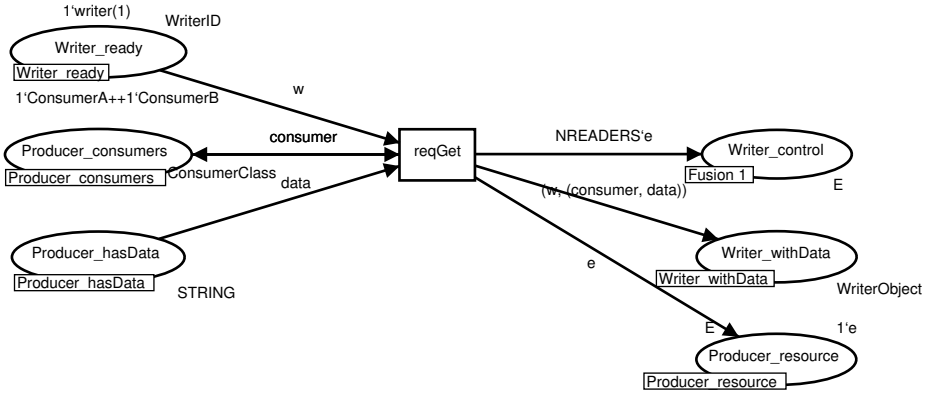


Fig. 23. ReqGet page

cally bounded variable. In our example, we have two different values for variable *consumer*. This variable specifies which class should be called for request "ReqPut". This translates to two request pages: one when the *consumer* variable has the value "ConsumerA" (see Fig. 24) and another when the *consumer* vari-

able has the value "ConsumerB" (see Fig. 25). This distinction is made by the transitions' guards.

Note that the object identifier can be specified as a simple request parameter but this is not needed here as both classes have only a singleton object.

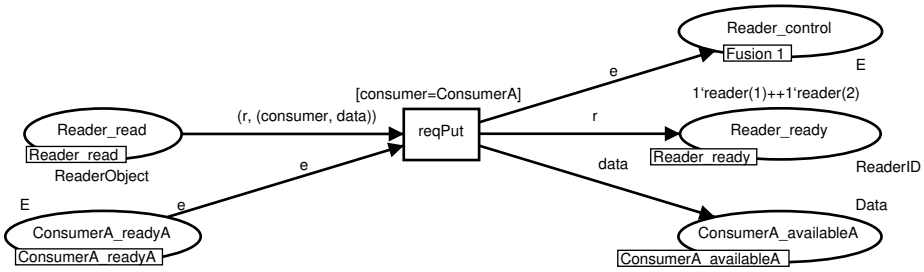


Fig. 24. ReqPut page for ConsumerA.

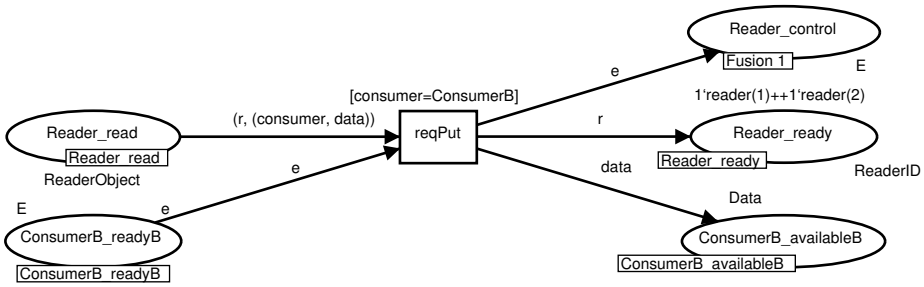


Fig. 25. ReqPut page for ConsumerB.

5 The Filter Tool

As already mentioned, the synchrony groups extension (more specifically the SEND and RECV declarations) can be used in the CPNTools through their specification as comments inside code segments. This CPNTools models are then translated to CPNTools models where a new page (the request page) is created for each SEND and RECV pair. The original send and receive transitions are removed from the original class pages where they appeared.

As a proof of concept we implemented a filter tool that is able to realise this translation: it takes a CPNTools model (a .cpn file) and generates another CPNTools model (another .cpn file).

This filter tool, named ccpn2hcpn, is written in Ruby [12], an object-oriented scripting language.

The translation for the example model in the previous section was created using the ccpn2hcpn tool. The tool's current version, together with a set of example models, is available at the associated web page [13].

6 Conclusions and future work

The use of send and receive transitions has yet to be tested in realistic size models. Even so, we believe its usage adds significant modelling convenience to hierarchical coloured Petri nets.

It is also clear that the translation approach used by the ccpn2hcpn tool is only useful due to the lack of direct support by CPNTools. If this becomes available, the translation tool will no longer be needed. Meanwhile the translation approach is useful and also serves as a proof-of-concept and a test-bed for additional enhancements. In this sense, we intend to extend it in the following directions:

- Stronger syntax verification;
- Support for multiple invocation by allowing multiple SEND declarations in one send transition;
- Support for net addition as proposed in [7] for the modelling of a simple inheritance type in CPNs.

References

1. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use - Volumes 1–3. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, Germany (1992–1997)
2. Design/CPN: Design/CPN homepage. `http://www.daimi.au.dk/designCPN/` (2004)
3. CPN Tools: CPN Tools homepage. `http://wiki.daimi.au.dk/cpntools` (2004)
4. Agha, G., de Cindio, F., Rozenberg, G., eds.: Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets. Volume 2001 of Lecture Notes in Computer Science. Springer (2001)
5. Lakos, C.A.: From coloured Petri nets to object Petri nets. In: Proceedings of the 16$^{\text{th}}$ International Conference on Application and Theory of Petri Nets, Turin. (1995) 278–297
6. Maier, C., Moldt, D.: Object coloured Petri nets – a formal technique for object oriented modelling. Lecture Notes in Computer Science: Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets **2001** (2001) 406–427
7. Barros, J., Gomes, L.: On the use of coloured Petri nets for object-oriented design. In Cortadella, J., Reisig, W., eds.: Applications and Theory of Petri Nets 2004 25$^{\text{th}}$ International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004. Volume 3099 of Proceedings Series: Lecture Notes in Computer Science. Springer (2004) 117–136 ISBN: 3-540-22236-7.

8. Christensen, S., Hansen, N.D.: Coloured Petri nets extended with channels for synchronous communication. Daimi PB-390 (1992) Also in: Valette, R.: Lecture Notes in Computer Science, Vol. 815; Application and Theory of Petri Nets 1994, Proceedings 15th International Conference, Zaragoza, Spain, pages 159-178. Springer-Verlag, 1994. Abridged version; available at `http://www.daimi.au.dk/CPnets/publ/full-papers/ChrHan1994.pdf`.
9. Kummer, O., Wienberg, F., Duvigneau, M., Schumacher, J., Köhler, M., Moldt, D., Rölke, H., Valk, R.: An extensible editor and simulation engine for petri nets: Renew. In Cortadella, J., Reisig, W., eds.: Applications and Theory of Petri Nets 2004 25th International Conference, ICATPN 2004, Bologna, Italy, June 21-25, 2004. Volume 3099 of Proceedings Series: Lecture Notes in Computer Science. Springer (2004) 484–493 ISBN: 3-540-22236-7.
10. Renew: Renew – the reference net workshop. `http://www.renew.de/` (2004)
11. OMG: Unified modeling language specification, version 1.5. `http://www.omg.org/cgi-bin/doc?formal/03-03-01` (2003) Unified Modeling Language, v1.5, Object Management Group.
12. Ruby Homepage. `http://www.ruby-lang.org/en/` (2004)
13. Barros, J., Gomes, L.: Composable CPNs homepage. `http://www.uninova.pt/gres/ccpn` (2004)

BULLWHIP EFFECT AND SUPPLY CHAIN MODELLING AND ANALYSIS USING CPN TOOLS

Dragana Makajić-Nikolić, Biljana Panić, Mirko Vujošević

Operations Research Laboratory »Jovan Petric«,
Faculty of Organizational Sciences, University of Belgrade
gis@fon.bg.ac.yu, bilja@fon.bg.ac.yu, mirkov@fon.bg.ac.yu

Abstract. The paper presents some of the results obtained by studying Petri nets' capability for modeling and analysis of Supply Chain performances. It is well known that the absence of coordination in Supply Chain management causes the so-called Bullwhip Effect, in which fluctuations in orders increase as they move up the chain. A simple three-stage supply chain with one player at each stage – a retailer, a wholesaler and a manufacturer – is considered. The model of the chain is developed using a timed, hierarchical coloured Petri Net. Simulation and performance analysis have been performed applying software package CPN Tools.

1. INTRODUCTION

A Supply Chain (SC) includes all the participants and processes involved in the satisfaction of customer demand: transportation, storages, retailers, wholesalers, distributors and factories. A large number of participants, a variety of relations and processes, dynamics, the uncertainty and stochastics in material and information flow, and numerous managerial positions prove that Supply Chains should be considered as a complex system in which coordination is one of the key elements of management.

Very important Supply Chain processes are ordering and delivery of purchased amounts. These are multiple entangled and their disorder can lead to various unwanted effects. One of them is the so-called Bullwhip Effect in which fluctuations in orders increase as they move up the chain. In order to illustrate the SC functioning and particularly the bullwhip effect, Beer Game [10] was created at the beginning of the sixties in *Sloan School of Management, Massachusetts Institute of Technology (MIT)*.

This game simulates the SC performance with one participant per each phase and today it is being played all over the world – among students and top managers, to improve the approach to Supply Chain functioning. All the simulations led to the bullwhip effect – the participants of Supply Chain reacted inadequately to sudden changes in customer demands. However, the bullwhip effect appears in those supply chains where there is no subjectivity, i.e. in cases where decision-making rules are the same for all participants and seem to be rational. This paper presents the bullwhip effect in Supply Chain simulation via a timed hierarchical coloured Petri Net [6].

Numerous papers give examples of Petri Nets used in modeling and SC analysis. In order to analyse the potential benefit of SC management, Van der Vorst, et. al. [12] modeled the SC dynamic behavior. The authors consider SC as a business process, which is to be redesigned, and in that case many different redesign scenarios are to be tested. PN is used to support a decision-maker in choosing the best-fit scenario. The authors developed the simulation/Petri-Net modeling approach to Supply Chain analysis and demonstrated their model using a case study from the food industry. Their model uses eight different performance indicators (three cost-based indicators and five service-based indicators).

Supply Chains in food industry are also modeled in [1]. The authors propose a supply chain management of perishable items combining the TTI and Wireless technologies. The proposed futuristic SC is compared with the existing situation using simulation of Generalized Stochastic PN.

An approach to Supply Chain process management (SCPM) based on high-level Petri-nets, called XML-nets is presented in [9]. It is shown that SCPM as a whole can be improved by employing XML-nets. The major advantages of the approach are: capturing process dynamics, concurrency and parallelism of processes as well as asynchronous operations; capabilities for modeling complex objects with a hierarchical structure; exchange of intra- and inter-organizational data. The paper also shows the architecture and functions of an XML-net based prototype software tool supporting SCPM.

Computer Integrated Manufacturing Open System Architecture (CIMOSA) based process behavior rules and Object-oriented PN (OPTN) are used in [5] to model the routing structures of a typical SC process. The OPTN, which they consider, consists of two parts: an internal structure, e.g. an object (composed of state places and activity transitions) and an external structure (a set of ports places which form the interface of OPTN). The authors use P-invariant to obtain the system properties from object properties.

In [7] the authors show how a Generalized Stochastic PN bridges the gap between application formalism, like process chains, and analysis methods for concurrent systems, like PN analysis methods. A small supply chain - involving a manufacturer, one of his suppliers and a transportation company, is used to illustrate their assertion.

In paper [8] the authors propose an implementation of an incremental approach to modeling discrete events systems at the structural level of systems specification. They consider the entire system and coordinate decisions at each stage of the Supply Chain. Using the example of the Beer Game, a systematic method supports the bottom-up construction of re-usable models of supply chains in the Petri nets domain.

This paper is intended to show that Petri nets may be used for studying the bullwhip effect, and especially for experimenting on how different replenishment strategies affect the parameters of certain participants and of the entire Supply Chain. Such an application of Petri nets to Supply Chains may be very useful for the education of postgraduate students. This application requires elementary knowledge of CPN and CPN tools software [3,4] as well as the knowledge of SC functioning and the bullwhip effect phenomenon.

The paper is organized as follows: Section 2 gives the basic terms of supply chains, bullwhip effect and beer game performed with two student groups, simulating two SCs; Section 3 includes the CPN model of a simplified SC, created in CPN Tools program. The same program was used for the simulation of CPN supply chain for identical customer demands as in the already mentioned beer game; Section 4 presents simulation results.

2. BULLWHIP EFFECT IN A SUPLLY CHAIN

Supply Chain coordination functions well as long as all stages of the chain take actions that together increase total supply chain profits. Each participant (phase) of the chain should maintain its actions in a good relation to other participants and the supply chain in general and make decisions beneficial to the whole chain. If the coordination is weak or does not exist at all, a conflict of objectives appears among different participants, who try to maximize personal profits. Besides, all the relevant information for some reason can be unreachable to chain participants, or the information can get deformed in non-linear activities of some parts of chain which leads to irregular comprehension. All these lead to the so-called Bullwhip Effect resulting from information disorder within a supply chain. Different chain phases have different calculations of demand quantity, thus the longer the chain between the retailer and wholesaler the bigger the demand variation.

A retailer can realize a small variation in customers' demands as a growing trend and purchase from a wholesaler more products than he needs. Increased order at wholesalers is larger than at retailers as the wholesaler cannot regularly comprehend the increased order. As the chain grows longer the order is larger. If a retailer plans the product promotion he can increase the order. If a manufacturer comprehends the increased demand as constant growth and in the same manner makes purchases, he will face the problem of inventory surplus in the end of promoting period [2]. A variation in demands increases production expenses and expenses of the whole supply chain in an effort to deliver the ordered quantity in time. A manufacturer accomplishes demanded capacity and production but when the orders come to a former level, he remains with the surplus of capacity and inventory.

Any factor that leads to either local optimization by different stages of the supply chain or an increase in information distortion and variability within the supply chain is an obstacle to coordination. The major obstacles are[2]:

- Incentive obstacles – a situation in which incentives are offered to different stages or participants in a supply chain and focus only on the local impact of an action result in decisions that do not maximize total supply chain profits.
- Information processing obstacles – situations in which demand information is distorted as it moves between different stages of the supply chain, leading to increased variability in orders within the supply chain. Demand forecasting based on the stream of orders received from the downstream stage results in a

magnification of fluctuations in demand as we move up the supply chain from the retailer to the manufacturer.

- Operational obstacles – actions taken in the course of placing and filling orders that lead to an increase in variability.
  - When a firm places orders in lot sizes that are much larger than the lot sizes in which demand arises, variability of orders is magnified up the supply chain.
  - The bullwhip effect is magnified if replenishment lead times between stages are long
  - A situation in which a high-demand product is in short supply often arises within the supply chain. Rationing schemes that allocate limited production in proportion to the orders placed by retailers lead to a magnification of the bullwhip effect.
- Pricing obstacles – situations in which the pricing policies for a product (lot size – based quantity discounts, price fluctuations) lead to an increase in variability of orders placed.
- Behavioral obstacles – problems in learning within organizations that contribute to the bullwhip effect. These problems are related to the way the supply chain is structured and the communication between different stages.

The bullwhip effect [2]:

- Increases the level of inventory, accordingly the warehouse space is more occupied, all of which leads to an increase in holding or carrying costs of storage services;
- Prolongs the lead time – the time period from the moment of purchasing to the moment of receiving the order;
- Demands more efficient transportation to satisfy the increased demand, which leads to a high transportation cost;
- Increases labor costs;
- Decreases the level of product availability, which can lead to deficiency of retail inventory;
- Causes problems in each phase disturbing relations within a supply chain since participants' efforts do not have a positive outcome – leads to distrust among participants.

2.1. BEER GAME

Originally, the game was created as a board game that demonstrates beer production and distribution, as shown in Figure 1.

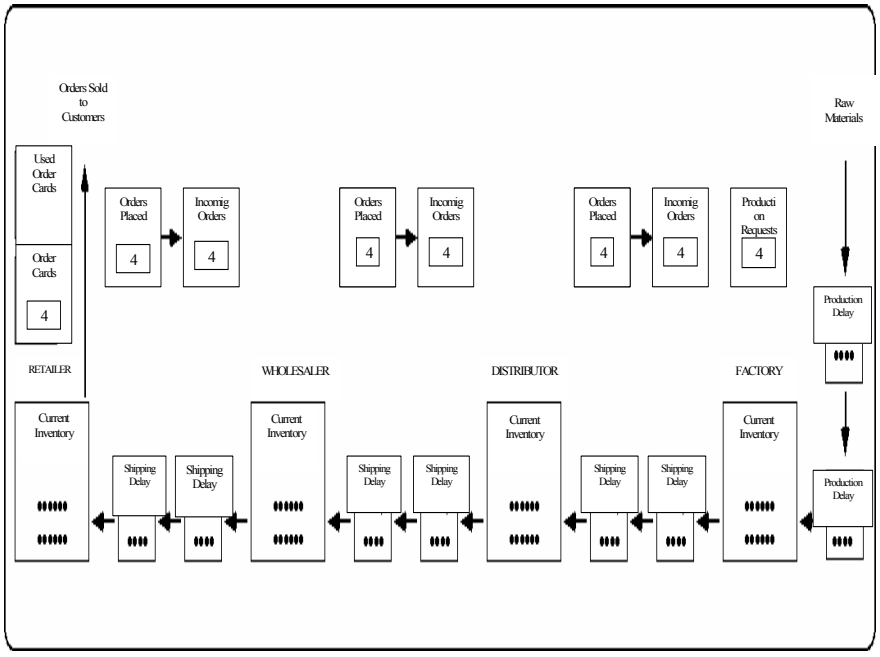


Fig. 1. Beer Game board, showing initial conditions [10]

Teams play a game whose objective is to minimize the expenses of an overall supply chain. The winner is the team with the lowest expenses. Each participant must initially invest $1, whereas the winner takes the whole investment. Teams are divided into four sectors: Retailer, Wholesaler, Distributor and Factory. One or two persons direct over each of the sectors. Coins represent beer cases, and a deck of cards is customer's demand. A customer comes to a retailer to buy a beer. The retailer meets customer's needs selling the beer from his inventory. Each incomplete order is the backorder in the following period. The retailer makes orders at the wholesaler that meets the demand from his inventory. The wholesaler purchases orders from a distributor, a distributor from a factory and the factory purchases raw materials from a supplier. The ordered products displace through an introduction phase and a transport phase, and two time units (two iteration simulations) are the period for which the product is moved from one phase to the other. Storage expenses are $0.50 per week, and the expenses incurred for a week in backorder are $1 per case.

The game start is equal for all: each participant has the inventory of 12 beer cases, and initial demand in each phase is 4 cases. In the course of several weeks, players learn the mechanisms of purchasing, to deposit inventory etc., and during that period the demand is constant – 4 cases per week. During the first three weeks, the players can order only 4 cases per week, which is logical since the demand is also 4 cases. In the beginning of the fourth week a player can order as much as he wants, and he is said that a customer's demand can vary. One of his tasks is to foresee the demand, according to what he makes orders, bearing in mind that the delivery period is two weeks. Thus, the player must foresee the demand in a two weeks' time and accordingly make an order. The game lasts for 50 simulated weeks, but the wanted effects are obvious far earlier [10].

Each player possesses good local information (about his inventory, remaining orders, receiving amounts from his direct supplier and amounts he has just delivered to the player he supplies), but he is not in possession of global information. Only the retailer knows the last customer's demand, and the others can get the information only on the demand of immediate customer. Communication between the participants is not allowed. Barriers in communication and lack of information lead to inadequate coordination in a supply chain.

The game indicated that the average expenses in MIT were about $2000, sometimes the expenses would come to $1000, but they were always higher than $1000. However, optimal expenses, calculated according to information available to players, are about $200. Each game shows the same behaviour models and reveals the bullwhip effect [10].

The beer-game was carried out with the students of specialist studies in industrial engineering, organized in cooperation by the Ecole Centrale Paris and the Faculty of Organizational Sciences at the University of Belgrade, within a course in Industrial Logistics. The students were divided into two teams, with each team representing one supply chain with four sectors: a retailer, wholesaler, distributor and manufacturer (factory). The game, developed according to the rules described above, lasted for 23 simulated weeks.

The game showed the expected results, close to the MIT's results. The total costs incurred by the first team were $3580.5 (inventory costs $2900,5, backorder costs $680) and of the second $2497.5 (inventory costs $1766,5, backorder costs $731), what is in conformity with the results obtained at the MIT. Cost distribution within teams was as follows:

- Team 1: retailer $201 (inventory $141, backorder $60), wholesaler $748.5 (inventory $545,5, backorder $203), distributor $1305 (inventory $1009, backorder $269) and manufacturer $1326 (inventory $1205, backorder $121);
- Team 2: retailer $327.5 (inventory $212.5, backorder $115), wholesaler $412 (inventory $272, backorder $140), distributor $428 (inventory $246, backorder $182) and manufacturer $1330 (inventory $1168, backorder $162).

Such results were expected – costs increase as we go from a retailer to a manufacturer.

When the students were asked to estimate the average demand, their estimation results were far higher than actual demand. Actual demands of the last-in-chain customers were the same for both teams: 4, 4, 4, 8, 10, 12, 12, 10, 8, 8, 8, 9, 8, 8, 8, 0, 0, 7, 8, 8, 8, 8, 8 cases. Such demand was selected in accordance with the beer game played at the MIT. Figure 2 shows customer demands and responding amounts of other SC participants.

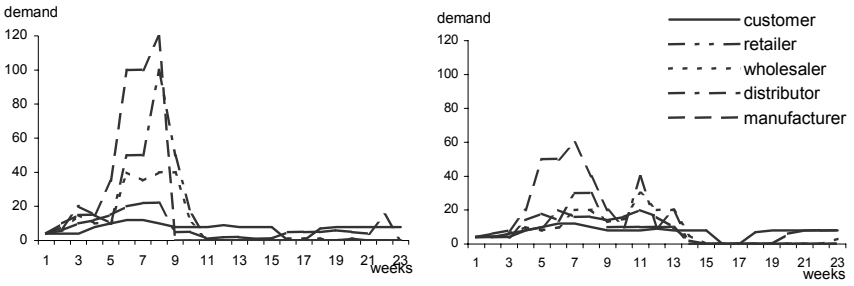


Fig. 2. Bullwhip Effect.

The left-hand side of the Figure shows the reactions by the participants of the first group and the right-hand side also shows the reactions of the participants of the first student group to specified demand. The group whose results are shown on the right-hand side took a higher risk in the game and this is why they had a lower bullwhip effect.

3. SUPPLY CHAIN MODELING USING CPN TOOLS

The primary objective of this paper is to illustrate how PN can be used in learning about and studying the bullwhip effect. Thus, the PN of a simplified SC was formed, with one participant per each phase – a retailer, wholesaler, distributor and manufacturer, and then it was simulated with CPN Tools [3,4].

The supply chain was simplified to a line of participants taking different places within a chain and making decisions according to the same rules. The two main activities of each participant are: delivery of purchased products and forming one's orders according to new orders. The selection of strategies by SC participants reduces to their decision on how they will perform these two activities. We have chosen here a simple scenario according to which all SC participants behave in the same way, i.e., they perform the mentioned activities as follows:

1. Delivery of purchased products. At the moment of receiving the order it has to be checked if the inventory stores the needed products. If the inventory shows sufficient amounts of products, the ordered, total amount is delivered. Otherwise, if the inventory amounts do not suffice it is possible either to wait for needed inventory product deposit or to deliver the incomplete inventory, with belated additional quantity. In the presented CPN the chain participants deliver even incomplete orders.

2. Forming one's orders according to new orders. Since a certain amount of products is delivered from a personal inventory, new orders are made from the next SC participant in order to keep the needed level of inventory. The decision on order quantity, in the modeled SC, is being made in the following way: if the inventory stores enough products, the same amount delivered to a customer is to be ordered, since the estimation is that the following delivery is to be of the same quantity. If the inventory shows the lack of products, the amount that could not have been delivered is to be ordered, and, as in the previous case, the amount that will suffice for the next order.

3.1. NET STRUCTURE AND DECLARATIONS

The described SC has been modeled with timed Hierarchical Coloured Petri Nets [6]. The net structure consists of five modules (pages). The top level of the model is the whole Supply Chain: a customer and the four mentioned phases, Figure 3. The three phases (participants): retailer, wholesaler, distributor and manufacturer are presented by means of substitution transitions. A sub-page models each participant. A special sub-page Manufacturer models a manufacturer, as a specific participant in the end of SC. Retailer, wholesaler and distributor are modeled by three instances of sub-page Supplier. A customer is presented by a place named customer whose initial marking represents demand in time and he directs the simulation.

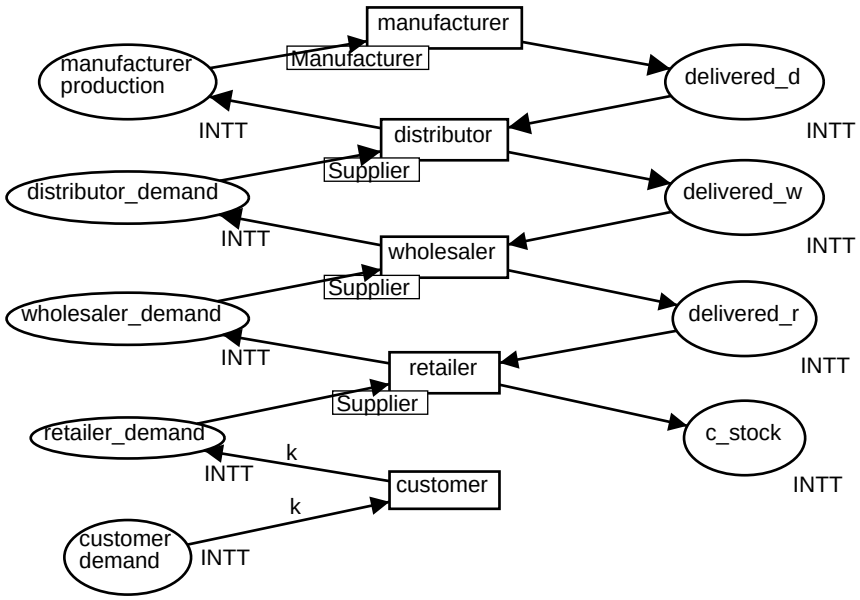


Fig. 3. Top-level Petri Net of simplified Supply Chain

It is possible to include various parameters within the SC model: prices, time, amounts of goods, a resource etc. Nevertheless, the bullwhip effect analysis requires only the follow-up on orders and deliveries in time. Figure 4 shows the declaration colour sets, variables, and function used in the net.

```
Declarations
color INT = int;
color INTT = INT timed;
var k, m, n, s: INTT;
fun rest(i,j) =if (i>=j) then i-j else 0;
fun order(i,j) = if j>=i then i else i+i-j;
val a=2;
val b=2;
```

Fig. 4. Declaration of colour sets, variables and functions

As can be seen from the declaration, to form the PN model of the SC, it is sufficient to use one standard colour set (INT) and define one timed colour set (INTT).

Function `rest(i,j)` models activity 1 described above – Delivery of purchased products. Variable i represents the order and variable j represents the inventory. The result of function `rest(i,j)` is the amount of ordered product which is not delivered. If the inventory shows sufficient amounts of products, the total amount is delivered and function `rest(i,j)` results in 0.

Function `order(i,j)` models activity 2 – Forming one's orders according to new orders. Variables i and j represent a received order and the inventory, respectively. Function `order(i,j)` behaves as follows: if the inventory stores enough products, the amount `i` will be ordered; if the inventory shows the lack of products, the amount that could not have been delivered (`i-j`) is to be ordered, and, as in the previous case, the amount `i` that will suffice the next order.

Values (constants) a and b refer to the duration of delivery and production periods. The SC is modeled by timed CPN. This simplified SC suggests creating immediate personal orders, whereas the production and delivery procedures last.

3.2. SUB-PAGE SUPPLIER

A sub-page Supplier, described in Figure 5, was used to model the retailer, wholesaler and distributor, each with one of its instances. It became possible to include more suppliers between the customer and factory. Each of them would be modeled by one instance of sub-page supplier. This was possible as the assumption of modeled SC was that each of the participants makes decisions according to the same rules. If we wish each SC participant's behaviour to be different from others', we

have to form a new Subpage for each of them (or, at least, for those whose behaviour is different).

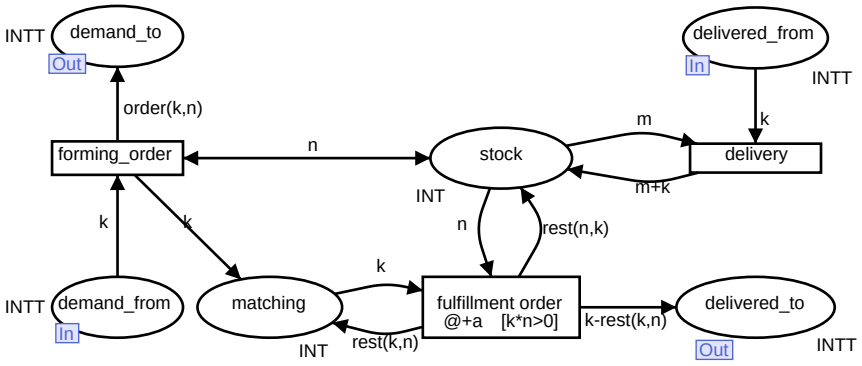


Fig. 5. Sub-page Supplier

According to the receiving order in place demand_from and inventory status in place stock, a supplier makes his own order (the second of the two basic activities), which has been modeled by transition forming_order and the function order (i,j), shown in Figure 4. The starting assumption was that forming one's order is performed much faster than the remaining activities in the SC, so the transition forming_order is momentary

Transition fulfillment_order models the first basic activity of SC participants: delivery of a needed. amount. Here, the inventory amounts are compared to the order quantity, and a decision on a delivery amount is to be made. If the inventory stores enough goods the value of function rest (k,n) (see Figure 4) will be 0, and of the function rest (n,k) = n-k. Thus, a token, which corresponds to the order quantity, will appear in the place delivered_to, the inventory will be reduced by k, and there will be no backorders. If the inventory goods are insufficient, the value of function rest (n,k) will be 0, and of the function rest (k,n) = k-n. A token n (total inventory goods) will appear in the place delivered_to, the inventory state will be 0, and a token k-n, modeling the quantity of undelivered goods, will appear in the place matching. The duration of transition fulfillment_order is (@+a), because, according to the rules of the game played at the MIT, product delivery lasts 2 weeks. It can be seen from Figure 4 that a=2. A guard function [k*n>0] in the transition ensures this transition to occur only when an order exists and when there are products in stock.

3.3. SUB-PAGE MANUFACTURER

A manufacturer is modeled by a sub-page Manufacturer (Figure 6). He decides, according to received orders and his inventory state, what quantity of goods to deliver. He is the last SC link and, in a way, he acts in the same way as suppliers, with a difference that instead of making an order for the next chain participant, he decides

what amounts to produce in the following period. This was modeled by the transition manufacturing. As the production and delivery were assumed to last 2 weeks each, duration (@+b) was assigned to this transition. It can be seen from Figure 4 that b=2.

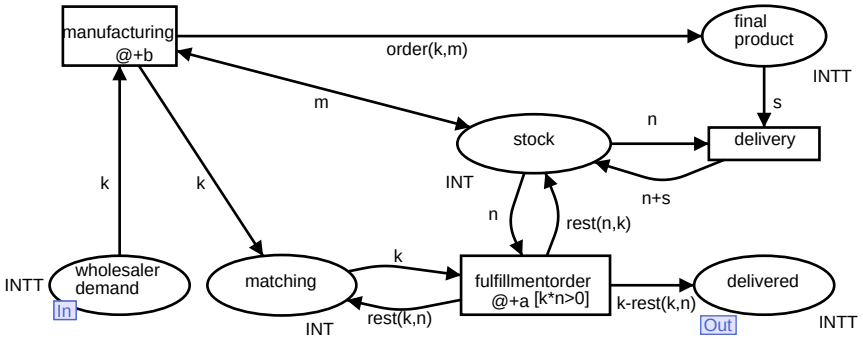


Fig. 6. Sub-page Manufacturer

Other transitions and places of this sub-page are analogous to the transitions and places of the sub-page Supplier.

4. SIMULATION RESULTS

A CPN formed in this way may be used for various experiments. It is possible to simulate different strategies of delivery and forming one's inventories and then analyze the bullwhip effect they cause. This may be accomplished by simply defining functions `rest(i,j)` and `order(i,j)` so as to model a selected strategy. It is also possible to add stochastic behaviour to a CPN in the sense of defining functions as stochastic or defining a stochastic duration of activities and transitions.

Instead of making this type of experiments, we will show here the effect obtained when simulating the demand described in 2.1 using the CPN described in section 3 as well as all the information obtainable from a simulation report. Simulation starts from the assumption that the initial system state was stable and remained stable for a certain time period and that a sudden increase in demand occurred after that. Since it is possible to compare simulation results, the initial net marking models the next state, the same state as in the beer-game with students:

- Retailer, wholesaler, distributor and manufacturer, each of them initially has 4 units of an inventory product, after the first week each has 4 units, and after the second week 4 units, which relates to the following initial marking:

1`4	in places stock@Supplier (all instances)
1`4@+1++1`4@+2	in places delivered_from@Supplier (all instances)
1`4@+1++1`4@+2	in place final product@Manufacturer

- A customer orders products once a week. Customer's demand in the first 3 weeks is 4 units per week, during the following weeks it is growing: 8, 10, 12, 12, 10, 8, 8, 8, 9, 8, 8, 8, 0, 0, 7, and in the last five weeks it is stabilized at 8 product units, which relates to the initial marking:

1`4@+0 ++ 1`4@+1 ++ 1`4@+2 ++ 1`8@+3 ++ 1`10@+4 ++ 1`12@+5 ++
1`12@+6 ++ 1`10@+7 ++ 1`8@+8 ++ 1`8@+9 ++ 1`8@+10 ++
1`9@+11 ++ 1`8@+12 ++ 1`8@+13 ++ 1`8@+14 ++ 1`0@+15 ++
1`0@+16 ++ 1`7@+171 ++ 1`8@+18 ++ 1`8@+19 ++ 1`8@+20 ++
1`8@+21 ++ 1`8@+22 in place customer@Top page

Various chain performances can be obtained by simulation of CPN models. Figure 7 demonstrates the bullwhip effect, i.e. the reactions of retailer, wholesaler, distributor and manufacturer to customer demand.

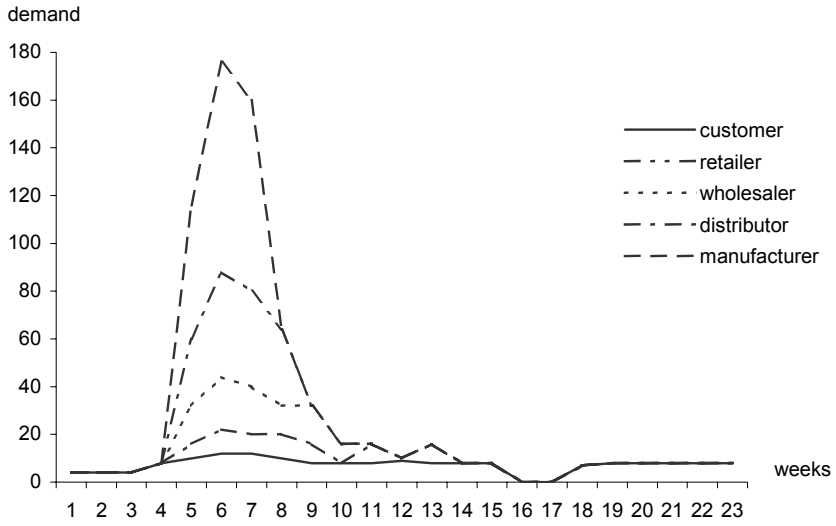


Fig. 7. Customer demand within Supply Chain – Bullwhip Effect

An expected result was obtained, close to the MIT's results. Each participant in the chain reacted inadequately and the most drastic result was recorded with the manufacturer – the last SC participant. Such behaviour is predetermined by function `order(i,j)` and some more sophisticated strategy would certainly have given beter results. This can be noted through a comparison with the results obtained in simulations with the students.

Figure 7 demonstrates a stronger bullwhip effect than Figure 2, than it was the case of students whose decisions were often very risky.

Simulation results demonstrate the consequence of bullwhip effect – inventory increase in the chain. Figure 8 shows the course of inventory state during the period of 23 weeks.

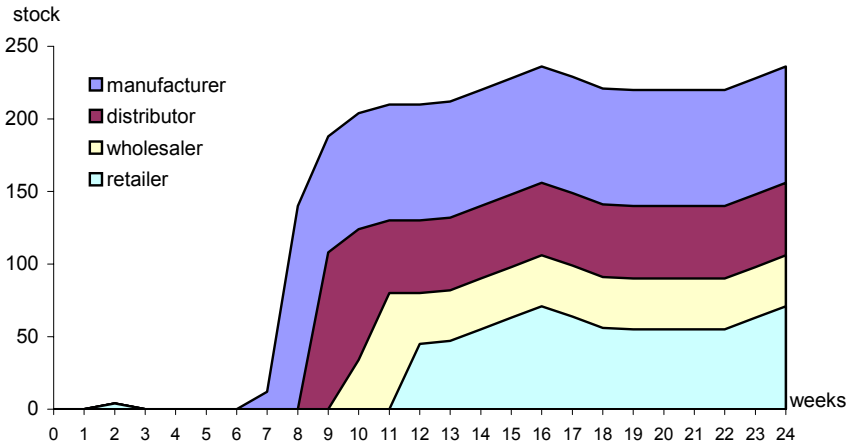


Fig. 8. The quantity of inventory goods in time

Figure 8 shows the course of backorder during the same period of 23 weeks.

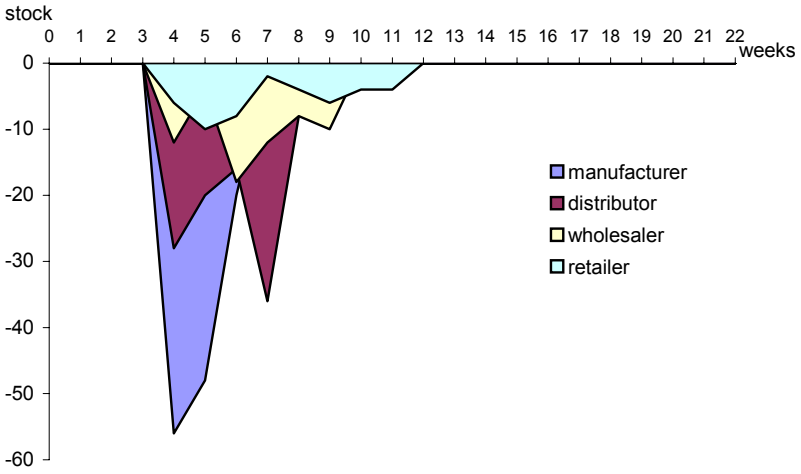


Fig. 9. Backorder in time

Using cost values from Section 2.1.: storage expenses -- $0.50 per week, and the expenses for a week in backorder - $1 per case, it is possible to calculate SC costs which represent the main indicator of chain functioning. The costs incurred by each participant and the costs of the entire SC are presented in Table 1.

Table 1

	retailer	wholesaler	distributor	manufacturer	SC
inventory cost	379,5	664	1113	1829	3985,5
backorder cost	44	64	108	124	340
total cost	423,5	728	1221	1953	**4325,5**

Simulations revealed one more phenomenon. Each chain participant has inventory goods surplus after 23 weeks. By analyzing the model, one could conclude that the results of all simulations have to be the same. However, it can be seen in Figure 10 that simulation results vary from one simulation to another. Figure 10 describes the variation in inventory goods surplus by the end of observation period. Here are the results of 20 successive simulations of the same initial marking, i.e. the same demand.

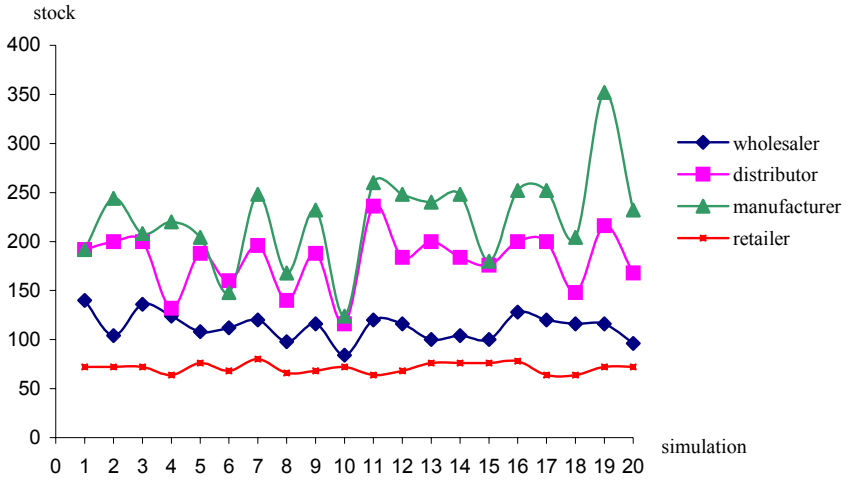


Fig. 10. Inventory goods surplus after 23 weeks in 20 successive simulations

Different results are obtained for the following reason: during the simulation, several transitions occur at the same time instant and the sequence of their occurrence affects the values tokens may take. For example, in the Sub-page Supplier, the following transitions may occur at the same time instant: forming_order, fulfillment_order and delivery. However, these transitions will not occur literally

simultaneously, but in some random sequence. The quantity of supplier's order, which is determined by the occurrence of transition forming_order and which affects directly an inventory goods surplus, depends on a received order (token k) and inventory status in place sctock (token n). Each occurrence of transition delivery increases the inventory status, while each occurrence of transition fulfillment_order decreases it. This means that a different sequence of occurrences of the transitions: forming_order, fulfillment_order and delivery results in a different supplier's order.

5. CONCLUDING REMARKS

The paper describes how a Supply Chain can be modeled and analysed by CPN and CPN Tools. The CPN Tools package permits the functions of forming one's orders and delivery to be defined in an easy and fast way, which is not possible by using the modeling tools known to the authors. According to the results, the simulations can measure different performances: reactions of participants to sudden changes in customer demands (the bullwhip effect), changes in supply level in a time, surplus of inventory goods, the costs of SC participants and of the entire SC, etc. It has been shown that it is possible to form a PN model of a Supply Chain which allows easy experimenting with different scenarios, i.e., different strategies of SC participants. Unlike other beer-game implementations, where one participant must be engaged for each stage, here one participant may simultaneously experiment with the strategies of all SC stages. With CPN, the phenomena, corresponding to a real situation, e.g. a random time delivery and different decision rules in phases, as well as several participants in each SC stage can be applied to CPN model. The bullwhip effect in the non-linear SC is of special interest.

REFERENCES

[1] N. Bhushan, K. Gummaraju *A Petri Net Based Simulation Approach for Evaluating Benefits of Time Temperature Indicator and Wireless Technologies in Perishable* Goods Retail Management FOODSIM'2002 The Second International Conference on Simulation and Modeling in the Food and Bio-Industry, June 17-18, 2002

[2] S. Chopra, P. Meindl, *Supply Chain Management: Strategy, Planning, and operation,* Prentice Hall, Upper Saddle River, New Jersey, 2001.

[3] Coloured Petri Nets at the University of Aarhus. www.daimi.au.dk/CPnets

[4] CPN Tools: www.wiki.daimi.au.dk/cpntools

[5] M. Dong, F.Frank Chen *Process modeling and analysis of manufacturing supply chain networks using object-oriented Petri nets* Robotics and Computer-Inetgrated Manufacturing, Volume 17, Issues1-2, pp. 121-129, February 2001.

[6] K. Jensen *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*.Volume 1,2,3, Springer-Verlag, 1997.

[7] P. Kemper. *Logistic Processes go Petri nets.*Philippi, S. (Hrsg.): 7. Workshop Algorithmen und Werkzeuge für Petri Netze, Koblenz: Universität Koblenz-Landau, pages 69-74, 7/2000.

[8] R.V. Landeghem, C.-V. Bobeanu *Formal modelling Of Supply Chain: An Incremental Approach Using Petri Nets*, Proceedings 14th European Simulation Symposium A. Verbraeck, W. Krug, eds. (c) SCS Europe BVBA, 2002

[9] M. von Mevius, R. Pibernik *Process Management in Supply Chains – A New Petri-Net Based Approach* Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) Big Island, Hawaii - January 05 - 08, 2004

[10] J. D. Sterman, *Teaching Takes Off: Flight Simulators for Management Education.* OR/MS Today (Oct), 40-44, 1992.

[11] T. H. Truong, F Azadivar *Simulation Based Optimization For Supply Chain Configuration Design* Proceedings of the 2003 Winter Simulation Conference, December 7-10 2003., pp. 1268-1275.

[12] van der Vorst, Jack G.A.J., A. J.M. Beulens, P. van Beek (2000). *Modelling and Simulating Multi-Echelon Food Systems*, European Journal of Operational Research, Vol. 122, pp. 354-366.