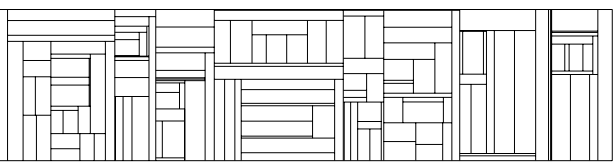
ISSN 0105-8517

Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools Aarhus, Denmark, August 29-31, 2001

Kurt Jensen
(Ed.)

DAIMI PB – 554
August 2001

DATALOGISK INSTITUT
AARHUS UNIVERSITET
Ny Munkegade, Bygn. 540
8000 Århus C



Preface

This booklet contains the proceedings of the Third Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, August 29-31, 2001. 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/workshop01/

Coloured Petri Nets and the CPN tools are now used by more than 750 organisations in 50 countries all over the world (including 150 commercial enterprises). 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:

Jonathan Billington	Australia	(j.billington@unisa.edu.au)
Søren Christensen	Denmark	(schristensen@daimi.au.dk)
Jorge de Figueiredo	Brazil	(abrantes@dsc.ufpb.br)
Nisse Husberg	Finland	(Nisse.Husberg@hut.fi)
Kurt Jensen (chair)	Denmark	(kjensen@daimi.au.dk)
Charles Lakos	Australia	(Charles.Lakos@adelaide.edu.au)
Alexander Levis	USA	(alevis@gmu.edu)
Daniel Moldt	Germany	(moldt@informatik.uni-hamburg.de)
Laure Petrucci	France	(petrucci@lsv.ens-cachan.fr)
Dan Simpson	UK	(Dan.Simpson@brighton.ac.uk)
Edwin Stear	USA	(estear@aol.com)
Robert Valette	France	(robert@laas.fr)
Rüdiger Valk	Germany	(valk@informatik.uni-hamburg.de)
Klaus Voss	Germany	(klaus.voss@gmd.de)
Jianli Xu	Finland	(jianli.xu@research.nokia.com)
Wlodek Zuberek	Canada	(wlodek@cs.mun.ca)

The programme committee has accepted 7 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 two CPN Workshops can be found via the web pages: http://www.daimi.aau.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

Modelling Feature Interaction Patterns in Nokia Mobile Phones using Coloured Petri Nets and Design/CPN

Louise Lorentsen[1], Antti-Pekka Tuovinen [2], and Jianli Xu [2]

[1]Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 ÅRHUS N, DENMARK,
louisel@daimi.au.dk
[2]Software Technology Laboratory, Nokia Research Center
P.O. Box 407, FIN-00045 NOKIA GROUP, FINLAND
{Antti-Pekka.Tuovinen,Jianli.Xu}@nokia.com

Abstract. This paper describes the first results of a project on modelling of important feature interaction patterns of Nokia mobile phones using Coloured Petri Nets. A modern mobile phone supports many features: voice and data calls, text messaging, personal information management (phonebook and calendar), WAP browsing, games, etc. All these features are packaged into a handset with a small screen and a special purpose keypad. The limited user interface and the seamless intertwining of logically separate features cause many problems in the software development of the user interface of mobile phones. In this paper, we look at the problem of feature interaction in the user interface of Nokia mobile phones. We present a categorization of feature interactions and describe our approach to the modelling of feature interactions using Coloured Petri Nets (CP-nets or CPN). The CPN model is extended with visualisation and interaction facilities to allow the user to control and get information from simulations without interacting with the underlying CP-nets. The CPN model will be used in the design of new features to identify and analyse the interactions of features. The CPN model constructed in the project successfully identified inconsistencies in the specifications. Furthermore, the construction of the CPN model has lead to interesting ideas for possible improvements in the architecture of the mobile phone UI software system.

1 Introduction

The modern mobile phones provide increasingly fancy and complex features to the user via their user interface (UI). These features are implemented by UI applications in the mobile phone UI software architecture. In the development of the user interface software for a mobile phone, it is important to identify and clearly specify the right interactions between the separate features of the mobile phone at an early stage of the development. This helps to avoid costly delays in the integration phase of a set of independently developed features. Feature interaction means a dependency or interplay of features. The feature interactions can be conceptually simple usage dependencies or more complex combinations of independent behaviors. Precise descriptions of the feature interactions are also needed when planning the testing of the UI software. The types and the number of interactions a feature has with other features are direct indicators of the cost of developing the feature which is important information for the planning and management of the development effort.

In this paper, we describe the work of a joint project of Nokia Research Center and the CPN Group at the University of Aarhus, in which Coloured Petri Nets (CP-nets or CPN)[3] and its supporting Design/CPN tool [5] are used to model important patterns of feature interactions that can occur in the user interface of Nokia mobile phones. Currently, feature interactions are not systematically documented. Often the most complex interactions are not fully understood before the features are implemented. The goals of this work are to identify

categories of interactions that are specific to the domain and to create behavioral models that capture the typical feature interaction patterns in each category. The models describe the behavior of the UI at the level that the user of a handset may observe it. The models will help estimate the cost of developing the features of a product for a specific UI style, and most importantly they will help the design of UI applications by exposing critical behavioral patterns of feature interactions. The CPN models will also help in assessing how well the common UI software architecture supports the development of features for different UI styles.

It is necessary to have precise behavioral descriptions of the typical interaction patterns that can occur in specific styles. The UI designers and UI software designers who are developing new features for a specific UI style will use the CPN models to identify and analyse the interaction patterns of their new features. From the software development point of view, it is important to create a connection from the behavioral patterns to typical implementation patterns of the UI applications. The explicit behavioral models of the typical interaction patterns will also guide the development of the common UI software architecture that provides the infrastructure for realizing the behavioral patterns.

Coloured Petri Nets (CP-nets or CPN)[3] is a graphical modelling language with a well-defined semantics allowing simulation of the behaviour specified by the CPN models as well as formal analysis [4]. In contrast to many other modelling languages CP-nets are both state and action oriented. CP-nets has proven powerful for modelling of concurrent systems and a number of successful projects have demonstrated its usefulness in modelling and analysis of complex systems. Examples of this can be found in [2] which is a list of published papers describing industrial use of CP-nets. The list is maintained by the CPN group at the University of Aarhus.

The paper is organised as follows. Section 2 gives an overview of the project organisation. Section 3 contains an introduction to features and feature interactions in Nokia mobile phones and discusses a categorisation of feature interactions. Section 4 presents selected parts of the CPN model. Section 5 presents extensions to the CPN model implementing visualisation and interaction techniques. Finally, Sect. 6 contains the conclusions and a discussion of future work. The reader is assumed to be familiar with the basic ideas of High-level Petri Nets.

2 Project Description

The work described in this paper is about the mid-term results of a cooperation project between Nokia Research Center and the CPN group from the University of Aarhus. The project is called the MAFIA (Modelling and Analysis of Feature Interactions in mobile phone Architectures) project. MAFIA is an ongoing project started in November 2000 and the planned work resources for the first year are about 16 man months. Four researchers (1 full time and 3 part time) from the CPN Group and two researchers (part time) from Nokia Research Center are involved in the project. The aim of the project is threefold:

1. To identify typical interaction patterns between features in Nokia mobile phones. We will explain the terms *feature* and *interaction* in Sect. 3.
2. To build CPN models which captures the patterns and validate these models.
3. To produce documentation of the interaction patterns that can be used and understood by the UI designers and UI software developers.

The initial work has been done at Nokia Research Centre to validate the use of CP-nets in modelling feature interaction patterns in mobile phones before the joint project started.

The researchers at Nokia Research Centre have practical experience with CP-nets and the Design/CPN tool in other Nokia research projects, so the modelling work started immediately at the beginning of the project. One researcher from the CPN group worked full time at Nokia Research Center for six months to construct the CPN models and the necessary visualisation facilities for the model with help from the Nokia researchers. Other project team members from the CPN Group at the University of Aarhus provide guidance and technical support on the modelling and model analysis work.

Mobile phone UI specification documents and UI software architecture design documents have been studied thoroughly in the early phase of the project, the knowledge gained here is fundamental to the modelling work. After the modelling framework and the models of several key features and their interactions were ready, we organised a workshop for project team members, UI designers and UI software developers from product development teams. During the workshop participants from the development team provided valuable feedback about how the models and model visualisation facilities will be used in the product development. More typical complex feature interaction scenarios were collected during and after the workshop. The project team has good and ongoing contacts and communications with the software architecture team, UI designers and UI software developers.

In the current phase of the project we focus on modelling and simulation of important and complex feature interaction scenarios to identify all the important interaction patterns, check their correctness and specify them with CPN models. In the next phase of the project we will focus on validating the CPN models and how to produce documentation of the interaction patterns using the CPN models.

3 Feature Interactions in Nokia Mobile Phones

Modern mobile phones are feature-rich products. Besides basic communication capabilities, e.g., making and receiving calls and sending and receiving short text messages, they have an in-phone directory (phonebook), calculator, calendar, games, WAP for accessing wireless internet services etc. The next generation of mobile devices will add location sensitive services and multimedia capabilities. The diversification in the product families is great due to the different national network standards and market segmentation.

The user interface of a mobile phone can be characterized as task-oriented. This means that the mobile phone UI is designed to support directly the main functions of the device. This is very different from a traditional PC that has a generic UI that supports a wide range of applications with a uniform way of launching the applications and accessing the data. For example, a phone may have keys assigned permanently for beginning and ending a call. Further, when browsing the contact information stored in the phonebook or calendar, it should always be possible to call those parties by a single press of a key. This design philosophy stems directly from the domain, i.e., from the requirements and needs of mobile phone users and from the physical and economical constraints of the devices.

The *UI specifications* of a product is a written document that define the appearance and behavior of the phone features. The UI specifications communicate the design of the user interface and the flow of user interaction to the UI software developers. The specifications may also have descriptions of the interactions between the features. However, feature interactions are not described in a systematic way and finding all interactions of a feature means reading through a large amount of documents.

Nokia mobile phones have several different UI styles for different families of products. The UI style is an important part of the product brand and it has a relatively long lifetime. It describes the physical structure of the UI and the basic mechanisms of user interaction. The UI style of a mobile phone captures many assumptions about the needs, expectations, and lifestyle of the intended user groups. Therefore it is important to consider also the feature interaction issues at an early stage of the system family conception when the key features are being identified. One of the goals of the MAFIA project is to develop a systematic methodology for describing interactions between phone features at the level of the behavior as observed by the user of the handset. From the system family perspective, the idea is to build models of typical interaction patterns shared by products conforming to the same UI style that dictates most of the interactions. In a system family, these models span a number of products and several generations of the UI style.

3.1 Types of Feature Interactions

By studying the UI specifications we have identified three main types of interactions that all stem from different sources. Below we give a categorisation of the three types (I - III). An interaction between two features can fall into more than one category.

I. The first category of interactions comes from the need of the features to use each other (called the *use interactions*). For instance, the task-oriented user interface design requires that when browsing the phone numbers stored in the phone, a call can be made to a number directly from the browser. This represents an interaction between the phonebook and mobile originated call features that is necessary to deliver a smooth and seamless service to the user.

II. The second category of interactions comes from the need to share the limited UI resources, e.g., the screen and the keypad, between many features that can be activated independently of each other. Because of the prioritisation of the users tasks (and the associated features), important events may interrupt less important activities, e.g.,
   - an incoming call screens phonebook browsing for the duration of the call
   - hang-up key stops search from phonebook (the browser is killed)
   - an incoming call suspends a game but the game is saved so the it can be continued after the call is terminated

III. The third category involves interactions where one feature affects other features by making them unavailable or by modifying their behavior in some other way. For instance, the any key answer feature makes it possible to answer an incoming call by pressing any key on the keypad and the key guard feature locks the keypad for accidental key presses. The combined effect of these features is that if any key answer is enabled and key guard is on, an incoming call can be answered only by pressing the 'off-hook' key. However, once the call is open, key guard is disabled for the duration of the call and then enabled again automatically. This scenario can be made more complex ad nauseatum by adding other simultaneous events, e.g., calendar alarm and warning of low battery level.

The use interactions (the first category) are thoroughly specified in the UI specifications and they are not problematic from the implementation point of view. However, the interactions of the second and the third categories are much more difficult to manage in the software design and implementation; they also lack systematic documentation. Therefore, it is the task of the

MAFIA project to concentrate on modeling and documenting the typical feature interaction patterns that belong to the latter categories.

In the first phase of the project we have mainly concentrated on interactions in category II of the list above. This is reflected in the CPN model presented in Sect. 4.

4 CPN model of Feature Interactions in Mobile Phones

This section presents selected parts of the CPN model developed in the MAFIA project. The CPN model does not capture the full mobile phone UI software architecture but it concentrates on a number of selected features that are interesting from the feature interaction perspective. The purpose of the section is twofold. First, to give an overview of the CPN model and second, to give an idea of the complexity of the CPN model and the abstraction level chosen.

4.1 Overview of the CPN model

Figure 1 gives an overview of the CPN model by showing how it has been hierarchically structured into 22 modules (subnets). The subnets are in CPN terminology also referred to as pages and we will use this term throughout the paper. Each node in Fig.1 represents a page of the CPN model. An arc between two nodes indicates that the source node contains a so-called substitution transition whose behaviour is described in the page represented by the destination node.

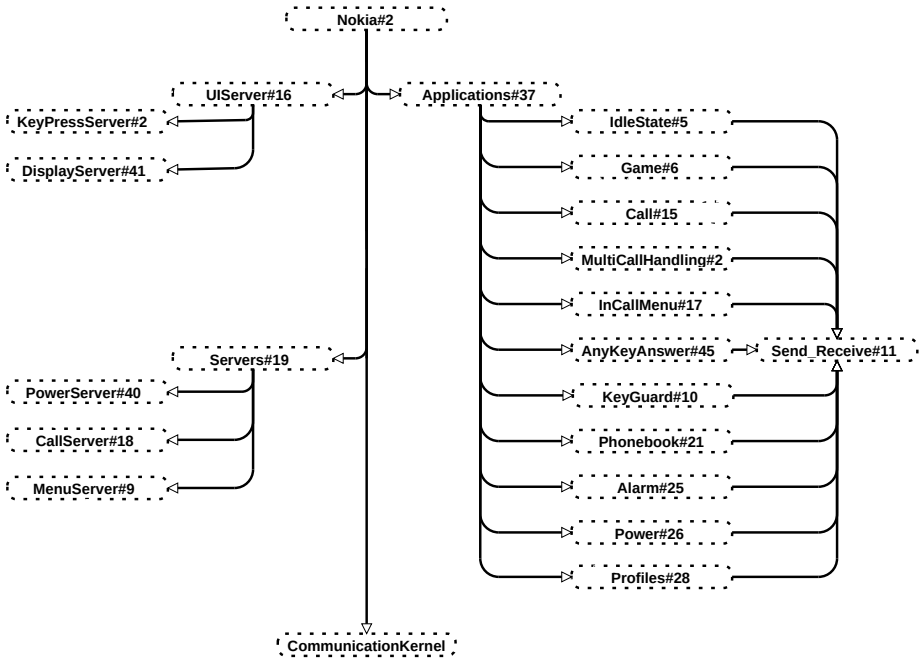


Fig. 1. The hierarchy page.

The CPN model consists of four main parts corresponding to four concepts of the mobile phone UI software architecture: applications, servers, UI controller, and communication kernel.

Applications. *Applications* implement the features of the mobile phone. The CPN model presented here includes 11 features: idle state, game, call, multi call handling, in-call menu, any key answer, key guard, phonebook, alarm, power, and profiles. Applications make the feature available to the user via a user interface.

Servers. *Servers* implement the basic capabilities of the phone. Applications implement the behaviour of features by using the services of servers. The CPN model presented here includes three servers: call server, power server and menu server. Servers do not have user interfaces.

UI controller. The role of the *UI controller* is to handle the user interfaces used by the applications to present the features to the user of the mobile phone. Such user interfaces include information about which text, icons, graphics etc. should be shown on the display of the mobile phone, a mapping of user keypresses to events, tones to be played etc. When the applications are running they request the UI controller to access the UI. The applications provide the UI controller with the graphics, key press mapping, tones etc. The UI controller will present the user interface to the user (using the display, tones etc. of the mobile phone) and use the key press mapping to map user input to actions which will be returned to the application. There are often several active applications at the same time. Hence, the UI controller will need to apply some scheduling mechanism to control the applications' access to the limited UI resources. We will give an example of this in Sect. 5.1.

Communication kernel. Servers and applications are communicating by means of asynchronous message passing. The messages are sent through the *communication kernel* which implements the protocol used in the communication between the applications, servers, and the user interface (UI controller).

The page Nokia depicted in Fig. 2 is the top-most page of the CPN model and provides the most abstract view of the mobile phone UI software architecture. The page consists of four substitution transitions (Applications, Servers, UIController and CommunicationKernel) corresponding to the four parts mentioned above. The phone element and the two adjacent places are used for visualisation and are not part of the Petri Net model of mobile phone UI software architecture system.

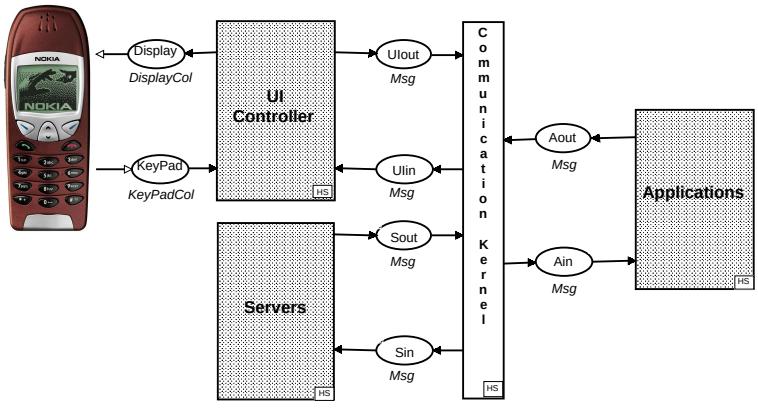


Fig. 2. Page Nokia.

4.2 Modelling the features

In the following we go into more detail on how the features of the mobile phone are modelled. We will use the game feature as an example. The game feature is chosen as an example because it is fairly simple and intuitively clear but still complex enough to illustrate the interesting aspects of the CPN model.

Figure 3 shows the page Game modelling the game feature. In Fig. 3 the places Idle, Selected, Playing, and Suspended all have the colour set `Application`, which denotes a cartesian product of an application and some internal data of the application, e.g., for the game feature whether there is a saved game. These four places model the possible states of the game feature. We will explain the rest of the places in Fig. 3 later.

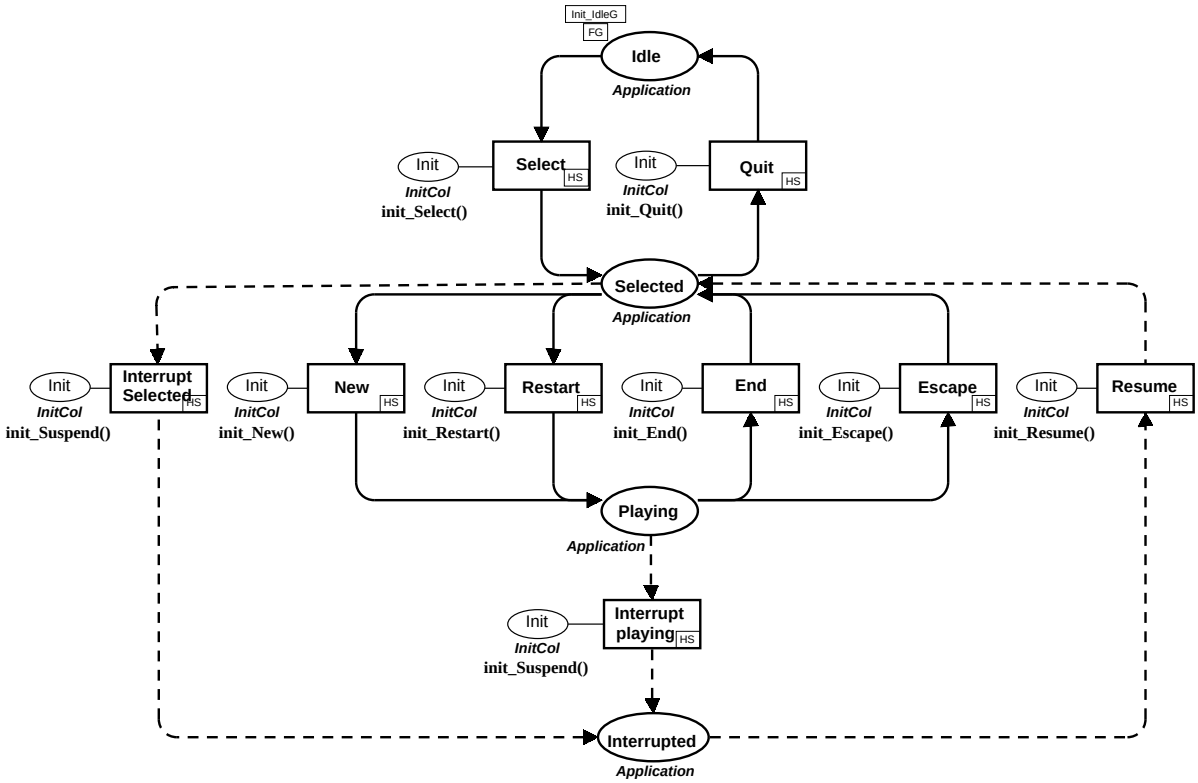


Fig. 3. Page Game.

Initially the game feature is Idle. The selection of a kind of game is modelled by the transition Select, which causes the game to change its state from being Idle to Selected. Transition Quit models the quitting of the game feature and causes the game to change its state from Selected to Idle. The transitions New and Restart model the start of a new game and restart of a previously saved game, respectively, and causes the game to change its state from Selected to Playing. The transitions End and Escape model the termination of a game and the save and escape from a game (which then can be resumed later), respectively, and causes the game to change its state from Playing to Selected.

While a game is Selected or Playing it can be suspended by other features in the system, e.g., an incoming call. This is modelled by the two transitions SuspendSelected and Suspend-

Playing which causes the game to change its state from Selected or Playing, respectively, to Suspended. A Suspended game can be resumed. This is modelled by the transition Resume which causes the game to change its state from Suspended to Selected. All the transitions in Fig. 3 are substitution transitions.

All state changes in the game feature, i.e., all substitution transitions in Fig. 3, correspond to the same overall pattern of behaviour based on sending a request to the UI controller and wait for an acknowledgement. The overall pattern corresponds to the items in the list below.

1. The state change is triggered by an incoming message to the feature, e.g., a message from the UI controller reporting some user action.
2. The feature requests the UI controller to put a user interface on the display (or if the feature already has a user interface shown on the display to update that user interface).
3. The feature waits for an acknowledgement from the UI controller.
4. When receiving the acknowledgement the feature completes its state change and changes its internal data accordingly, e.g., when restarting a previously saved game the game feature will change its data from `saved` to `in_progress`.

This overall pattern is modelled by the page Send_Receive shown in Fig. 4.

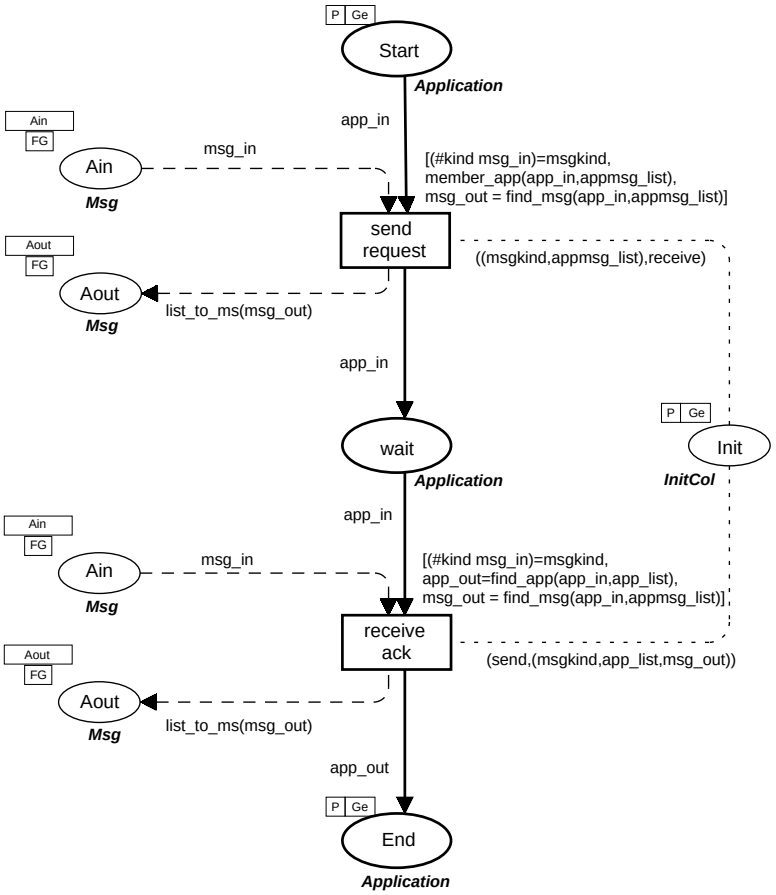


Fig. 4. Page Send_Receive.

The page Send_Receive contains three *port places*: Start, End and Init. Places Start and End are bound to the input place and the output place of the substitution transition, respectively. Place Init is bound to the place connected to the substitution transition with a line (a line corresponds to an arc with arrowheads in both directions). Places Ain and Aout model the input and output buffers between the communication kernel and the application (see Fig. 2). Ain (and Aout) are specified as fusion places which means that all the places occurring with name Ain (and Aout) are identical even though they are drawn as individual places. The two transitions SendRequest and ReceiveAck models the basic communication pattern listed above.

- SendRequest models items 1 and 2. The guard checks that `msg_in` is a message which triggers the state change (the first line in the guard), that the application is in a state where it is allowed to perform the state change (the second line in the guard) and finally the guard ensures that `msg_out` is bound to the messages sent (the third line in the guard).
- ReceiveAck models items 3 and 4. The guard checks that `msg_in` is an acknowledgement from the UI controller (the first line in the guard), that the application performs the state change (the second line in the guard) and finally the guard ensures that `msg_out` is bound to the messages sent (the third line in the guard).

The concrete information about which messages 'triggers' the event, which messages are sent, what change in data are performed etc. is read from the (initial) marking of the Init place. All the substitution transitions in the page Game shown in Fig. 3 are bound to instances of the page Send_Receive shown in Fig. 4 with individual instantiations of the corresponding Init places. For the sake of readability all the initial markings are specified by means of functions which all evaluate to constants.

All the features in the CPN model follow the same idea as the game feature. Hence, when adding a new feature a page modelling the overall state changes is provided together with a description of the display requests, use of servers or other applications and the internal data of the feature. This way of modelling the features reflects the way the features currently are described/documented in the UI specifications. Here the features are described using a textual description of the possible states and state changes of the feature together with a description of the user interfaces, use of other applications, and the reactions of the feature to user key presses. Hence, the CPN model of the features closely follows both the UI designers' and UI software developers' current understanding of features as well as the current available documentation of the features.

We have now presented the CPN model of the game feature to give an idea of the complexity of the CPN model and the efforts required to include a feature in the model. As can be seen from the above presentation the overall infrastructure of the mobile phone UI software system has been modelled and adding a new feature is relatively easy. More importantly is that new features can be added without changing the models of the existing features which makes it very easy to add or disable features in the future formal analysis.

We model the individual features of the mobile phone using the same ideas as described for the game feature. The feature interactions are captured in the CPN model as the communication and interaction between the individual features in the CPN model. In the first phase of the MAFIA project we have used simulations of the CPN model to detect and investigate interactions between the features in the CPN model. Using simulations we have obtained detailed knowledge about the feature interactions and identified the important patterns of interaction. Later phases of the project will include more formal analysis, e.g., state space analysis, of the CPN models.

5 Simulations of Scenarios

The UI designers and UI software developers who are developing new features will use the CPN models to identify and analyse the interaction patterns of their new features. One way of using the CPN models is by means of simulation; both interactively (step-by-step) for detailed investigation of the feature interactions and more automatically for investigation of larger scenarios. In this section we will present techniques which allow UI designers and UI software developers (who are not familiar with CP-nets) to control and gain information from simulations without interacting directly with the underlying CP-net and its token game. Section 5.1 presents two extensions to the CPN model providing information about simulations. Section 5.2 presents two extensions to the CPN model allowing the UI designers and UI software developers to control simulations.

5.1 Visualisation of Simulations

Two extensions are made to the CPN model allowing the current state of the CPN model and the behaviour of the CPN model during simulation to be visualised. Firstly, the state of the phone as the user observes it on the handset is visualised via an animation of the display. Secondly, the CPN model is extended with Message Sequence Charts (MSCs)[1] to be automatically constructed as graphical feedback from simulations. The reason MSCs are chosen to visualise the behaviour of the CPN model is that diagrams very close to MSCs are already in use in the design process at Nokia. MSCs therefore allow the behaviour of the CPN model to be visualised in a way that is familiar to the UI designers and UI software developers.

The state of the phone as the user observes it on the handset is visualised via an animation of the display. Figure 5 shows a snapshot of the animation taken during a simulation of the CPN model. The snapshot shown corresponds to a state where the game feature is Selected (a kind of game has been selected) and has a user interface on the display. The user now has the possibility to either start a new game (transition New in Fig. 3), restart an old game (transition Restart in Fig. 3) or set the level of the game (not included in the CPN model). The animation is implemented using the Mimic library [8] of Design/CPN which allows graphical objects to be displayed and updated during simulations (using the code segments associated with transitions in Design/CPN). We have added a code segment to the transition in the UI controller modelling the update of the display, i.e., when the transition occurs the corresponding code segment is executed causing the animation to be updated. Hence, the animation reflects the contents of the display during simulation.

The animation of the display will provide information about the state of the CPN model as the user of the mobile phone will observe it. However, the UI software developer is also interested in gaining information about the UI software system at a more detailed level. This is obtained through the use of MSCs which capture the communication between applications, servers and the UI controller in the mobile phone UI software architecture.

Figure 6 shows an example of a MSC automatically generated from a simulation of the CPN model. The MSC contains a vertical line for each of the relevant applications and servers in the phone UI software system and a vertical line for the UI controller and the user of the handset. The arrows between the vertical lines correspond to messages sent in the system. The communication sequence considered corresponds to a scenario where the mobile phone receives an incoming call while the user is playing a game (an interaction between the game

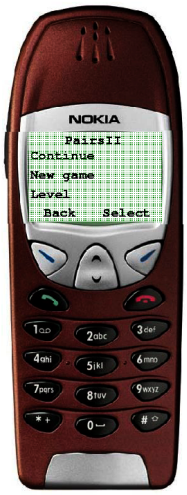


Fig. 5. Animation of the display using the Mimic library of Design/CPN.

and call features). The scenario causes the following sequence of events to occur. The numbers in the list below correspond to the line numbers found in the MSC.

- The user selects a kind of game from the menu (line 1). The game feature is notified and it requests the display (lines 2-3).
- The user selects to start a new game (line 4). The game feature is notified and it changes the contents of the display accordingly (lines 5-6).
- An incoming call arrives. The call server notifies the call feature (lines 7) which requests the display (line 8).
- The display is currently used by the game feature. The UI server interrupts the game feature (line 9) and after the interruption has been acknowledged the display is granted to the call feature (lines 10-11).
- The user rejects the call (line 12). The call feature is notified and the display is removed (lines 13-14).
- The game feature is resumed (lines 15-16).

The above scenario is an example of a feature interaction between the game and the call features in the mobile phone UI software system. The interaction is caused by the need for the game and incoming call features to share the display of the mobile phone and the interaction therefore falls into category II of the categorization given in Sect. 3.

Note that in the scenario the UI controller is responsible for handling the interrupt (lines 9-11) and resume (lines 15-16) of features. Hence, the features do not have to know which features they potentially interrupt or are interrupted by. This makes it very easy to add and remove features from the CPN model without changing the subnets modelling the rest of the features. The CPN model presented here (where the UI controller is in charge of the interrupt and resume of features) is not a model of the current implementation of the UI controller but an improved design suggested as a result of the modelling activity in the MAFIA project. We will return to this in Sect. 6.

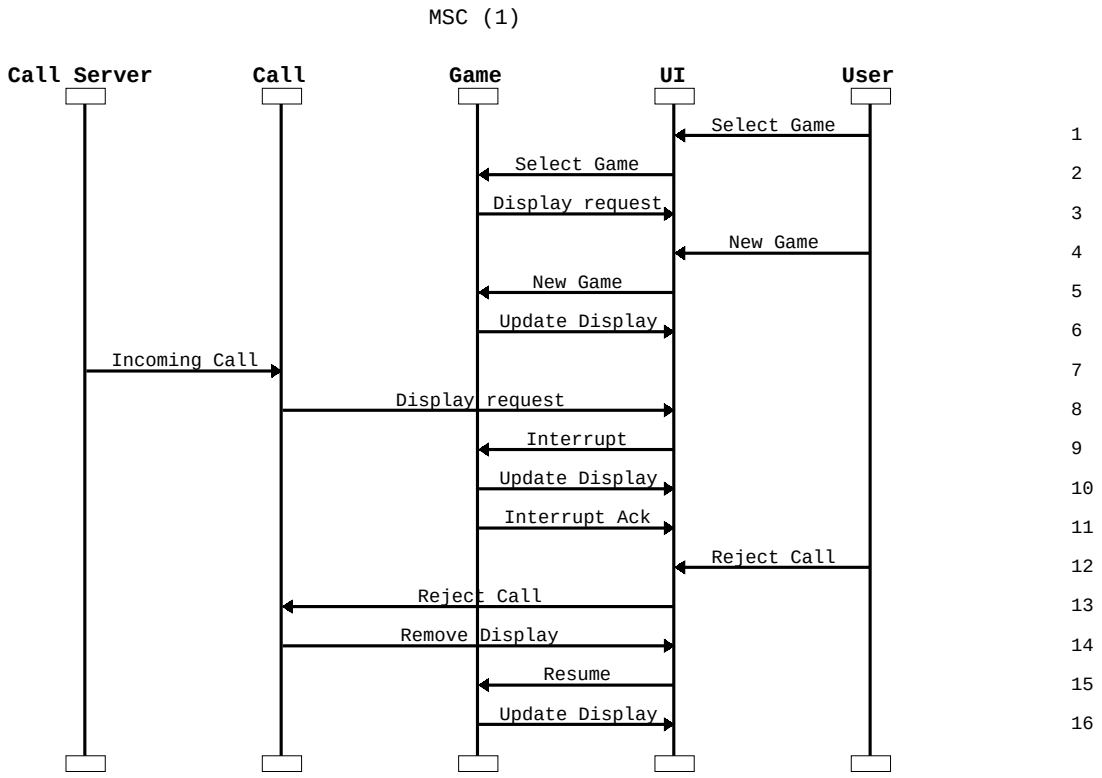


Fig. 6. A MSC automatically generated from a simulation of the CPN model.

5.2 Controlling the Simulation

Two extensions have been made to the CPN model to make it possible to control the simulations without directly interacting with the underlying CP-nets. The first extension makes it possible to control simulations interacting by means of key presses with the mouse on the picture of the mobile phone in Fig. 5 again using the Mimic library of Design/CPN. Different regions has been associated with each of the keys in the picture of the mobile phone. From the code segment of the transition in the UI controller modelling keypresses from the user of the handset we call a ML function which reads which region (i.e., key) has been selected.

The second extension makes it possible to set up a scenario to be simulated. The scenario can be specified as a series of events (together with an ordering between the events). The left part of Fig. 7 shows how the scenario corresponding to the MSC in Fig. 6 (where the mobile phone receives an incoming call while the user is playing a game) is specified. In this way it is possible to inspect interesting scenarios without manually pressing the keys of the mobile phone in Fig. 7.

The right part of Fig. 7 shows how a collection of scenarios can be specified as a partial order of events, i.e., only a partial order is given between the events. This example covers 60 different orderings of the six events. We expect this way of specifying (collections of) scenarios will be very useful in later phases of the MAFIA project where we will concentrate on formal analysis of feature interactions and investigate the use of CPN models in the planning of test cases in the feature development. The possible orderings of the events in the partial order correspond to test cases used when testing features and feature interactions in UI software development of Nokia mobile phones. From the 60 possible orderings of the six events in

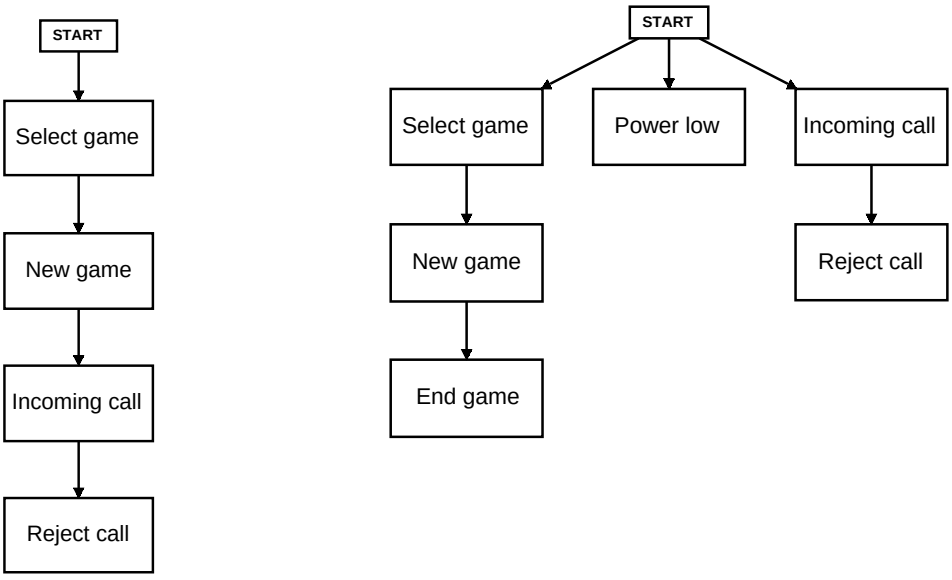


Fig. 7. Guided simulations of the CP-net.

the example only 24 orderings are valid (according to description of the features in the UI specifications the incoming call has to be rejected before the game can proceed). A way to investigate which orderings can actually occur in the CPN model is to combine the use of these partial orderings together with state space analysis, construct the part of the state space corresponding to events and orderings of events allowed by the specification given via the partial ordering can be constructed. In this way a partial state space capturing the possible feature interactions caused by a sub-set of the events for the features in the mobile phone is obtained.

In the first phase of the MAFIA project we have only investigated the possibility of extending the CPN model with such guided simulations. The precise definition, e.g., the language to describe it and evaluation of the applicability will be evaluated in later phases of the project.

6 Conclusions and Future Work

Modern mobile phones provide a wide range of features to the user via the user interface. Detecting and describing these feature interactions is a complicated and cumbersome task and currently the most complex interactions are not fully understood before the features are implemented. In this paper we have presented first results of the MAFIA project in which important feature interaction patterns in Nokia mobile phones is modelled using Coloured Petri Nets.

The current CPN model serves as a framework and provides the basic UI infrastructure where we can plug in features. The modelling of the UI controller is essential in obtaining a CPN model where features can be added and removed without changing the subnets of the rest of the features in the CPN model. The current CPN model now includes a few important features and we have already identified some basic interaction patterns.

We have presented the modeling approach to our internal customers and they have accepted the general idea. We are now adding more features to the model to build a comprehen-

sive set of interaction patterns. One important task will be to link the interaction patterns to existing implementation patterns.

An aspect of the MAFIA project is to provide support and generate ideas for the mobile phone UI software architecture development work. The construction and simulation of the CPN model has successfully identified some shortcomings of the current design. Also demonstrations of the current CPN model to our internal customers have generated discussions and ideas for possible improvements in the mobile phone UI software architecture to obtain better support for feature interactions in the UI software development. The identified shortcomings are

- The implemented design of the UI controller is based on a strategy where features need to know which other features coexist. Also the priority of possible events needs to be known when accessing the display of the mobile phone. We have suggested an idea for new design of the UI controller where the UI controller implements the protocol for accessing the display of the phone. Hence, the UI controller is in charge of interrupting and resuming features. This is the design presented in the CPN model in Sect. 4.
- Ambiguities in the specification of caller groups and ringing tones which, as a consequence, can lead to a situation where the phone does not play a ringing tone when the mobile phone has an incoming call.
- Inconsistencies and unnecessary communication in the system when handling menus on the display. A simple change which reduce communication between the UI controller and the server handling the menus.

Future work in the MAFIA project will focus on the detection of feature interactions in the CPN model. We will evaluate the usefulness of CP-nets for automatic detection of feature interactions and compare our technique to other techniques currently in use in the field of feature interactions, e.g., [7, 6]. We are planning to include state space analysis, investigate the possibility for using the CPN models to support the planning of test cases, and we will further develop and evaluate the idea of guided simulations.

References

1. ITU (CCITT). Recommendation Z.120: MSC. Technical report, International Telecommunication Union, 1992.
2. Examples of Industrial Use of CP-nets and Design/CPN. Available from `http://www.daimi.au.dk/CPnets/intro/examples_indu.html`.
3. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
4. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods.* Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
5. K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN Reference Manual.* Department of Computer Science, University of Aarhus, Denmark, 1995.
   Online: `http://www.daimi.au.dk/designCPN/`.
6. M. Nakamura, Y. Kakuda, and T. Kikuno. Analizing non-determinism in telecommunication services using p-invariant of petri-net model. In *Proceedings of the 16th IEEE Conference on Computer Communications (INFOCOM'97)*, pages 1253–1260, April 1997.
7. K.P. Pomakis and J.M. Atlee. Reachability Analysis of Feature Interactions: A Progress Report. Technical report, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1995.
8. J. L. Rasmussen and M. Singh. *Mimic/CPN. A Graphical Simulation Utility for Design/CPN. User's Manual.* Available from `http://www.daimi.au.dk/designCPN/`.

Analysis and Simulation of Steady States in Metabolic Pathways with Petri Nets

Monika Heiner
Computer Science Department
Brandenburg University of Technology at Cottbus
D-03013 Cottbus

Ina Koch
Department of Bioinformatics
Max-Planck-Institute for Molecular Genetics
D-14195 Berlin-Dahlem

Klaus Voss
SCAI – Institute for Algorithms and Scientific Computing
GMD – German National Research Center for Information Technology
D-53754 Sankt Augustin

Abstract

Computer assisted analysis and simulation of biochemical pathways can improve the understanding of the structure and the dynamics of these systems considerably. The construction and *quantitative* analysis of kinetic models is often impeded by the lack of reliable data. However, as the topological structure of biochemical systems can be regarded to remain constant in time, a *qualitative* analysis of a pathway model was shown to be quite promising as it can render a lot of useful knowledge, e.g., about its structural invariants. This paper deals with pathways whose substances have reached a dynamic concentration equilibrium (steady state). It is argued that appreciated tools from biochemistry and also low-level Petri nets can yield only part of the desired results, whereas executable high-level net models lead to a number of valuable additional insights by combining symbolic analysis and simulation.

1 Introduction

With the rapidly growing amount of new experimental data, the modelling of biological pathways occuring in the cell regained great interest. For this challenge in biosciences, biologists need theoretical methods and computational tools in order to prove, analyse, compare, and simulate these complex networks for different organisms and tissues. The results are of major importance also for the biotechnology and the pharmaceutical industry.

"The main focus in the mathematical modelling in biochemistry has traditionally been on the construction of *kinetic* models. The aim of these models is to predict the system dynamics" [HeSc98]. Their analysis is commonly based on the solution of systems of differential equations. In this way, numerous kinetic models for different metabolic systems and membrane transport processes have been developped (for a review see [HeSc96]). A severe restriction, often encountered in the construction of these models, is the imperfect knowledge of the kinetic parameters.

On the other hand, a *structural* analysis of metabolic pathways mainly deals with the topology of how substances are linked by reactions. A central role is played by stoichiometric matrices, which indicate how many molecules of each substance are consumed or produced in the single reactions. Their analysis is based on the solution of algebraic equations, and is independent of any kinetic parameter. Of particular interest are biochemical systems persisting in a *steady state* (see section 2), i.e., in which the concentrations of their substances have reached an equilibrium. An *elementary mode* (this term has been coined in [ScHi94]) can be regarded as a minimal set of reactions (resp. of the enzymes catalyzing them) that can operate at steady state. Knowledge about the flux rates and the elementary modes of a system allows "to define and comprehensively describe all metabolic routes that are both stoichiometrically and thermodynamically feasible in a given group of enzymes" [SFD00].

A metabolic system can be modelled as a Petri net in a straightforward way, as has been demonstrated for low-level nets in [RML93, Hof94] and for high-level nets in [GKV00]. The Petri net structure then truly reflects the biochemical topology, and the incidence matrix of the net is identical to the stoichiometric matrix of the modelled metabolic system. Accordingly, the mentioned structural invariants and elementary modes correspond almost directly to the T-invariants known from the Petri net theory. An actual account of the structural analysis of metabolic networks and the analogy to Petri nets is given in [SPM00].

The use of Petri nets for modelling quantitative (kinetic) properties of biochemical networks, especially for genetic and cell communication processes, was discussed in [Hof94, HoTh98]. Other contributions followed, using various types of Petri nets like stochastic nets [GoPe98, GoPe99] and hybrid nets [MDNM00]. Executable high-level net models of metabolic pathways, and their (almost automated) construction, simulation, and quantitative analysis are described in [GKV00].

The application of Petri nets to this field began in the nineties with the publications of Reddy *et al.* [RML93, RLM96]. They present a low-level (place/transition) net to model the structure of the combined glycolytic pathway (GP) and pentose phosphate pathway (PPP) of erythrocytes. They use the well-known algebraic methods to compute S- and T-invariants from the incidence matrix of the net. A thorough analysis of an extended form of this pathway was performed by Koch *et al.* [KSH00], which forms the starting point for this paper.

For computing conservation relations (S-invariants) and elementary modes (T-invariants) of metabolic pathways, the software package METATOOL [Pfe99] has been developped (by biochemists) and successfully applied in a number of cases. However, merely the integer weighted S-invariants are detected. Moreover, only the overall reaction equations, i.e., the net effects of a pathway execution, can be computed, and any consideration of its dynamics, in particular of the partial order of the reaction occurrences, is missing.

The main achievements reported in this paper rely on the use of executable high-level net models and symbolic analysis. This allows to consider the following crucial aspects:

(a) It is well known that the detection and interpretation of invariants can substantially improve the understanding of systems. In the context of certain problems, however, the most interesting system properties are *not* invariant. In these cases, very often the divergence of these structures from an invariant is of major importance as it indicates a *defect* or *effect* of the substructure in question. We shall introduce and formally define these concepts for high-level Petri nets in section 2 and use them extensively, in section 4, for the symbolic structural analysis of the quite complex sample pathway presented in section 3.

(b) In coloured nets, the model designer can distinguish tokens via their colours. This

is a prerequisite for overcoming the restrictions of low-level nets or METATOOL, i.e., for both detecting a hitherto unknown S-invariant of our model and determining its partial order dynamics, as shown in section 4.

The software tool of our choice – for graphical editing, analyzing and executing the net models – is Design/CPN [Design].

2 Steady State Pathways, Elementary Modes

First some Petri net notions are recalled that we want to apply to metabolic pathways later on. The algebraic analysis of Petri nets mainly relies on the notion of invariants. However, in a great number of systems, like metabolic pathways, variable substructures (with non-null defects or effects) deserve even more attention. Hence the following definitions (see [Gen01]) will turn out to be useful.

Let $\mathcal{N}$ be a coloured Petri net with the sets S of places and T of transitions. The *incidence matrix* C of $\mathcal{N}$ is an $|S|\times|T|$ matrix whose elements c_{ij} are the positive (negative) labels of the arcs pointing from (to) transition t_j to (from) place s_i. A *distribution* is a mapping transforming the elements (colours) of a colour set D into linear combinations of elements of a not necessarily different colour set D'. Now let y ($\geq O$, the zero vector) be an S-vector such that, for every place s, the component y_s is a combination of distributions of s, and all the y_s have the same range. The transpose matrix C^T can be multiplied by the vector (one column matrix) y. An example is the S-vector σ' in section 4.1.

The product $C^T \cdot y$ is called the *defect* of y.

A (column) S-vector $y \geq O$ is called an *S-invariant* of $\mathcal{N}$ iff $C^T \cdot y = O$.

An S-invariant represents a *state quantity* of the net system, i.e., a quantity which, starting from an initial state, is maintained during the whole life time of the system. It describes a *conservation rule*, as known from many areas in (natural) science. Among others, such a mandatory S-invariant can be a valuable means to detect inconsistencies of a system specification or model. An example is the S-vector σ in section 4.1.

A *process* π is a partially ordered set of transition occurrences leading from a state M_1 to a state M_2, $M_1 \xrightarrow{\pi} M_2$. Ignoring the order of occurrences yields a T-vector x ($\geq O$) of combinations of transition occurences which is called the *action* performed by π. Note that, in each element of x, all variables have to be *substituted* by colours of the same colour set. This corresponds to the bindings of the variables around a transition which determine the particular kind of its occurrence. An example is the T-vector τ' in section 4.2.

The state difference $\Delta M = M_1 - M_2 = C \cdot x$ is called the *effect* of x.

A (column) T-vector $x \geq O$ is called a *T-invariant* of $\mathcal{N}$ iff $C \cdot x = O$.

A process $M_1 \xrightarrow{\pi} M_2$ performing a T-invariant leads from one state to the same state again ($M_1 = M_2$), it *re-generates* the state M_1. An example is the T-invariant τ in section 4.2.

Next we shall introduce those notions from biochemistry that allow to characterize the kind of metabolic pathways considered in this paper and to discuss the possibilities of applying the Petri net theory to them.

A metabolic network consists of highly integrated chemical reactions. More than a thousand chemical reactions take place even in such a simple organism as the bacteria *Escherichia coli*. The chemical compounds in these reactions are called *substrates* or *metabolites*. An *enzymatic reaction* is catalyzed by a reaction-characteristic *enzyme*. These enzymes are needed for initializing the reaction and do not change their structures during the reactions, whereas the substrates are converted into each other, thus changing their concentrations. A great number of reactions can be treated as *irreversible* under normal

conditions. That means, such a reaction permanently occurs in a preferred direction, consuming those substances called *reactants* and producing its *products*. In principle, however, each enzymatic reaction is *reversible*. For given reactant concentrations, the higher the concentrations of the products get, the slower the reaction will occur in the preferred direction, and at a certain value of those concentrations the reaction will change its direction, i.e., will consume the products and produce the reactants.

As mentioned in the introduction, the approach described in this paper concentrates on the mere structure of the pathways, i.e., on the topology of the interconnections of metabolites via enzymatic reactions. Hence, it is *structural* or *qualitative* as it does not deal with the kinetic details of the reactions. Constructing a Petri net of a metabolic pathway is straightforward, representing metabolites as places, reactions as transitions and the stoichiometric relations by labelled directed arcs between them. Examples can be found in [RML93, Hof94, RLM96, KSH00] (low-level nets), or in [GKV00] and section 3 of this paper (high-level nets). In the following, such a net is called *the* Petri net model of the pathway.

In our context, a distinction is made between *external* and *internal* metabolites according to whether or not they are involved in reactions outside the system considered. External metabolites are called *sources* resp. *sinks* of the pathway if they are produced resp. consumed by those (external) reactions. A metabolic pathway is said to persist in a *steady state* if the concentrations of all internal substances have reached a dynamic equilibrium: for each internal metabolite, the total rate of its consumption equals that of its production. Assuming a constant activity of all enzymes involved in the system, many (but not all) metabolic pathways reach such a dynamic equilibrium after some time.[1] Structural analysis of metabolic systems in steady state aims at, among others, "elucidating relevant relationships among system variables" [HeSc98] and does not rely on imperfectly known or doubtful kinetic data.

A formalization of steady state and related notions is given in [SHWF96]. For our paper, we need the following. The *stoichiometric matrix* N of a metabolic pathway with n metabolites and r reactions, is an $n \times r$ matrix where the element N_{ij} denotes the flux from the i-th metabolite to the j-th reaction, i.e., the amount $\delta c_i/\delta t$ of the metabolite concentration produced or consumed by that reaction. Obviously, the stoichiometric matrix of a pathway precisely corresponds to the incidence matrix of its low-level net model. A metabolic pathway is in *steady state* if and only if the reaction rates fulfil the condition

$$\delta c_i/\delta t = \Sigma_{j=1}^{r}\, N_{ij}\, \nu_j = 0,\ i = 1, ..., n, \quad \text{or, in matrix notation,} \quad N \cdot \nu = O,$$

for an integer vector $\nu = (\nu_1, ..., \nu_r)^T$, called *flux vector*, where a component ν_j is an integer weight factor of the j-th reaction.

In biochemistry, 'flux modes' constitute the core concept of the algebraic analysis of steady state metabolic pathways. "An *elementary flux mode* is a minimal set of enzymes that could operate at steady state, with the enzymes weighted by the relative flux they carry. 'Minimal' means that if only the enzymes belonging to this set were operating, complete inhibition of one of these enzymes would lead to cessation of any steady-flux in the system" [SFD00]. Before relating biochemical analysis methods to the corresponding Petri net algorithms, two particular questions have to be discussed.

Firstly, in steady state analysis, those processes are of particular interest that start with the source substances of the investigated pathway and finish with its sink substances. For

[1] That and how this happens, has been demonstrated for glycolysis, gluconeogenesis, citric acid cycle (TCA), and combinations of them in [GKV00], by simulation runs of quantitative (i.e., including kinetic reaction functions) high-level Petri net models.

these external metabolites, constant concentrations have to be assumed to reach a steady state. Using METATOOL, this is achieved by excluding external metabolites from the stoichiometry matrix, but including the reactions affecting them: hence, their concentrations remain unchanged. In contrast, in our net models, we include the external substances and introduce an extra transition *StartEnd* that closes the pathway to a cycle by supplying the initially needed substrates and consuming the finally produced ones. This measure enables us to also identify those *internal* metabolites requiring initial markings and to compute their amounts.

Secondly, we have to take into account the reversibility of reactions. An obvious solution is to admit negative factors in the T-vector to denote the occurrences of reversible reaction transitions in the backward direction. This would lead to T-vectors $x \not\geq O$, not satisfying the standard definition of T-invariants for Petri nets. However, instead of deviating from this definition, we can – with (almost) the same result – introduce, for every reversible transition t, an additional complementary (reverse) transition t' to the net. This t' is then connected to the same set of neighbour places as t, but with all arcs pointing in the opposite direction. Doing that, we introduce, for each reversible reaction t, a potentially endless loop $(t, t', t, t', ...)$ which is biochemically meaningless. This slight disadvantage can be turned into an advantage in higher-level nets where we can and shall discriminate the directions of certain reactions according to the flux modes to which they belong.

3 Models of the Glycolysis and Pentose Phosphate Pathway

Any cell extracts energy from its environment and converts foodstuffs into cellular components by its metabolism. One fundamental question in biology is, how do cells extract and store energy from their environment. The main pathway for the generation of metabolic energy is the *glycolysis*. The glycolysis pathway (GP) is a sequence of reactions that converts glucose into pyruvate with the concomitant production of a relatively small amount of ATP. Then, pyruvate can be converted into lactate. The version chosen for this paper is that one for red blood cells, see [Str96]. In the Petri net $\mathcal{P}$ (Figure 1), the GP consists of the reactions $l1$ to $l8$.

The *pentose phosphate pathway* (PPP), also called *hexose monophosphate pathway*, again starts with glucose and produces NADPH and ribose-5-phosphate (R5P) which then is transformed into glyceraldehyde-3-phosphate (GAP) and fructose-6-phosphate (F6P) and thus flows into the GP. In Figure 1, the PPP consists of the reactions $l1$, $m1$ to $m3$, $r1$ to $r5$, and $l3$ to $l8$. From now on we use acronyms for most metabolic substances; their full names are listed in the Abbreviations appendix.

Whereas the GP generates primarily ATP with glucose as a fuel, the PPP generates NADPH, which serves as electron donor for biosyntheses in cells. The interplay of the glycolytic and pentose phosphate pathway enables the levels of NADPH, ATP, and building blocks for biosyntheses, such as R5P and Pyr, to be continously adjusted to meet cellular needs. This interplay is quite complex, even in its somewhat simplified version that shall be discussed in this paper. In [RLM96], this pathway has already been modelled as a low-level Petri net (place/transition net) and then – qualitatively – analyzed by means of the well known linear algebraic methods. The analysis in [RLM96] is not completely correct, and it yields neither a *full* S-invariant (i.e., comprising all substances from the sources to the sinks) nor a non-trivial T-invariant. Hence it reveals some deficiencies that we will overcome by switching to high-level (coloured) net models. Additionally, our models allow not only to be analyzed but also to be executed (simulated).

It shall be noted that molecules like ADP, ATP, NAD^+, P_i, and NADH play a somewhat special role in metabolic networks. They are called *ubiquitous* because they are found in sufficiently large amounts in almost all organisms. For ease of distinction, the remaining substances from Gluc to Lac shall be named *primary* in this paper. When talking about reactions in the following, the involved ubiquitous molecules commonly are not mentioned because the primary substances are those that characterize the reactions and hence are of particular interest.

The construction of the low-level Petri net starts with a modest simplification of the original pathway model. The left branch of Figure 1 represents the GP. Its second part is a strictly linear path, transforming BPS into Lac via 3PG, 2PG, PEP, and Pyr (depicted in the small box at the lower right hand of Figure 1). This path can and shall be reduced to just one super-transition $l8$ (with the appropriate connections to ADP, ATP, NADH, and NAD^+) without altering the crucial properties of the net. In Design/CPN, such a node is called a *substitution transition*: $l8$ stands for and equivalently represents the mentioned linear path. With this modification and disregarding for a moment the guards and replacing all arc labels by '1', we get the place/transition net $\mathcal{P}$ of Figure 1, which is identical to the net found in [RLM96].

In real organisms, the amount of the molecules is high enough to tolerate transient deviations from the theoretically postulated equilibrium concentrations. In the long run, a steady state is approximated: according to the kinetic equations, those reactions with too high reactant (resp. too low product) concentrations are preferred to those with too low (resp. too high) ones. When qualitatively analyzing metabolic models, however, one is confronted with very few, even minimal numbers of molecules for which the desired invariant properties have to be proven. Moreover, molecules of the same substance are chemically undistinguishable although their 'roles' (weight factors in the invariant vectors) may differ depending on the reaction environment in which they appear. As will be shown in section 4, some invariant properties cannot be proven, and a proper simulation cannot be performed as long as different roles of molecules cannot be respected.

To be more specific, the crucial point of using coloured instead of low-level Petri nets is the following.[2] Applying higher-level places allows to discriminate between different molecules of the same metabolite via their identifiers (*colours*) C, D, F, ...[3] This enables the designer to separate different branches of the compound pathway and to distinguish among molecules on the same place according to their origin and destination reaction. Although this distinction is not motivated by the biochemical reality (where molecules of the same substrate are identical), it increases the potential of the qualitative analysis we have in mind (see chapter 2). Moreover, it is a prerequisite for executing such models properly, e.g., without running into unexpected deadlocks or the like. We have simulated all models in this paper, clearly not to get new results about their kinetics but mainly to

[2] Legend for the Design/CPN nets in this paper:
– All places have the colourset $CS = \{C,D,F,G,H,G',H'\}$.
– The underlined inscription 1`D inside the place *StartEnd* denotes its initial marking.
– A term in brackets [ ] is a *guard* (boolean expression) of the transition. If the value of the guard is *true* the transition may be enabled, if *false* it cannot.
– A place name in italics denotes a *fusion place*. All members of a fusion set are treated as the same place. Their names are numbered consecutively.
– The places for the substrates NAD^+, $NADP^+$, P_i are named NADp, NADPp, Pi, respectively.
– A (dashed) transition t' denotes the reverse counterpart of the reversible reaction t.

[3] Note that the (theoretical) distinction by colours applies to chemically identical molecules, e.g., a token C on G6P is distinguished from a token G on G6P. On the other hand, the substance which a molecule represents is unambiguously determined by the place it belongs to. Hence, a token C on G6P represents a different substance than a C on F6P.

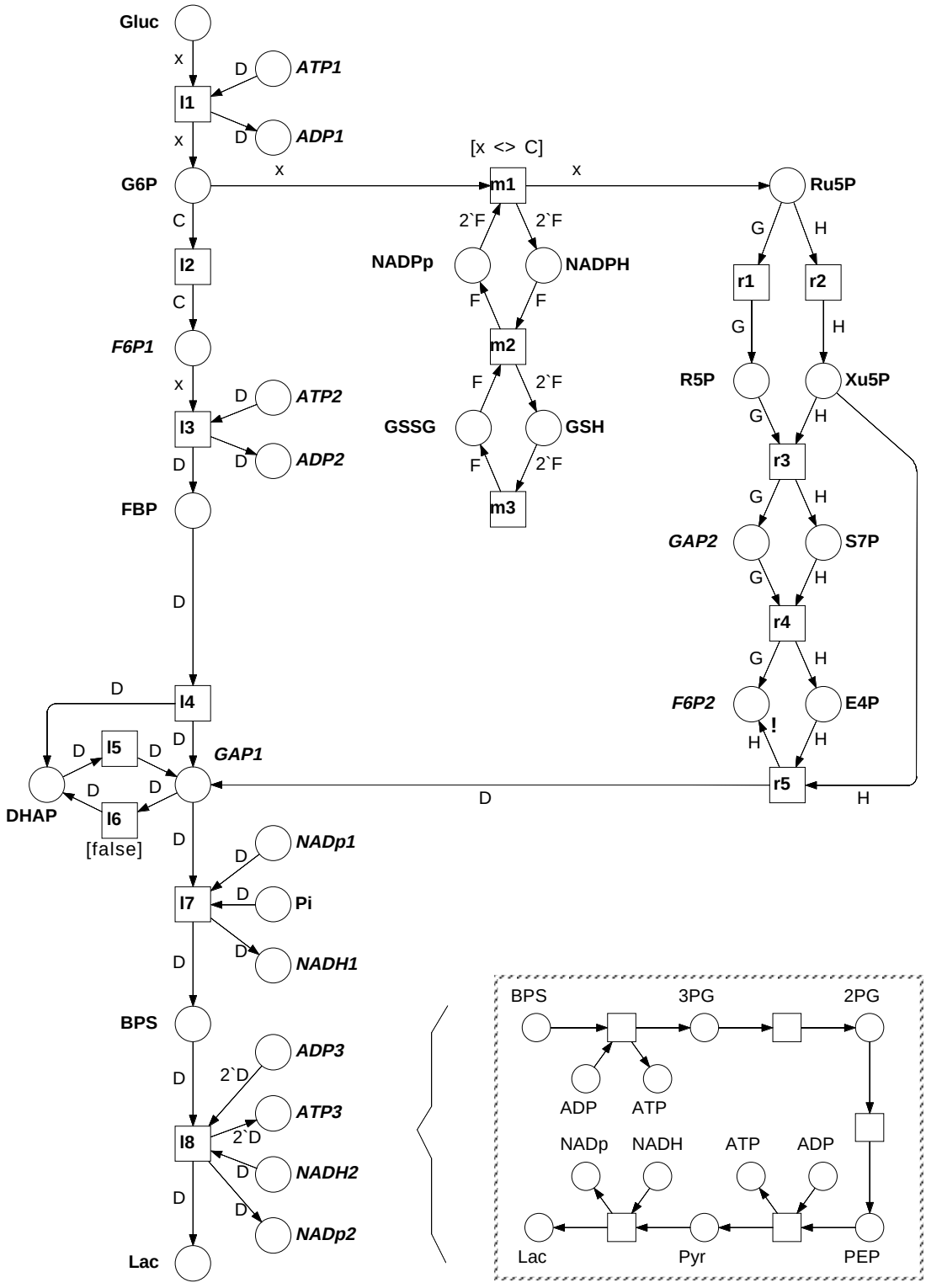


Figure 1: The original Petri net model $\mathcal{P}$ of glycolysis/pentose phosphate pathway

gain confidence in the chosen colour specifications.

What is the strategy of attributing colours to the tokens (molecules) along a given pathway model? Starting with a primary source substance of the pathway, we look for *conflicts* on the way to the sink(s). By definition, p is a conflict place if it has more than

one output transition, and all are enabled if p carries a suitable token. If one of these alternative transitions occurs, all remaining transitions are disabled. In our context, a conflict would cause no harm as long as all but one alternative paths starting at p would end up again at this p without any lasting marking change. In general however, this is not the case. When looking at the metabolism in one specific organism, alternative metabolic paths most often result in different metabolic overall reactions. Therefore, they shall be discriminated and must not be combined deliberately. This discrimination is performed by attributing different identifiers to the molecules and by additionally blocking certain transitions for particular molecules (using guards).

This shall be demonstrated by use of the sample net $\mathcal{P}$, treated again as a low-level net by disregarding all arc labels and guards. Starting with the source Gluc, the first conflict is encountered at G6P which can be the reactant of either the reaction $l2$ or $m1$. A G6P-molecule with destination $l2$ gets a colour, say C, and that one with destination $m1$ gets a different colour (to be decided upon later). The guard [$x \neq$C] prevents a C-token to be consumed by $m1$. Proceeding downwards the GP, at the left-hand side of the figure, we examine F6P1, a fusion place. As F6P2, on the right-hand side, has no outgoing arc, a conflict does not exist. The next conflict on the way down is found at GAP (the fusioned GAP1 and GAP2), a conflict among the three transitions $l6$, $l7$ and $r4$. The reaction $l6$ is trivial, and can be disabled by the guard [false], as the loop GAP1 $\rightarrow l6 \rightarrow$ DHAP $\rightarrow l5 \rightarrow$ GAP1 returns the token to GAP1 without affecting any other places. The conflict between $l7$ and $r4$ is difficult to discuss at this moment without knowledge about the situation in the PPP at GAP2. We postpone it to the end of this paragraph. Instead, the path from G6P into the PPP, in the middle and right part of the figure, shall be inspected. Choosing a separate colour F for the molecules of the middle part is not mandatory because there is no conflict here; it is just a matter of taste. The next conflict occurs at Ru5P with the choice to continue via $r1$ or $r2$. The colours of these two molecules must be different from each other and from C; we choose G and H. The last conflict at GAP is the postponed one. However, because the colour G has been maintained from Ru5P via R5P until GAP2 and the tokens on GAP1 have the (different) colour D, their distinction is accomplished already: the G-molecules are removed by reaction $r4$ and the D-molecules by $l7$. The resulting model is again $\mathcal{P}$, but now regarded as a coloured net by including the arc labels and transition guards of Figure 1.

As mentioned in section 2, in case of a reversible transition t, we add its complementary transition t' to the net. In case of the sample net model $\mathcal{P}$ we know that the reactions $l1$ and $l3$ are irreversible. The linear path from BPS to Lac, replaced by the substitution transition $l8$, contains at least one irreversible reaction, namely from PEP to Pyr; thus $l8$ should also be treated as irreversible. Hence we introduce the new complementary transitions $l2', l4', l5', l7'$, and $r1'$ to $r5'$ to the net $\mathcal{P}$ and thus obtain the model $\mathcal{P}_{rev}$ in Figure 2. Note that the transition $l6$ in the model $\mathcal{P}$ is identical to $l5'$ in $\mathcal{P}_{rev}$. Thus $l6$ in the former net model is just replaced by $l5'$ in the latter.

$\mathcal{P}_{rev}$ shall be used to compute the S-invariants (subsection 4.1).

The introduction of the reverse transitions in $\mathcal{P}_{rev}$ may entail additional conflicts which have to be resolved when dealing with T-vectors and simulation. We observe:
– $l4', l5', l7'$, and $r1'$ to $r5'$ merely create uncritical loops and can be deleted,
– $l2'$ must not appear in the GP. Hence, $l2'$ must be prevented from occurring for tokens $x = C$. In the course of PPP, one G- and two H-molecules on G6P are transformed into one H-token on GAP1 (via $r5$) and two H-tokens on F6P2 (via $r4$ and $r5$). For the latter H-tokens there are three possibilities to be processed further:
(1) both move to FBP via $l3$ and continue on the 'normal' way to Lac,

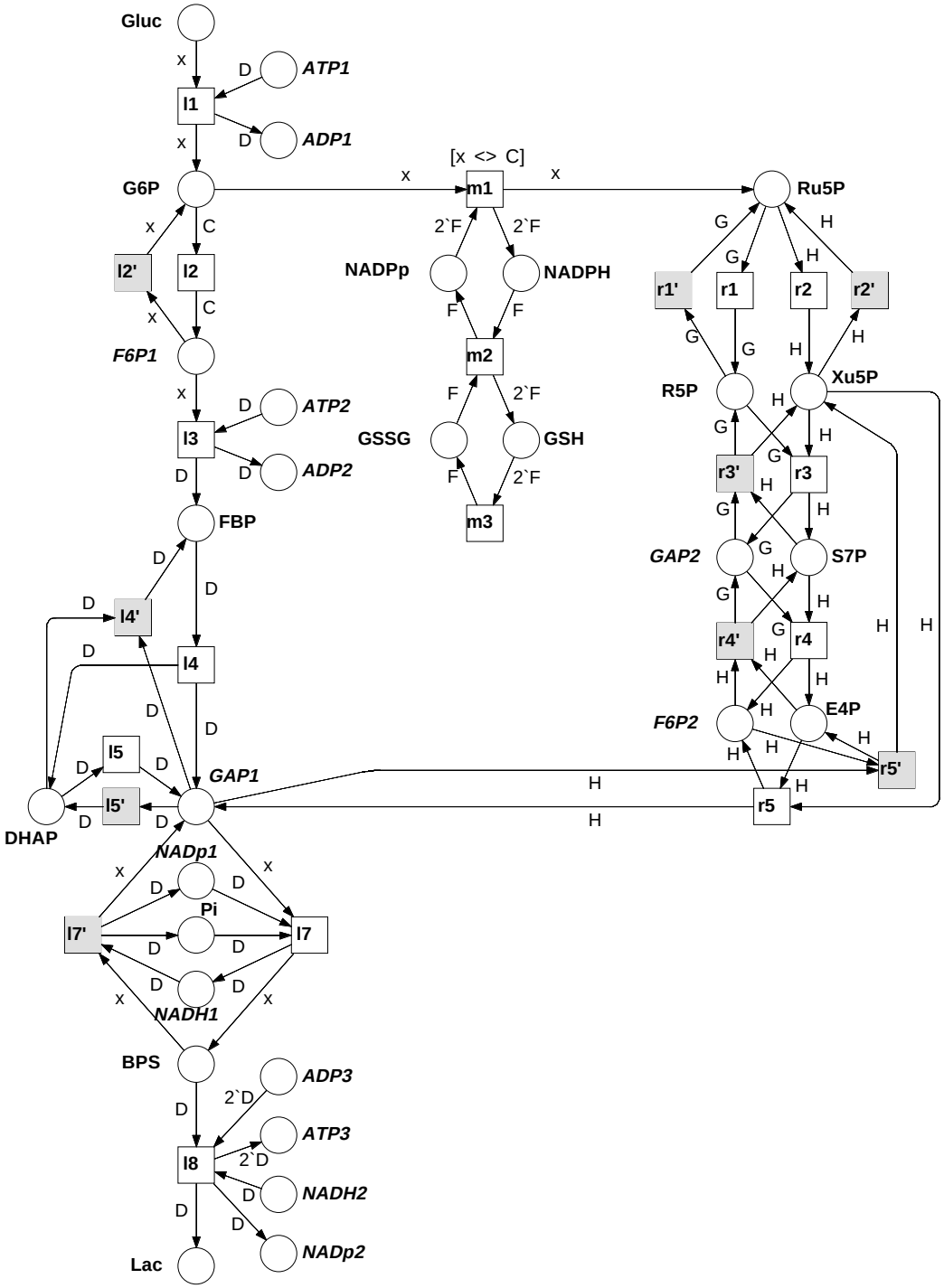


Figure 2: The Petri net $\mathcal{P}_{rev}$ with reversible reactions

(2) both move to G6P via $l2'$ and thus regenerate the initial two H-tokens there,
(3) one of them is moved by $l3$ to FBP, and the second one by $l2'$ to G6P. This case is a combination of glycolysis and *gluconeogenesis*, which cannot occur in steady state.

To distinguish the 'normal' PPP path (1) from the new reaction path (2) we have to introduce new token-colours, G$'$ and H$'$ say, for (2). With exception of $l2'$, the processing of G$'$ and H$'$ is very much the same as that of G and H; therefore in the PPP-branch, the token colour instances (identifiers) are replaced by the variables x (for G or G$'$) and y

(for H or H′), and – a technicality – appropriate guards are attributed to the reactions $r1$ to $r4$. Finally, because the molecules moved onto F6P by the glycolysis resp. the 'normal' PPP are C resp. H, the reaction $l2'$ may be enabled only for H′-molecules. Hence the arc pointing to $l2'$ gets the label H′ and the reaction $l3$ gets the guard [x <>H′].
This leads to $\mathcal{P}^*$ in Figure 3 which is appropriate both for computing T-invariants and for simulation (subsection 4.2) as unreasonable processes and cyclic loops have been excluded.

4 Defects, Effects and Invariants

The following calculations were made by use of an experimental software package SY, written by H. Genrich in Standard ML, for the symbolic analysis of coloured Petri nets.[4] This package supports symbolic calculations based on the incidence matrix of an executable coloured net in Design/CPN. It inspects and adopts the internal tables produced by the Design/CPN simulator for the graphical model and its data base. It allows, among others, to form symbolic dot products and matrix products, and to apply useful reduction rules and different formats for presenting the results.

4.1 Defects and S-invariants

We start with looking for S-invariants in the net $\mathcal{P}_{rev}$ of Figure 2. Obviously, there are two pairs of ubiquitous substrates which, if produced or consumed by a reaction, are transformed into each other, namely ADP and ATP, respectively NAD^+ and NADH. This is verified by applying the function `DEFECT` of the package SY to the S-vectors

[(ADP, 1`D), (ATP, 1`D)] resp. [($NADP^+$, 1`D), (NADPH, 1`D)] .

Doing this yields the null defect in both cases, which means that the S-vectors above constitute S-invariants.[5]

Clearly, it is of much greater importance to deal with the *full* set of all primary (non-ubiquitous) substances. In steady state, there should exist an S-invariant comprising that *full primary set*. To find out the weight factors of a *full* S-vector, i.e., covering this set, we proceed step by step. First we observe that each molecule of FBP is transformed into two GAP-molecules by the reactions $l4$ and $l5$. We conclude that in any S-vector, to finally become an invariant, the places (concentrations) of the glycolysis pathway GP from Gluc unto FBP must get a weight factor twice as high as GAP and the following places down to Lac. Trying to adopt the same principle to the pentose phosphate pathway PPP, however, leads to a non-null defect of the S-vector: applying the function `DEFECT` to

σ' = [(Gluc, 2`D), (G6P, 2`D), (F6P, 2`D), (FBP, 2`D), (DHAP, 1`D),
($NADP^+$, 1`D), (NADPH, 1`D), (GSSG, 2`D), (GSH, 1`D),
(Ru5P, 2`D), (R5P, 2`D), (Xu5P, 2`D), (S7P, 2`D), (E4P, 2`D),
(GAP, 1`D), (BPS, 1`D), (Lac, 1`D)]

[4] Persons interested in this package should contact the author via email at `hartmann.genrich@gmd.de`.

[5] Usually, in tools like METATOOL and in low-level Petri nets, S-vectors are represented as n-tuples of integers which are ordered according to a fixed sequence of the n metabolites (places). In this paper, the conventions of the package SY are adopted in a simplified version: S-vectors are written as lists of pairs (node name, weighted token name), where the second elements are composed of an integer factor followed by a multiplicity sign " ` ", an optional boolean expression " [*bool*] %", and finally a token name. This latter construct represents a *distribution* (see section 2), mapping an element of the standard colourset CS (see footnote 3) to a linear combination over CS.

The defect of an S-vector σ^*, computed symbolically by the function `DEFECT`, is given as a list, in which each member $t : lico(CS)$ denotes a linear combination $lico(CS)$ of tokens that have to be added to an input or output place of transition t to make σ^* an S-invariant.

Dealing with the syntactic details of SY is beyond the scope of this paper.

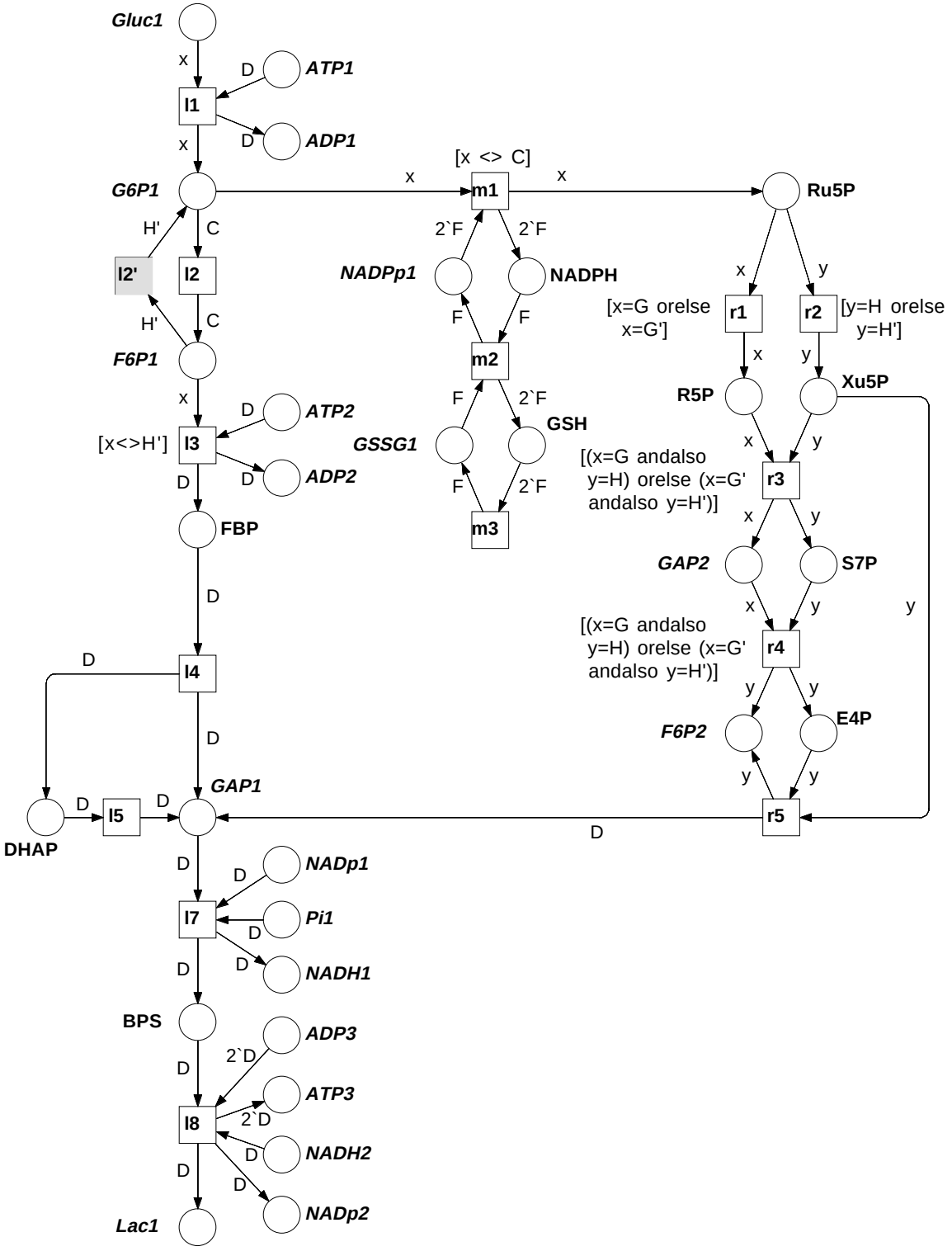


Figure 3: The net $\mathcal{P}^*$ with three flux modes

yields the defect $r3 : -\mathrm{D}$, $r3' : \mathrm{D}$, $r4 : \mathrm{D}$, $r4' : -\mathrm{D}$, $r5 : -\mathrm{D}$, $r5' : \mathrm{D}$.

To be more specific, the simulation of $\mathcal{P}^*$ and also its T-invariants computed in the next subsection show that, starting with 3 Gluc molecules, the GP produces 6, and the PPP 5 Lac molecules. Because these alternative paths share the metabolites G6P, F6P, FBP,

GAP, and BPS, it is not possible to find integer weight factors for these substances to make the full S-vector an invariant, as long as the metabolites are represented as undistinguishable tokens. We conclude that it is impossible, in this case, to find a full S-invariant with 'standard' means like low-level Petri nets or the algebraic package METATOOL ([Pfe99]). Hence, neither in [RLM96] nor in [SHWF96], a full S-invariant is reported.

Using high-level (coloured) nets with individual tokens offers the possibility to distinguish the mentioned metabolites according to the paths along which they are produced and consumed. Having done this in $\mathcal{P}_{rev}$, the stepwise construction leads to the S-vector

$\sigma =$ [(Gluc, 2`D), (G6P, 2`D), (F6P, 2`D), (FBP, 2`D), (DHAP, 1`D),
(NADP$^+$, 1`D), (NADPH, 1`D), (GSSG, 2`D), (GSH, 1`D),
(Ru5P, 2`D), (R5P, 2`D), (Xu5P, 2`D), (S7P, 2`D), (E4P, 2`D),
(GAP, (1`[x=D] % D + 2`[x=G] % D + 2`[x=H] % D)),
(BPS, (1`[x=D] % D + 2`[x=G] % D + 2`[x=H] % D)),
(Lac, (1`[x=D] % D + 2`[x=G] % D + 2`[x=H] % D))] .

Applying the function `DEFECT` to σ yields the null defect. Hence, σ is an S-invariant.

A brief inspection of σ reveals that it suffices to choose the weight 2 for all metabolites from Gluc unto FBP, F6P2, and E4P, irrespectively of the path on which they occur. For GAP, a threefold distinction has to be made: GAP-molecules produced by $r3$ or $r4'$ ([x=G]) or by $r5$ ([x=H]) get the weight 2, whereas those produced by $l4$ or $l5$ ([x=D]) get the weight 1. And this distinction is kept also for BPS and Lac.

The result then is that, in the GP/PPP network, the weighted sum of all metabolite concentrations has to obey the conservation rule represented by the S-invariant σ.

4.2 Effects and T-invariants

Looking for T-invariants which describe the feasible processes in the net we stay, for a moment, with $\mathcal{P}_{rev}$. Clearly, for each reversible reaction t and the reverse reaction t', the vector $[(t,1),(t',1)]$ is a T-invariant. But these T-invariants lack a sensible biochemical interpretation: a reversible reaction occurring permanently in both directions does not make sense. For this reason all reverse reactions except $l2'$ have been omitted at the end of section 3. On the other hand, we observe that $l2'$ gives rise to a process that is different from both the GP and the PPP, namely the gluconeogenesis. The tokens of that extra process got the identifiers G$'$ and H$'$, leading to the net model $\mathcal{P}^*$.

As with the S-vectors, also the T-vectors shall be established stepwise, i.e., not automatically but systematically. Apart from being able to see, from the *effects* computed at each step, the weight factor(s) of the transition(s) to be added successively to the T-vector, there is still another advantage of this approach. If we proceed along the causal chain of transitions, i.e., at each step selecting a subsequent transition which is enabled, we can compile knowledge about the amount of molecules which are needed *in course of* the run from Gluc to Lac and which have to be restored later during the run or after its end. This information is not provided by the effect of the complete T-vector: it only shows the *overall* effect of the vector.

When looking for the possible processes in the GP/PPP system $\mathcal{P}^*$ we soon find out that there are (at least) three sorts of processes (modes) that can be run totally independent from each other. Therefore we can attribute to each of them a characteristic parameter by which the weighted vector elements are multiplied additionally:

(G) Glycolysis: parameter GLY(),[6] or *gly* for short,

[6]The parameters are implemented as so-called *global functions*, to facilitate the experimental computation of the T-invariants. For convenience, the short-hands are used in the following.

(P) Pentose (or hexose) phosphate pathway: parameter HEX(), or *hex* for short,
(R) A mode including the reverse reaction $l2'$: parameter REV(), or *rev* for short.

During the stepwise construction of the T-vector(s) we gather, on the one hand, information about those molecules that are needed at the beginning or in the course of a run to reach its end at Lac. These molecules and their amounts are:

Consumed Substances			
ADP:	$(2*gly+5*hex+rev)$`D	ATP:	$(2*gly+5*hex+rev)$`D
F6P:	$2*rev$`H′	Gluc:	gly`C+$hex$`G+$2*hex$`H+$rev$`G′
GSSG:	$(6*hex+6*rev)$`F	NAD^+:	$(2*gly+5*hex+rev)$`D
$NADP^+$:	$(6*hex+6*rev)$`F	P_i:	$(2*gly+5*hex+rev)$`D

On the other hand, we gather information about those molecules that are provided during or at the end of the full run. These molecules and their amounts are:

Produced Substances			
ATP:	$(4*gly+10*hex+2*rev)$`D	F6P:	$2*rev$`H′
GSSG:	$(6*hex+6*rev)$`F	Lac:	$(2*gly+5*hex+rev)$`D
NAD^+:	$(2*gly+5*hex+rev)$`D	$NADP^+$:	$(6*hex+6*rev)$`F

The final result of the construction is the complete parameterized T-vector[7]

$\tau' =$ [($l1$, [(1, gly`(x $\leftarrowtail$ C)) , (1, $hex$`(x $\leftarrowtail$ G)) ,
(2, hex`(x $\leftarrowtail$ H)) , (1, $rev$`(x $\leftarrowtail$ G′))]),
($l2$, [(1, gly`( ))]),
($l3$, [(1, gly`(x $\leftarrowtail$ C)) , (2, $hex$`(x $\leftarrowtail$ H))]),
($l4$, [(1, $(gly+2*hex)$`( ))]),
($l5$, [(1, $(gly+2*hex)$`( ))]),
($l7$, [(1, $(2*gly+5*hex+rev)$`( ))]),
($l8$, [(1, $(2*gly+5*hex+rev)$`( ))]),
($m1$, [(1, hex`(x $\leftarrowtail$ G)) , (1, $rev$`(x $\leftarrowtail$ G′)) ,
(2, hex`(x $\leftarrowtail$ H)) , (2, $rev$`(x $\leftarrowtail$ H′))]),
($m2$, [(6, $(hex+rev)$`( ))]),
($m3$, [(6, $(hex+rev)$`( ))]),
($r1$, [(1, hex`(x $\leftarrowtail$ G)) , (1, $rev$`(x $\leftarrowtail$ G′))]),
($r2$, [(2, hex`(y $\leftarrowtail$ H)) , (2, $rev$`(y $\leftarrowtail$ H′))]),
($r3$, [(1, hex`((x,y) $\leftarrowtail$ (G,H))) , (1, $rev$`((x,y) $\leftarrowtail$ (G′,H′)))]),
($r4$, [(1, hex`((x,y) $\leftarrowtail$ (G,H))) , (1, $rev$`((x,y) $\leftarrowtail$ (G′,H′)))]),
($r5$, [(1, hex`(y $\leftarrowtail$ H)) , (1, $rev$`(y $\leftarrowtail$ H′))]),
($l2'$, [(2, rev`( ))])] .

Applying the function `EFFECT` from the package SY to τ' yields the effect

ADP: $-2*gly$`D $-5*hex$`D $-rev$`D
ATP: $2*gly$`D $+5*hex$`D $+rev$`D

[7] Adopting the conventions of the package SY in a simplified version, T-vectors are written as lists of pairs (transition name, list of weighted substitutions), see also footnote 5. A weighted substitution consists of an integer, followed by a (parameter) integer, a multiplication sign " ` " and a substitution in parentheses " ( ) ". A substitution, represented by " $\leftarrowtail$ ", indicates which variable(s) of the arc labels have to be substituted by which colour(s); if empty, the arc labels are constant colours and need no substitution.

The effect of a T-vector τ^*, computed symbolically by the function `EFFECT`, is presented as a list of constructs $s : lico(CS)$, each one denoting a linear combination of tokens, $lico(CS)$, that has to be added to (or subtracted from) place s to make τ^* a T-invariant.

Gluc: $-gly$`C $-hex$`G $-2*hex$`H $-rev$`G$'$
Lac: $2*gly$`D $+5*hex$`D $+rev$`D
P_i: $-2*gly$`D $-5*hex$`D $-rev$`D .

This leads to the parameterized equation for the effect of τ'

$(2*gly+5*hex+rev)$`ADP+$(gly+3*hex+rev)$`Gluc+$(2*gly+5*hex+rev)$`P_i
$=(2*gly+5*hex+rev)$`ATP+$(2*gly+5*hex+rev)$`Lac

yielding the three *overall reaction* equations

	parameters	overall reaction
(G)	$gly=1, hex=0, rev=0$	2 ADP + Gluc + 2 P_i = 2 ATP + 2 Lac
(P)	$gly=0, hex=1, rev=0$	5 ADP + 3 Gluc + 5 P_i = 5 ATP + 5 Lac
(R)	$gly=0, hex=0, rev=1$	ADP + Gluc + P_i = ATP + Lac

T-invariants describe processes in a Petri net which restore the marking with which they started and thus can be executed cyclically. Obviously, τ' is *not* a T-invariant. A thorough inspection of $\mathcal{P}^*$ reveals that *no* full T-vector *at all*, i.e., comprising all primary metabolites from Gluc to Lac, is an invariant. To find one, we have to modify the model $\mathcal{P}^*$. We do this by glueing $\mathcal{P}^*$ with a subnet that closes the cycle from Lac to Gluc. This subnet contains a place StartEnd, initially marked by a dummy token D, and the transitions $s1$ for starting and $s2$ for ending a cyclic run. The transitions $s1$ and $s2$ are intended to compensate the non-null effects. To this end, we connect some of the substances of $\mathcal{P}^*$ (as fusion places) via arcs from $s1$ or to $s2$. At this stage, one of the great advantages of the stepwise construction of the T-vector τ' becomes clear. The tables *Consumed Substances* resp. *Produced Substances*, derived above, exactly inform about the substances and their amounts which have to be provided by $s1$ resp. removed by $s2$ to arrive at a (parameterized) null effect process. The resulting subnet $\mathcal{P}^{se}$ is depicted in Figure 4.

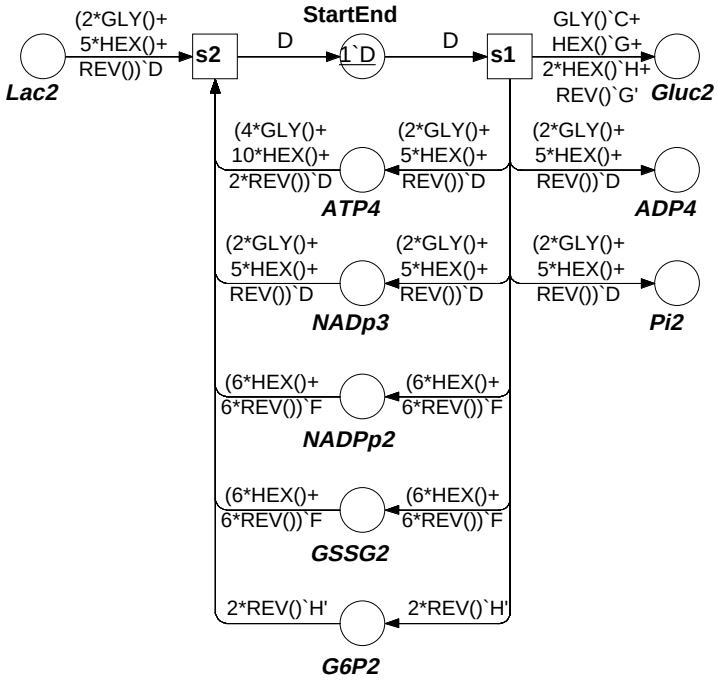


Figure 4: The subnet $\mathcal{P}^{se}$ completing $\mathcal{P}^*$ to form a cycle

Combining this subnet $\mathcal{P}^{se}$ with $\mathcal{P}^*$ by means of the fusion places, yields the cyclic net model that we aimed at. Let τ denote the T-vector achieved by adding the elements $(s1, [\ (1, (\ ))\ ])$ and $(s2, [\ (1, (\ ))\ ])$ to τ'. Then this T-vector τ has no effect and hence is a parameterized T-invariant.

The minimal T-invariants (elementary modes) derived from τ by setting one of the parameters gly, hex, or rev to 1 (and the remaining two to 0) are, in a short-hand notation,

$gly = 1, hex = 0, rev = 0$:
(tG) = [($l1$, C), ($l2$, D), ($l3$, C), ($l4$, D), ($l5$, D), ($l7$, 2∗D), ($l8$, 2∗D), ($s1$, D), ($s2$, D)]

$gly = 0, hex = 1, rev = 0$:
(tP) = [($l1$, G + 2∗H), ($l3$, 2∗H), ($l4$, 2∗D), ($l5$, 2∗D), ($l7$, 5∗D), ($l8$, 5∗D),
($m1$, G + 2∗H), ($m2$, 6∗D), ($m3$, 6∗D),
($r1$, G), ($r2$, 2∗H), ($r3$, (G,H)), ($r4$, (G,H)), ($r5$, H), ($s1$, D), ($s2$, D)]

$gly = 0, hex = 0, rev = 1$:
(tR) = [($l1$, G′), ($l2'$, 2∗D), ($l7$, D), ($l8$, D), ($m1$, G′ + 2∗H′), ($m2$, 6∗D), ($m3$, 6∗D),
($r1$, G′), ($r2$, 2∗H′), ($r3$, (G′,H′)), ($r4$, (G′,H′)), ($r5$, H′), ($s1$, D), ($s2$, D)] .

The three T-invariants (tG), (tP), (tR) are linearly independent, hence form a basis.

4.3 Biochemical Evaluation of the T-invariants

The software package METATOOL [Pfe99] allows to compute the *elementary (flux) modes* (corresponding to the minimal T-invariants) of a metabolic pathway. For each elementary mode, it computes (1) the T-vector, determining which reactions have to occur how often to restore the initial concentrations (marking) and (2) the overall reaction equation.

With the coloured Petri net approach and applying the package SY, we get additional information not only about the T-invariants but also about the dynamics of the system. The symbolic treatment of the T-vectors yields as one crucial result the marking (amount of molecules), needed at the beginning and provided by the starting transition $s1$, to run the system without deadlock from its source to the sink. This initial marking is 'appropriate' because it is the minimum amount of molecules necessary for a simulation with maximum concurrency. Moreover, the stepwise construction of the symbolic parameterized T-invariants yields knowledge not only about the frequency of transition occurrences (during a run along the invariant) but also about the partial order in which these transitions have to occur.

An interesting question arises concerning the independence of the three T-invariants. Theoretically, they are linearly independent from each other because the transitions $l2$ and $l2'$ are treated as not being related to each other. If however $l2$ and $-l2'$ are identified, as they represent complementary reactions, the T-vectors get linearly dependent. This corresponds to the observation that the overall reactions (G), (P), (R) are related to each other by the equation (P) = 2·(G) + (R).
The problem, however, lies in the fact that a steady state process including both a reaction and its reverse is biochemically not feasible. And on the other hand, T-vectors with negative elements cannot be T-invariants according to the definition given in section 2.

The construction of the compound net $\mathcal{P}^*$ can also be looked at from a different perspective, throwing more light on the nature of the token colours and the conflicts. Let us discuss the three independent modes identified in the previous subsection 4.2, as separate net models. They are depicted in a simplified version as Figure 5, omitting all ubiquitous molecules and the 'uncritical' reactions $m2$ and $m3$.

The first mode (G), glycolysis, contains no conflict. So, in principle, only one token colour would be needed. The colours C and D are retained merely to exhibit the relation

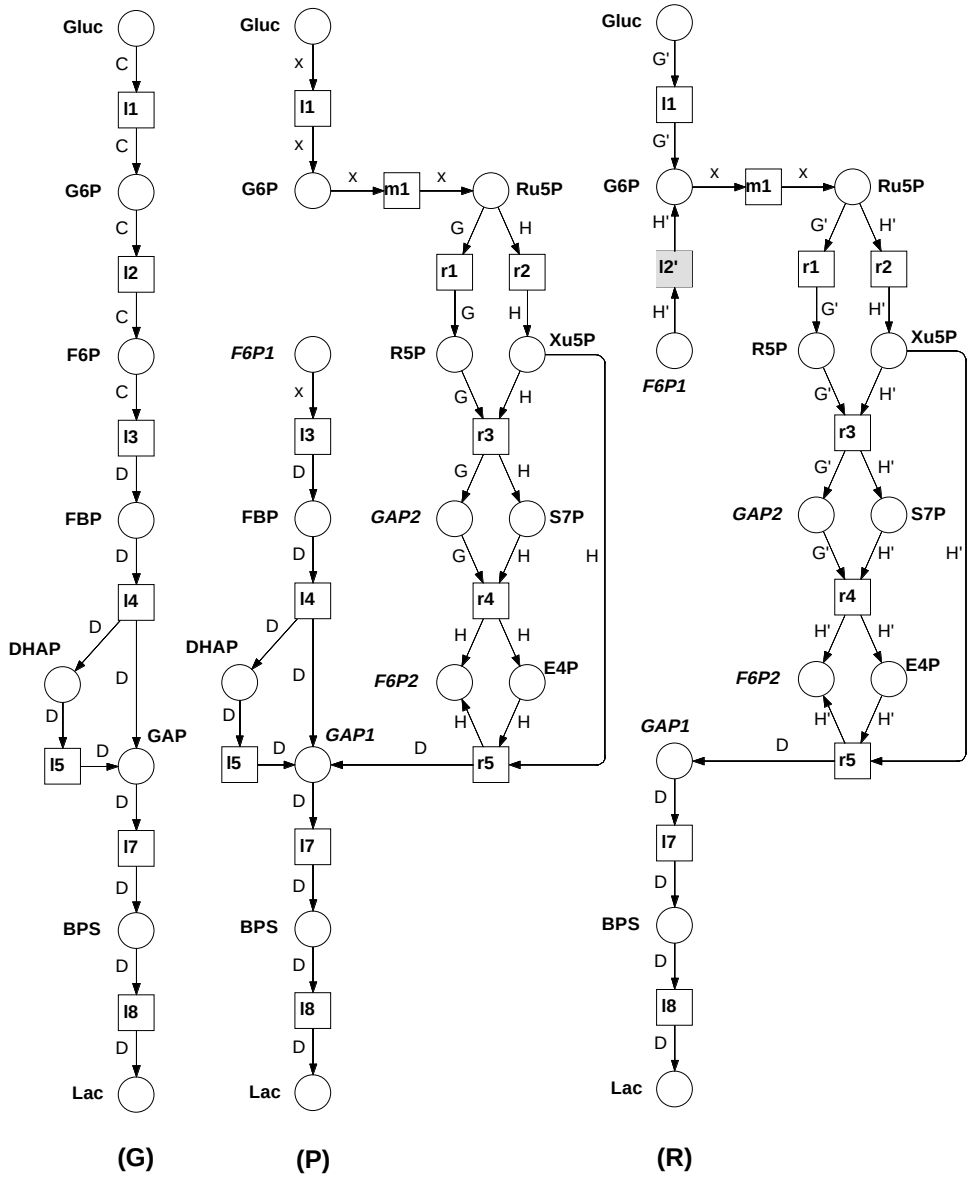


Figure 5: The (simplified) three modes of $\mathcal{P}^*$

with $\mathcal{P}^*$. The second mode (P) has two internal conflicts at Ru5P resp. GAP which are decided by use of the two colours G and H resp. G and D. The third mode (R) contains the same two conflicts as (P), solved by G′ and H′ resp. G′ and D.

These 'mode specific' conflicts describe (model) situations as happening in reality, with a great number of molecules of every substance involved. From the definition of steady state it follows that, sloppy speaking, no molecule inserted by the source may get stuck on its way to the sink. If it would, the concentration of an intermediate substance would be increased, contradicting the definition. Looking at the right hand branch of (P) in Figure 5, the tokens entering that branch at Ru5P can leave it only as F6P- or GAP-molecules by means of $r4$ and $r5$. The reaction $r4$ needs one G and one H, and $r5$ one additional H. The one G resp. two H tokens can only be provided by $r1$ occurring once resp. $r2$ occurring twice. In real organisms, with great molecule numbers, the molecules of one substance cannot be distinguished and cannot be forced to choose one out of more

alternative paths. Yet, the transitory increase of a substance concentration leads to a slowing down of reactions producing it and an acceleration of reactions consuming it. The opposite happens in case of a concentration decrease. So, in the long run, a relative occurrence ratio of 1:2 will be established among $r1$ and $r2$ if the pathway is in a steady state.

In contrast to the mode specific conflicts, the remaining ones are consequences of glueing the mode nets (G), (P), and (R) into one single model $\mathcal{P}^*$. As these three processes are independent from each other, performing linear independent T-invariants, the parameters gly, hex, rev can, in principle, be chosen arbitrarily. This implies that the relative frequency among this kind of conflicting reactions, for example $l2$ and $m1$, depends merely on the choice of the parameters and not on a biochemical law that would require a constant frequency ratio. In a real organism, these reaction ratios are controlled mainly by the relative activities and concentrations of the respective enzymes.

From a biochemical point of view, there exist further restrictions for the concurrent execution of the three processes (G), (P), (R). For example, the concurrent execution of (G) and (R) would require simultaneous activities of both reactions $l2$ and $l2'$, the reverse of $l2$. However, as the direction in which a reversible enzymatic reaction occurs is controlled by the concentrations of its reactants and products, it cannot run in *both, opposite* directions in a steady state, with constant enzyme concentrations. These concentrations depend on the current requirements of the cell.

In a real organism, the flow of G6P or Gluc depends on the need for NADPH, R5P, and ATP in the cell. Based on experimental observation, biochemists distinguish between four 'modes' (which we will call *T-modes*, in order to not mistake them for the elementary flux modes) of the combined GP/PPP, see [Str96]. We finish this section with a short discussion of these T-modes and their relationships to our results. Doing this, we shall neglect the ubiquitous molecules H^+, H_2O, and CO_2 mentioned in [Str96].

T-Mode 1 is adopted when more R5P than NADPH is required, for example in rapidly dividing cells needing R5P for the synthesis of nucleotide precursors of DNA. Most of G6P is converted into F6P and GAP by the GP ($l2, l3, l4$). Transaldolase ($r4'$) and transketolase ($r3'$) then convert 2 F6P- and 1 GAP- into 3 R5P-molecules. The reaction reads

5 G6P + ATP $\rightarrow$ 6 R5P + ADP .

First, we recognize that we have to return to the model $\mathcal{P}_{rev}$ which contains all reverse reactions needed. Secondly, we observe that the process (reaction path) does not transform Gluc into Lac, hence, does not represent a full T-vector. As a consequence, the chosen token colours are no longer appropriate. Bearing this in mind, we construct the T-vector

$[\,(l2,[(5,(\ ))]),(l3,[(1,(\ )]),(l4,[(1,(\ )]),(l5,[(1,(\ )]),(r1,[(4,(\ )]),$
$(r5',[(2,(\ )]),(r4',[(2,(\ )]),(r3',[(2,(\ )]),(r2',[(4,(\ )])\,]$

and compute its effect, yielding

ADP: D, ATP: $-$ D, F6P: 5`C $-$ 2`H $-$ 3`x, G6P: $-$ 5`C,
GAP: 2`D $-$ 2`H, R5P: 6`G, Ru5P: $-$ 4`G + 4`H .

By identifying all token colours with D, say, the effects for F6P, GAP, and Ru5P disappear, leading to the desired overall effect ADP: D, ATP: $-$ D, G6P: $-$ 5`D, R5P: 6`D which exactly reflects the reaction formula above.

T-mode 2 is adopted when the needs for NADPH and R5P are balanced. Then the oxidative branch of the PPP is executed, converting G6P into NADPH and R5P ($m1, r1$). The reaction formula is

G6P + 2 $NADP^+$ $\rightarrow$ R5P + 2 NADPH .

For the T-vector $[\,(m1,[(1,(x \leftarrowtail \mathrm{G}))]),(r1,[(1,(\ )])\,]$
we verify the appropriate effect G6P: $-$ G, NADPH: 2`F, $NADP^+$: $-$ 2`F, R5P: G .

T-modes 3 and 4 are adopted when much more NADPH than R5P is required.

T-mode 3 includes, apart from $l4'$ and $l2'$, a reaction catalyzed by fructose-1,6-biphosphatase which is part of the gluconeogenesis and hence outside the scope of the GP/PPP system covered by the models of this paper. Note that, although the mentioned enzyme converts FBP into F6P, it is *not* identical to the enzyme phosphofructokinase of $l3$ converting F6P into FBP. Rather, both reactions are irreversible.

T-mode 4, according to [Str96], is characterized by the reaction formula

3 G6P + 6 $NADP^+$ + 5 NAD^+ + 5 P_i + 8 ADP

$\rightarrow$ 5 Pyr + 6 NADPH + 5 NADH + 8 ATP .

If this process is expanded to start with Gluc (instead of G6P) and to end with Lac (instead of Pyr), and if the reactions $m2$ and $m3$ are included, the result corresponds precisely to the T-invariant (P) derived in the previous subsection 4.2.

5 Conclusions

To our best knowledge, this paper is the first one to apply higher-level Petri nets to the design, qualitative analysis, and execution of metabolic steady state system models. Compared to low-level Petri nets and to algebraic methods and tools from biochemistry, this approach renders important new results about the invariants and the processes of (sufficiently complex) metabolic pathways. The crucial point of using high-level nets is the ability to discriminate metabolites, if necessary, according to their topological environment, i.e., the reaction chains in which they are involved. On this basis, models can be developped which can be simulated smoothly and can be subjected to a rigorous symbolic analysis. This has been demonstrated for the rather complex sample of the combined glycolysis and pentose phosphate pathways. The main results reported in this paper are the following.

Firstly, a full S-invariant of the sample net was found that represents an interesting, non-trivial preservation law for the amounts of all metabolites in the system.

Secondly, the T-invariants and the overall reaction equations known from earlier publications have been verified. Moreover, not only the number of reaction occurrences related to a T-invariant, but also their partial order has been determined.

Thirdly, the sample net model can be simulated cyclically, restoring the initial system state at the end of each cycle, avoiding deadlocks, and with maximal concurrency.

Fourthly, a biochemical interpretation of high-level Petri net models of steady state pathways and their invariants requires great care, but may throw a new light on the nature of metabolic processes.

A most interesting topic for further research is the question whether or to which extent the search for and the construction of S- and T-invariants can be automated. Moreover, the significance and the biochemical interpretation of (full) S-invariants has been neglected almost totally in the past, and deserves a thorough investigation. Finally, the application of symbolic analysis to less understood metabolic systems is expected to lead to valuable new insights.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable hints and suggestions.

References

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

[Gen01] Genrich, H.: Dynamical Quantities in Net Systems. To appear in Formal Aspects of Computing, 2001

[GKV00] Genrich, H., Küffner, R., Voss, K.: Executable Petri Net Models for the Analysis of Metabolic Pathways. In Jensen, K.: Practical Use of High-level Petri Nets. Univ. of Aarhus, DAIMI PB–547, June 2000, 1-14.
Revised version to appear in Software Tools for Technology Transfer, Springer-Verlag, Berlin

[GoPe98] Goss, P.J.E., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. Proc.Natl.Acad.Sci.USA **95**, 1998, 6750-6755

[GoPe99] Goss, P.J.E., Peccoud, J.: Analysis of the stabilizing effect of Rom on the genetic network controlling ColE1 plasmid replication. Pac.Symp.Biocomp.'99, Hawaii, 1999, 65-76

[HeSc96] Heinrich, R., Schuster, S.: The Regulation of Cellular Systems. Chapman & Hall, New York, 1996

[HeSc98] Heinrich, R., Schuster, S.: The modelling of metabolic systems. Structure, control and optimality. BioSystems **47**, 1998, 61-77

[Hof94] Hofestädt, R.: A Petri Net Application of Metabolic Processes. Journal of System Analysis, Modelling and Simulation **16**, 1994, 113-122

[HoTh98] Hofestädt, R., Thelen, S.: Quantitative Modeling of Biochemical Networks. In Silico Biol. **1**, 1998, 980006

[KSH00] Koch, I., Schuster, S., Heiner, M.: Using time-dependent Petri nets for the analysis of metabolic networks. In Hofestädt, R., Lautenbach, K., Lange, M. (eds): DFG-Workshop: Informatikmethoden zur Analyse und Interpretation großer genomischer Datenmengen, Magdeburg, 19. - 20.5.2000. Fakultät für Informatik, Univ. Magdeburg, 2000, 15–21

[MDNM00] Matsuno, H., Doi, A., Nagasaki, M., Miyano, S.: Hybrid Petri Net Representation of Gene Regulatory Network. In Proceedings of the Fifth Pacific Symposium on Biocomputing, Hawai. World Scientific Press, 2000, 338-349

[Pfe99] Pfeiffer, T. et al.: METATOOL: For Studying Metabolic Networks. Bioinformatics **15**, 1999, 251-257

[RLM96] Reddy, V.N., Liebman, M.N., Mavrovouniotis, M.L.: Qualitative Analysis of Biochemical Reaction Systems. Comput. Biol. Med. **26**(1), 1996, 9-24

[RML93] Reddy, V.N., Mavrovouniotis, M.L., Liebman, M.N.: Petri Net Representation in Metabolic Pathways. In Hunter, L. et al. (eds.): Proc. First Intern. Conf. on Intelligent Systems for Molecular Biology, AAAI Press, Menlo Park, 1993, 328-336

[ScHi94] Schuster, R., Hilgetag, C.: On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. **2**, 1994, 165-182

[SFD00] Schuster, R., Fell, D.A., Dandekar, T.: Systematic exploration of the generic metabolic potential of sets of enzymes. Nature Biotechnol. **18**, 2000, 326-332

[SHWF96] Schuster, R., Hilgetag, C., Woods, J.H., Fell, D.A.: Elementary Modes of Functioning in Biochemical Networks. In Cuthbertson, R., Holcombe, M., Paton, R.: Computation in Cellular and Molecular Biological Systems, World Scientific, Singapore, 1996, 151-165

[SPM00] Schuster, S., Pfeiffer, T., Moldenhauer, F., Koch, I., Dandekar, T.: Structural Analysis of metabolic Networks: Elementary Flux Modes, Analogy to Petri Nets, and Application to Mycoplasma pneumoniae. In Bauer, E.-B., Rost, U., Stoye, J., Vingron, M. (eds): Proc.Germ.Conf.Bioinf., Heidelberg, 5.-7.10.2000, Logos Verlag Berlin, 2000, 115–120

[Str96] Stryer, L.: Biochemistry. W.H. Freeman and Co., New York, 1996

Abbreviations

Metabolites / Compounds			
ADP	Adenosine diphosphate	ATP	Adenosine triphosphate
BPS	1,3-Biphosphoglycerate	DHAP	Dihydroxyacetone phosphate
E4P	Erythose-4-phosphate	FBP	Fructose biphosphate
F6P	Fructose-6-phosphate	GAP	Glyceraldehyde-3-phosphate
Gluc	Glucose	GSH	Glutathione
GSSG	Glutathione disulfide	G6P	Glucose-6-phosphate
NADH	Nicotinamide adenine dinucleotide, reduced form		
NAD^+	NADp, Nicotinamide adenine dinucleotide, oxidized form		
NADPH	Nicotinamide adenine dinucleotide phosphate, reduced form		
$NADP^+$	NADPp, Nicotinamide adenine dinucleotide phosphate, oxidized form		
Lac	Lactate	PEP	Phosphoenolpyruvate
P_i	Orthophosphate, ionic form	Pyr	Pyruvate
Ru5P	Ribulose-5-phosphate	R5P	Ribose-5-phosphate
S7P	Sedoheptulose-5-phosphate	Xu5P	Xylulose-5-phosphate
2PG	2-Phosphoglycerate	3PG	3-Phosphoglycerate

Correspondence between Petri net transitions and enzymatic reactions			
l1	Hexokinase	l2	Phosphoglucose isomerase
l3	Phosphofructokinase	l4	Aldolase
l5	Triosephosphate isomerase (forw.)	l6	Triosephosphate isomerase (backw.)
l7	GAP dehydrogenase		
l8	*Reaction path consisting of:* phosphoglycerate kinase, phosphoglycerate mutase, enolase, pyruvate kinase, and lactate dehydrogenase		
m1	G6P *oxidation reactions*	m2	Glutathione reductase
m3	Glutathione oxidation reaction	r1	Ribulose-5-phosphate isomerase
r2	Ribulose-5-phosphate epimerase	r3	Transketolase
r4	Transaldolase	r5	Transketolase

Equivalent Coloured Petri Net Models of a Class of Timed Influence Nets with Logic

Bo Lindstrøm[1] and Sajjad Haider[2]

[1]Department of Computer Science, University of Aarhus, Denmark
E-mail: blind@daimi.au.dk

[2]System Architectures Laboratory, George Mason University, Fairfax, VA, USA
E-mail: shaider1@gmu.edu

Abstract

This paper discusses coloured Petri net models of so-called timed influence nets with logic. Previous work has developed a method to translate a timed influence net into a coloured Petri net. The work in this paper describes a new and more compact translation from timed influence nets with logic into coloured Petri nets. The translation has the property that the net structure of the coloured Petri net will be the same for all translated timed influence nets – only the initial marking changes depending on the actual timed influence net. This more compact translation avoids the generation of simulation code for each timed influence net. The paper also presents some validation results to establish that the coloured Petri net models from the two methods are equivalent, i.e. that the new compact coloured Petri net model gives the same simulation results as the old less compact model does.
Key words: Coloured Petri nets, Timed influence nets with logic, Equivalence, Folding, Information assurance.

1 Introduction

When making decisions it may be very important to be able to maximise or minimise the probability of certain events to appear. Being able to solve this problem is relevant in several different areas. For command and control systems in the military it is, e.g. important to take the best possible actions to maximise the probability that an enemy will surrender. For a system administrator it is important to be able to minimise the down-time for some service.

This paper will use an example from the area of information assurance [12]. Information assurance aims to minimise the probability that an intruder breaks into a system and accesses secret information. Even though a lot of effort may be put into making systems secure, a system will not be more secure than the weakest point of the system. Therefore, it is very important to be able to analyse a system to find the probability that an intruder can access the secret information, and to find the weak points in a system.

The method described in this paper makes it easier for an analyst to determine the probability that an intruder can access the secret information. It requires a model of the environment to be created by the analyst. The model is specified as a non-cyclic graph with probabilities to determine how likely an event is to appear. The model will be specified using a slightly modified version of influence nets [1,11] which is called timed influence nets with logic (TINL). *TINL* is a variant of Bayesian nets [5] which makes it simple to specify probabilities. The logic of a TINL has nothing to do with temporal logic, instead the word logic refers to different kinds of nodes in a TINL. We will give more details on the definition of TINLs in Sect. 2.

Based on the specification of the TINL, coloured Petri nets (CP-nets or CPNs) [6–9] are used to estimate the probability that a certain event occurs, and the time of the appearance. Most often the analyst is only interested in the final probabilities, and is not at all interested in the details of the CPN model. However, in some situations the details of how a simulation has evolved may be of interest, e.g. the order of occurrences of binding elements may contain useful information.

Previous research in this field [14] has developed a method to automatically convert a TINL into a CPN model using the tool Design/CPN [2]. Given a TINL specification file, the method automatically creates a CPN model which has a net structure very similar to the TINL, i.e. for each node in the TINL a specific part of the CPN model can be identified to model that node. Therefore, the fact that the structure of the CPN model looks like the TINL means that it is easy for an analyst to recognise nodes from the TINL in the CPN model. This close relationship between the CPN model and the TINL has been important when trying to convince analysts who are used to working with TINLs that CPN models are able to solve the problem and give the correct results.

In a later project the method has been extended to give access to creating and simulating models via a web browser. Given a TINL-specification file which is uploaded to a web server, the method automatically generates the CPN model and the corresponding simulator code using Design/CPN. Based on the simulator code, a CGI script [3] is automatically generated using the method described in [10]. Using an automatically generated web page which contains TINL-specific information, the user is able to set initial conditions for the simulator of the CPN model, and to start a batch of simulations using the CGI script. After having performed a number of simulations, results including graphics are displayed in the web browser. Figure 1 shows one of the graphs that are included on a web page to display the results to the analyst using domain-specific graphical user interfaces. The graphs show how the probabilities of certain nodes in the TINL evolve during time. Using this web-based graphical user interface, the analyst will never see the CPN model. Actually, the analyst does not even need to know that a CPN model is used to produce the results.

The previous method performs its job as expected. However, the method has some drawbacks. The fact that a completely new CPN model (with all the places and transitions) has to be generated for every influence net, means that it takes rather long time to apply the method. Even though the CPN model is generated automatically, it takes time both to generate the net structure and declarations, and then afterwards to generate the executable simulator code for that specific CPN model.

In this paper we describe how to avoid to generate simulator code for every instance of a TINL. In that way we eliminate the time used for each TINL to both create the specific CPN

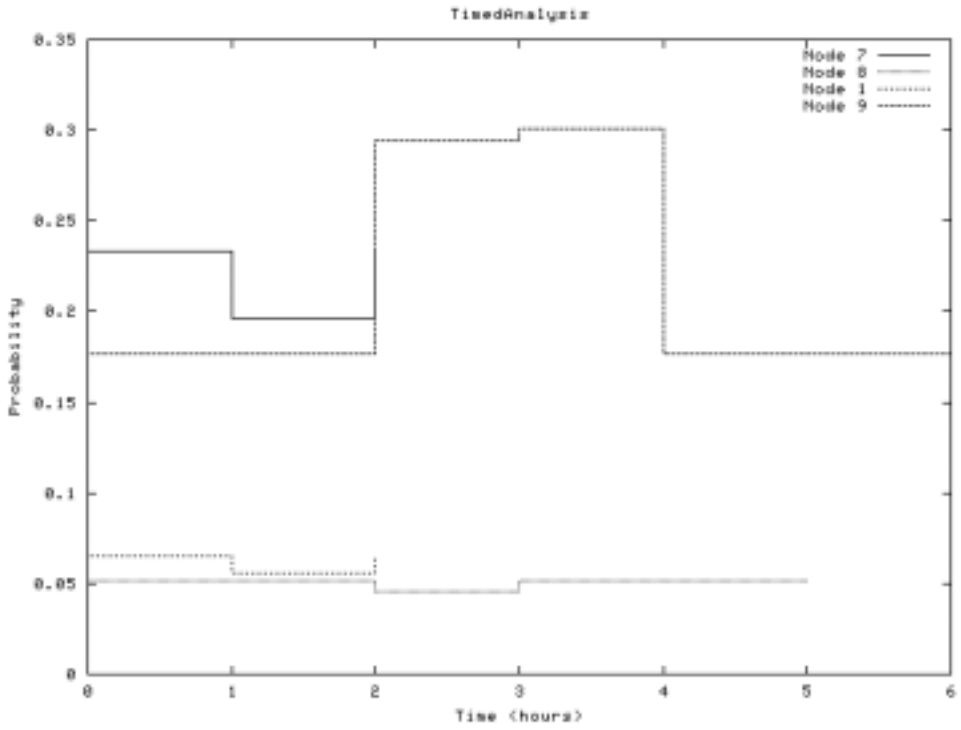


Figure 1: Graph from web page showing graphical analysis results of the TINL in Fig. 3.

model and to generate the corresponding simulator code. This is done by creating a general CPN model (in the following referred to as the *folded CPN model*) which can simulate any TINL for any set of parameter values, i.e. the whole class of TINLs. The general CPN model models the actual computation or execution of a TINL, but it is not fixed to a specific TINL. The information of a specific TINL is maintained using colours, and not net structure as in the former project. That means that the folded CPN model is fixed to a specific influence net by using a specific initial marking which contains information about the net-structure of the specific TINL. In other words, the structure of the specific TINL is encoded in colours.

The old method which generates a CPN model with net structure similar to the structure of the TINL (in the following referred to as the *unfolded CPN model*), is used in combination with the folded CPN model. The folded CPN model is only used for automatic simulations, i.e. simulations controlled via a web browser, while the unfolded CPN model is used when a person with knowledge of TINLs has an interest in looking at the actual CPN model.

The fact that the behaviour of the two models is expected to be the same requires some validation. Figure 2 illustrates the method applied in this paper. From a TINL a specification file is generated to contain all static information about a TINL. From that file, the old method developed in the former project is used to generate the unfolded CPN model, and then the corresponding state space is generated. The TINL specification file is also used to initialise the marking of the folded CPN model to reflect the current TINL. The corresponding state space is also generated

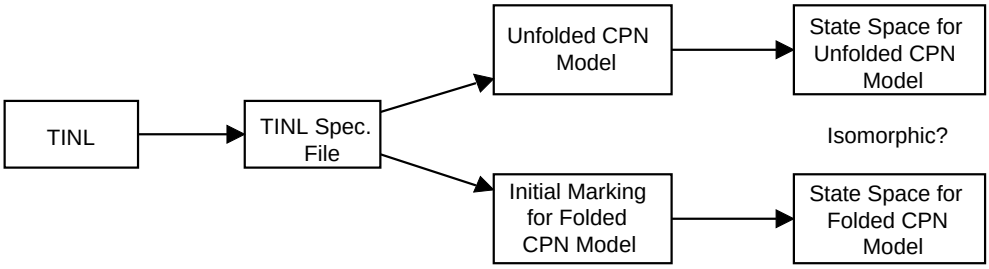


Figure 2: Overview of the applied method.

for the folded CPN model after the initialisation. Finally, the the equivalence of the behaviour of the two models has been validated by making a number of tests. This has been done by comparing the state spaces from the two CPN models.

The paper is structured as follows. Section 2 presents TINLs informally, and introduces an example which will be used for illustration purposes in the rest of this paper. Section 3 briefly describes how an unfolded CPN model will look like when it is generated using the old method. Section 4 describes the folded CPN model, and how it has been constructed from the unfolded CPN model. Section 5 discusses how we have validated that the behaviour of the two models are equivalent. Finally, in Sect. 6 we conclude and give directions for future work.

2 Timed Influence Nets with Logic

In this section we informally introduce timed influence nets with logic (TINL) by presenting an example in the field of information assurance. The example will be used as a running example in the following sections.

TINLs can be used to estimate the probability of a certain event when that event depends on other events which contains an element of uncertainty. Like e.g. Petri nets, TINLs have both a graphical and a mathematical representation. They are specified as directed acyclic graphs where the nodes represent events as random variables (a numerical quantity defined in terms of the outcome of a random experiment, p. 110 in [4]), while the arcs represent dependencies between the events. In addition timing information and probabilities are specified for the nodes.

Consider Fig. 3 which contains an example of a TINL. The purpose of the TINL is to estimate the probability that an intruder can unlock a door. To unlock the door two options exist. It is possible to unlock the door using both a password and an access card. It is also possible to unlock the door using a physical key and the same password which is used with the access card. It is of interest to an analyst to know which of the two options is most likely to be broken by an intruder, and how long time it will take to do it.

An intruder may be able to find either a physical key or an access card somewhere. The probability that the intruder finds a key is estimated to 30% (which is indicated by a so-called baseline probability *b=0.3* next to the node Find Key with number 4) while it is 40% for the access card (node number 3). The $h = 0.68$ on the arc to the node Have Key and Password

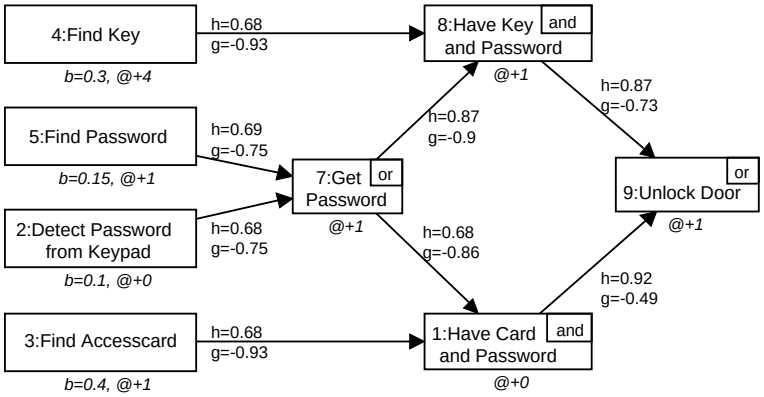


Figure 3: An example of an influence net for information assurance.

indicates that there is 68% probability that the node Have Key and Password will be true if the node Find Key is true. Likewise $g = -0.93$ indicates that there is 93% probability that the node Have Key and Password will be false if the node Find Key is false. The time to find a key is estimated to 4 time-units, which is indicated by the expression @+4 below the node Find Key.

The intruder is assumed to be able to obtain the password in two different ways. Either the intruder can find the password on a piece of paper Find Password with an estimated probability of 15%, or a special device can be used to detect the password from the keypad where the users types in the password Detect Password from Keypad with an estimated probability of 10%.

The node Get Password indicate the event that the intruder has obtained the password in some way, and therefore now knows the password. Notice that the node is a so-called *or-gate* which is reflected by the *or* in the upper right corner of the node. This indicates that we are only interested in the probability of the predecessor-event which has the highest probability, i.e. the post probability of the node is only to be based on the probability of the node with the maximal probability of all the predecessor nodes. Assume that n_7 denotes the random variable of the node Get Password and the predecessor nodes are denoted by p_2 and p_5. Then the probability of the or-gates is computed using the following formula:

$$P(n_7) = Max[P(n_7|p_2)P(p_2) + P(n_7|\neg p_2)P(\neg p_2), P(n_7|p_5)P(p_5) + P(n_7|\neg p_5)P(\neg p_5)]$$

Probabilities denoted by, e.g. $P(n_7|p_2)$ are so-called conditional probabilities. A conditional probability denotes the probability that a random variable (n_7) will be true when another random variable (p_2) is assumed to be true. Conditional probabilities are computed using the g and h values included in the TINL. We will not go into more details with how the probabilities are computed, but will refer to [13] for further details.

The two nodes Have Key and Password and Have Card and password indicate the events of having obtained two of the necessary items to unlock the door. These nodes are so-called *and-gates* which means that we require both predecessor events to be true. Therefore, the probability of the node will depend on both of the predecessor nodes, and not only one of them as it is the case for or-gate nodes. Consider node Have Key and Password and assume that n_8 denotes the

random variable of the node and that the predecessor nodes are denoted by p_4 and p_7. The probability of and-gates is computed using the following formula (where e.g. $P(n_8|p_4, p_7)$ denotes the probability that n_8 is true when the two conditional random variables p_4 and p_7 are true):

$$\begin{aligned} P(n_8) = \quad & P(n_8|p_4, p_7)P(p_4)P(p_7)+ \\ & P(n_8|p_4, \neg p_7)P(p_4)P(\neg p_7)+ \\ & P(n_8|\neg p_4, p_7)P(\neg p_4)P(p_7)+ \\ & P(n_8|\neg p_4, \neg p_7)P(\neg p_4)P(\neg p_7) \end{aligned}$$

Again, the conditional probabilities can be computed based on the g and h values on the arcs from the predecessor nodes. Notice that the formula for computing the probability of an and-gate is more complex than the formula for or-gates due to the fact that it requires conditional probabilities for more than one predecessor, e.g. the conditional probability stated by $P(n_8|p_4, p_7)$.

Finally, the node Unlock Door indicate the event that we are actually interested in, i.e. to estimate the probability that an intruder is able to unlock the door using any of the legal possibilities. This node is also specified as an or-gate because we do not care which of the predecessor events are true. It is sufficient that one of them are true to be able to unlock the door.

When having specified probabilities for an influence net, an algorithm using the two formulas stated above can be used to propagate the probabilities forward through the net. The propagation algorithm starts by updating the initial nodes (nodes without predecessors). Then it updates the successor nodes using the newly computed values, and continues to update nodes until no more successor nodes exists. Therefore, for each initial node the probability is propagated through the TINL to the terminal nodes.

Figure 1 from Sect. 1 actually shows a graph showing the evolution during time of the probabilities of the intermediate and terminal nodes of the TINL presented in this section. We see that the probability for unlocking the door (Node 9) will be about 17.6 after time 4.

To complete this informal presentation of TINLs, we briefly relate TINLs to two closly related classes of nets which are also used to estimate probabilities. TINLs are a variant of influence nets which again are a variant of Bayesian nets [5]. The difference is that for influence nets the predecessor nodes of a node are assumed to be independent from each other, while in Bayesian nets the predecessor nodes are not assumed to be independent. The independence of predecessor nodes in influence nets simplifies both the specification of probabilities in an influence net, and the actual computation of the probabilities. TINLs extend influence nets with the so-called or-gate, and with a time-concept.

3 Old Approach: Unfolded CPN Model

In this section we will briefly summarise how a CPN model of a TINL looks like when constructed using the method presented in [14]. However, the method has been slightly modified and extended to handle the or-gate discussed in Sect. 2 as well. Focus will be on the general pattern of a CPN model created from a TINL. We will not go into details with how to automatically generate the CPN model. Details on the automatic generation of a CPN model can be found in the paper [14].

Revisit the TINL in Fig. 3 in Sect. 2. Notice the three different types of nodes: initial (nodes without predecessors), intermediate (nodes with both predecessors and successors), and terminal nodes (nodes without successors).

Figure 4 presents the net-structure of an automatically generated CPN model for the TINL in Fig. 3. Each of the three types of nodes in the TINL is converted into one of three standardised subnets. Each rounded box models exactly one node in the TINL. Notice for later use in Sect. 4 the similar net-structure of the subnets, and that the subnets are connected only via places with colour-set Store.

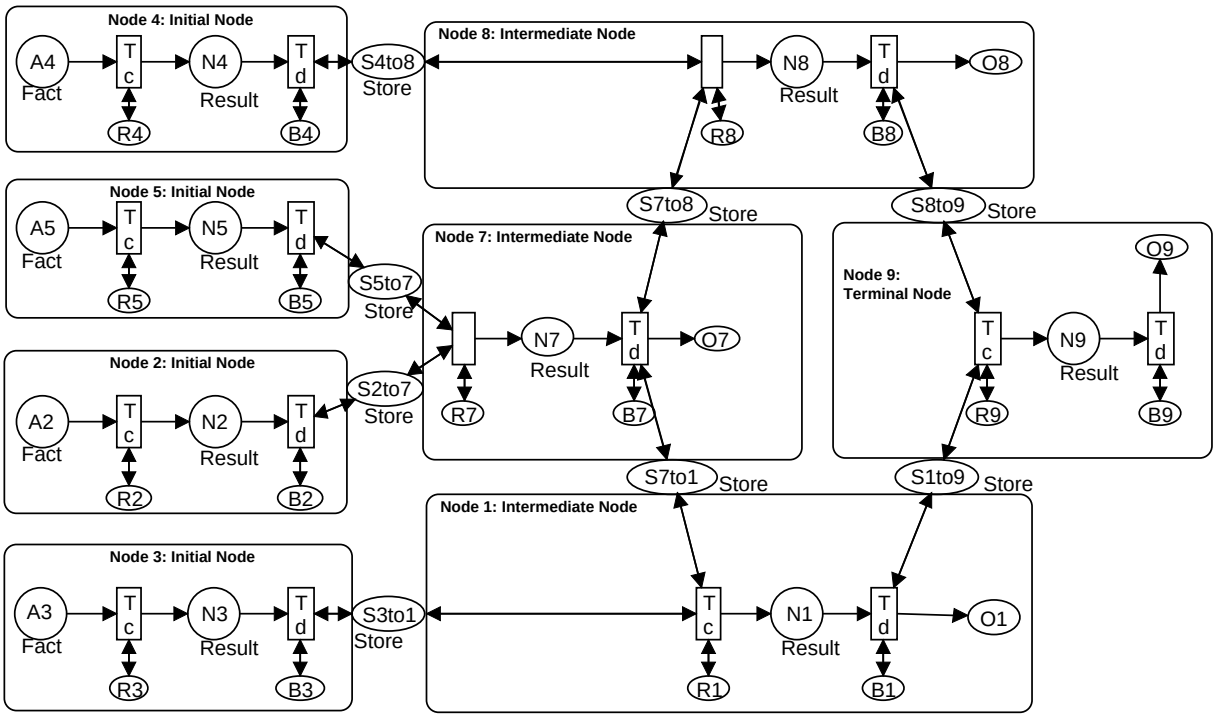


Figure 4: Net-structure of CPN model for the TINL in Fig. 3.

The number of predecessor and successor nodes of a given node may be different. Therefore, the number of Store places may be different for different subnets. Notice in Fig. 4 that the intermediate node number 8 has one successor node (S8to9) while intermediate node number 7 has two successor nodes (S7to8 and S7to1).

The model has two types of transitions and five types of places. The transitions named Tc are used to calculate new probabilities while the transitions named Td are used to distribute the newly calculated probabilities to the successor nodes. Figure 5 shows the details of the intermediate node number 7.

Places with the timed color-set Fact contain pairs of node id and the corresponding probability. These places are used to model the initial probabilities of a node (also called initial beliefs).

Places with color-set Store has a similar purpose as the Fact places. Like colour-set Fact, the colour-set Store contains a node id and a probability. However, in addition it contains a boolean

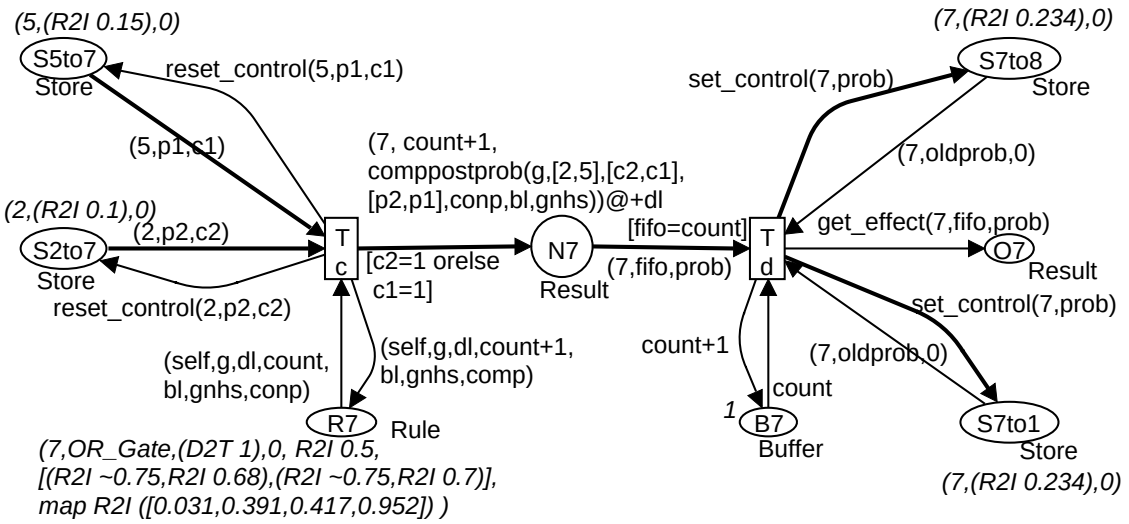


Figure 5: Details of intermediate node 7 in Fig. 4.

control value which is used to indicate whether the probability of that store has been recalculated by the predecessor node and not yet been processed by the successor node. This control value is used to propagate newly computed values forward.

The timed color-set Result is also a triple. It contains a node id, a sequence number, and the probability of the node. The purpose is to capture the resulting probabilities so that the analyst can inspect the values of the tokens when a simulation ends.

Color-set Rule is a 7-tuple. First of all, it contains the node id. The second value is the gate value indicating whether the node is an and-gate or an or-gate. The third value is the time-delay of the node which indicate how much time it takes to perform the action represented by the node. The fourth value is a counter used to give tokens a sequence number to be used for "first in, first out" handling of tokens. The fifth value is the baseline value of the node. The sixth value is a list of the g and h values described in Sect. 2. Finally, the last value is a list of conditional probabilities. The values in the list gives all the conditional probabilities stating how likely the node is to be true when any subset of the predecessors are true.

Finally, color-set SeqNo is a counter or sequence number which is used to ensure a first in, first out protocol when distributing tokens to the successor nodes.

As mentioned above, the transition Tc calculates the probability of the node given the probabilities of the predecessors. The function `comppostprob` computes the value based on probabilities from the predecessor stores and the values at the Rule place. However, for the intermediate and terminal nodes, the guard of the transition prevents the transitions from being enabled unless the probability of at least one predecessor node has been recalculated. That is done using the control value of the tokens on the predecessor Store places. When the tokens are put back to the Store places, the control value is reset to indicate that the probability has been propagated forward. This is the essential control mechanism of the forward propagation algorithm used for TINLs.

The transition Td is used to distribute computed probabilities to the successor nodes, and to the result place. The transition gets the next probability from the place N in fifo order, i.e. the token which has first arrived to the place N is first removed. In addition the transition removes the old probability from the successor-Store places, and replace these tokens with the new probability, and sets the control variable of these tokens to indicate that a new probability has been calculated.

The most important part of this section is to understand the general pattern used for constructing subnets for the individual nodes, and that any CPN model of a TINL can be created by combining these subnets into a complete CPN model. The detailed behaviour of the model is of secondary importance. In the next section we will describe the general or folded model which does not have so much net structure, but is instead able to model any TINL.

4 New Approach: Folded CPN Model

The method described in [14] generates a new CPN model for every different TINL as discussed in Sect. 3. In this section we describe a CPN model which can simulate any TINL when initialised with a proper marking. The CPN model (folded CPN model) is created as a folding of the CPN model (unfolded CPN model) presented in Sect. 3. This has several advantages: it is easy to validate that the folded CPN model has behaviour equivalent to the behaviour of the unfolded CPN model; when one knows the details of one of the models it is easy to learn the details of the other model; and it is easy to modify and maintain the models.

4.1 Hierarchy Page

The folded CPN model is created as a hierarchical CPN model with the hierarchy page depicted in Fig. 6. The CPN model has two prime pages. The page InitModel is used to load and set the initial markings of the CPN model, and will be discussed in the end of this section. The page Top is the top-page of the model which gives an overview of the CPN model. The three pages Initial Node, Intermediate Node, and Terminal Node models the three different types of TINL nodes.

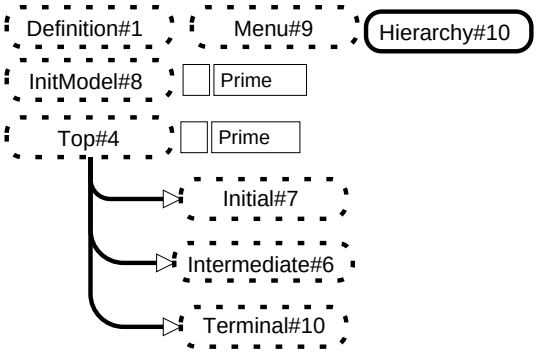


Figure 6: Hierarchy page for the folded CPN model.

4.2 Top Page

After being initialised, the transition Driver on the top page of the CPN model in Fig. 7 starts to activate the simulation of the TINL nodes. First at least one initial node must be activated, next all three kinds of nodes may be activated. The left part (left of the substitution transition Initial Node) and the output place O is similar to the same page in the unfolded CPN model except that in the unfolded CPN model, one output places existed for each node with output. In other words, we have folded several output places from the unfolded CPN model into a single output place O in the folded CPN model. The colour-set Result includes the id of the node. That implies that it is possible to distinguish tokens belonging to different nodes even though they are stored on a single output place.

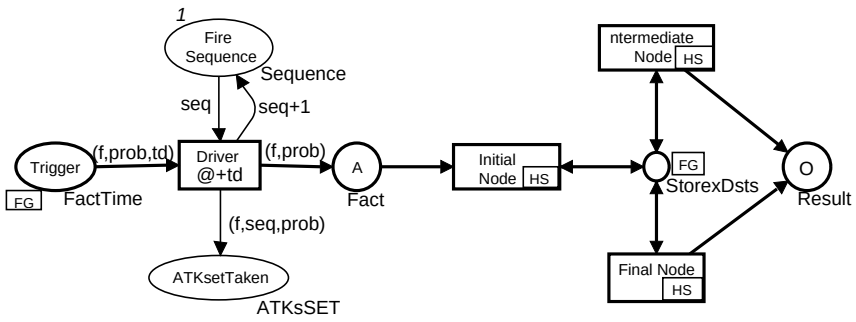


Figure 7: Top page of folded CPN model.

4.3 Folding Initial, Intermediate, and Terminal Nodes

Revisit Fig. 4 in Sect. 3. Notice that the structure of all initial nodes (nodes 2-5) is the same. Each initial node consist of four places and two transitions. The structure of the intermediate nodes (nodes 1, 7, and 8) is also the same like it is for terminal nodes (node 9). Only the number of Store places representing connections between nodes may be different.

Let us consider intermediate nodes in more detail. As mentioned above, all intermediate nodes have the same net structure. That means that we can fold all intermediate nodes into a single general folded node. Figure 8 shows the CPN model of the folded intermediate node which is a generalisation of nodes like Fig. 5.

4.3.1 Folding Places

The folded intermediate node has been obtained by folding all similar places and transitions. In this section we focus on folding places, while we consider folding transitions in Sect. 4.3.2.

Revisit Fig. 4 and focus on the places of the intermediate nodes 1, 7, and 8. Places with the same colour-set are folded into one place. E.g. the rule places (R_i, where i is the node id) are folded into one place (R in Fig. 8) representing rules for all intermediate nodes.

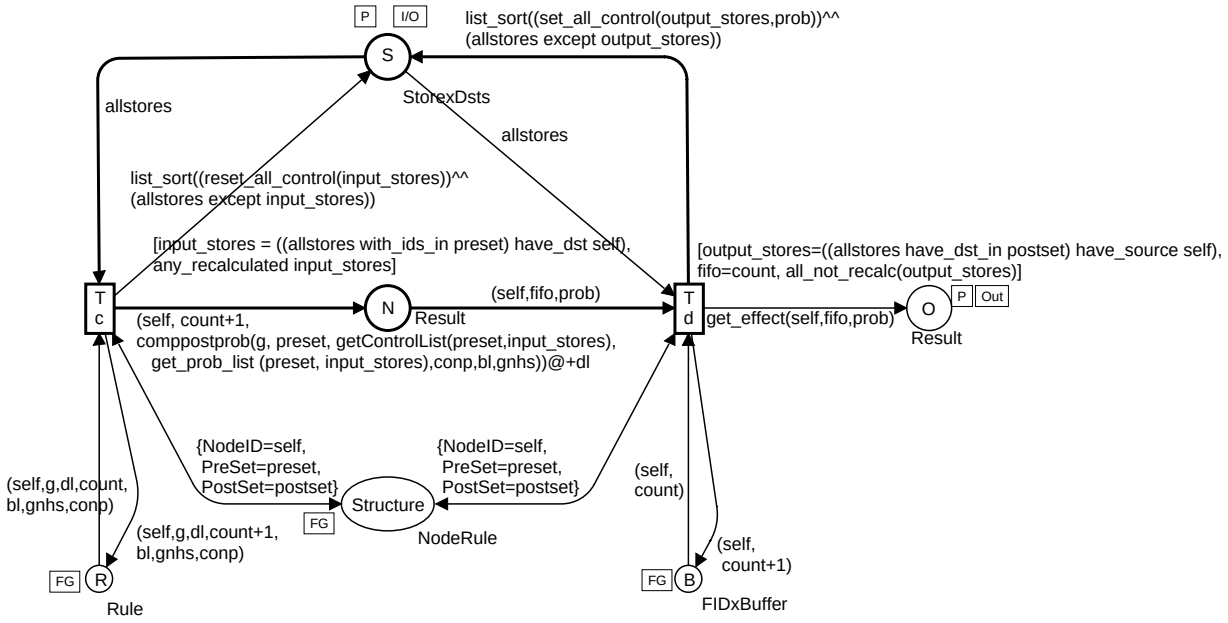


Figure 8: Folded intermediate page.

The actual folding has been done using the well-known method of adding a unique identity to each token to indicate which place in the unfolded CPN model the token belongs to. The colour-set Rule already contains the node id which makes the tokens from different nodes distinguishable or unique. In a similar manner the place N has been obtained by folding all places N_i, and the place O by folding places O_i. The place B in the folded CPN model is also a folding of all B_i places from the unfolded model. However, this colour-set has been modified to include the node id to make it possible to distinguish tokens from different nodes.

The colour-set StorexDsts of the place S in the folded model is the colour-set which has been changed the most. It is indeed the folding of all places S$i$to$j$, where i and j are respectively the ids of the predecessor node and the successor node. The colour-set Store in the unfolded model already contains the id of the predecessor node. However, as discussed above, a node may have several successors (node 7 has both nodes 1 and 8 as successor nodes in Fig. 4). Therefore, the id of the successor node (or destination) has been added to the colour-set StorexDsts to be able to distinguish tokens belonging to different successor nodes. For reasons related to the effectiveness of calculating enablings during simulations in Design/CPN, the colour-set is changed to a list of tokens. In summary, the place S contain a list of tokens from the places S$i$to$j$ from the unfolded CPN model which are made unique by adding the id of the successor node to each token.

We have added one new place which does not exist in the unfolded CPN model. It is the place Structure. It contains information about how intermediate nodes are connected to other nodes, i.e. structural information about how intermediate nodes are connected in the TINL. The colour-set NodeRule is a record with three values. The value `NodeID` is the id of the node. The values `PreSet` and `PostSet` contains lists of the ids of the predecessor and successor nodes

of the current node. As an example, a token representing all the arcs to/from the Store-node 7 in Fig. 5 is: `1‘{NodeID=7,PreSet=[2,5],PostSet=[1,8]}`. The nodes 2 and 5 are predecessors of node 7 while nodes 1 and 8 are successors. Using this place it is possible to encode all arcs between nodes into colours, and by initialising this place with tokens containing the appropriate information we can model different dependencies between nodes in a TINL.

4.3.2 Folding Transitions

Let us now consider the transitions `Tc` and `Td` in Fig. 8 for intermediate nodes. Like for the places, these two transitions are foldings of all intermediate `Tc` and `Td` transitions in the unfolded CPN model in Fig. 4. When a transition occurs in the folded model it reflects the behaviour of the occurrence of exactly one transition in the unfolded model.

Let us consider how transition `Tc` can be enabled. The simulator starts by taking a random token from the places `R` and Structure which has the same node id (`self`). The variable `allstores` on the arc from the place `S` to transition `Tc` binds to the single list on the place `S`.

Now the guard of transition `Tc` is evaluated. First the variable `input_stores` is assigned the list of values returned by evaluating the expression `((allstores with_ids_in preset) have_dst self)`. This expression runs through the list of all the stores and finds the stores with ids in the `preset` variable from the Structure place which have destination (or successor) `self`. Now the variable `input_stores` contains a list of the stores of the predecessor nodes of the currently considered node. Next, the expression `any_recalculated stores` tests if any of the `input_stores` have the control value set to indicate that they have been recalculated by predecessor nodes. This function corresponds to the guard of `Tc` in the unfolded CPN model. If the guard (the function `any_recalculated`) evaluates to true, the transition can occur.

When the transition `Tc` occurs a token is put on the place `N` to indicate that the node is in the process of being updated. The function `comppostprob` is exactly the same function as used in the unfolded model (see the corresponding arc expression in Fig. 5). By using the same function in the folded model, we avoid introducing errors by writing new functions for computing the probabilities. The functions `getControlList` and `get_prob_list` returns the control and probability lists from the `input_stores`. The control values of the `input_stores` are reset using the function `reset_all_control` on the arc from `Tc` to `S`, and appended to the unchanged stores (`allstores except input_stores`). The resulting list is sorted using the function `list_sort` to make a canonical representation of the list. The reason is related to generating the state-space for the model which is used for validation in Sect. 5. If we do not sort the list, the order of interleavings of transitions in the CPN model will have impact on e.g. the multi-set bounds of the place `S`.

Consider transition `Td` which distributes the newly computed probabilities to the `S` place. First the transition match the token on place `N`, the counter-token on place `B`, and the token on the place Structure with the corresponding node id. It also binds the variable `allstores` to the list on place `S`. A guard somewhat similar to the guard on transition `Tc` calculates the `output_stores` corresponding to the node which is currently considered. The expression `((allstores have_dst_in postset) have_source self)` returns the list of all

the stores with node id in the `postset`-list of the node – but only those which have source or node id `self`. The expression `fifo=count` in the guard ensures fifo-handling of the tokens on place `N`. This is similar to the guard in the unfolded model. Finally, the function `all_not_recalc` tests if all of the `output_stores` have control value equal to 0. This corresponds to the `0` in Fig. 5 on the arc from $S_7$to$_8$ to `Td`.

When transition `Td` occurs it updates the probability and the control field of the corresponding `output_stores`, and returns the list to the place `S` added to the remaining stores.

The folding of the initial and terminal nodes is conducted in a way similar to the one described above for intermediate nodes, and is therefore not included here.

4.4 Initialisation of the CPN Model

The fact that the folded CPN model is to simulate any TINL means that the model has to be initialised with appropriate data to reflect the structure of a specific TINL. This data is loaded into the model from a TINL-specification file. This file is generated using the CAT-tool[1] and contains all data needed to create the initial marking. The file is exactly the same file as the one used to create the unfolded model as described in details in [14].

Figure 9 depicts the page where tokens are computed and distributed to the places of the CPN model. The transition Distribute Tokens is the only transition being enabled initially. When it occurs, several things happens. First, the code segment of the transition loads the TINL-specification files for initialising the CPN model. Then multi-sets of tokens are generated using functions like `createBufferTokens` in Fig 10. Given a list of node ids from the TINL-specification file, the recursive function generates for each node id, a token `1‘(nid, 1)`. The rest of the markings for the remaining places are generated using similar more or less complex functions.

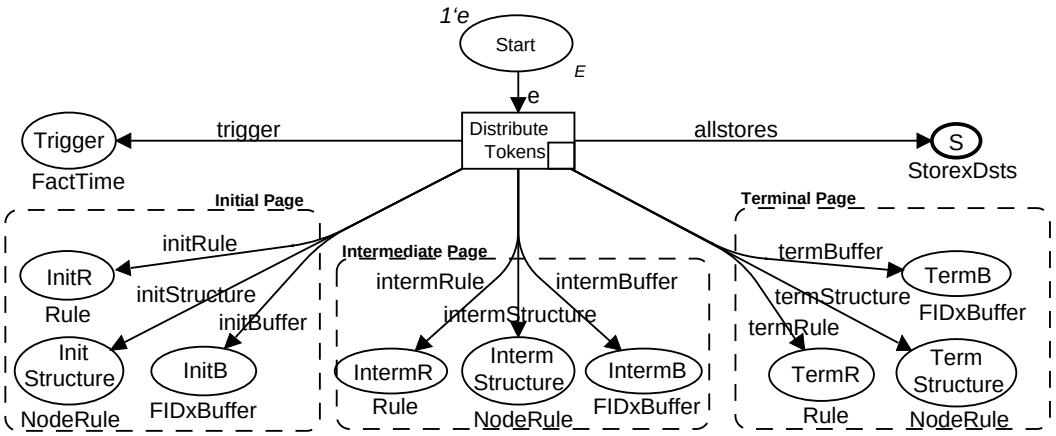


Figure 9: Loading and distributing initial tokens.

[1]CAT is the Effects Based Campaign Planning and Assessment Tool under development at the US Air Force Research Laboratories (AFRL/IF).

```
fun createBufferTokens ([]): FIDxBuffer ms =   empty
  | createBufferTokens (nid::rest) =
    1`(nid,1) ++ (createBufferTokens rest);
```

Figure 10: Create tokens with nid and a counter with initial value 1.

The distribution of the tokens is done by means of fusion places. For example, the places IntermR, IntermB, IntermStructure, and S are fused with the corresponding places (R, B, Structure, and S) on the intermediate page in Fig. 8. In this way tokens are loaded and computed on a single page of the CPN model, but are distributed to the places on several pages of the CPN model.

5 Validation

In this section we describe how we have validated that the folded CPN model and the unfolded CPN models gives the same simulation results. The folded CPN model is constructed from the unfolded CPN model in a way which makes us expect that the state spaces from the two CPN models will be equivalent.

The fact that the folded CPN model can model any TINL by being initialised with different initial markings means that we have a whole class of different CPN models. That means that when we have an initial marking of the folded CPN model, then we have one unfolded CPN model which has the same behaviour, and vice versa.

We could have conducted a mathemathical proof to prove that the two CPN models have equivalent behaviour. This would ensure that the equivalence holds for any TINL. However, we have chosen to use state space analysis to check that for the given TINLs, the equivalence is likely to hold. We will show that given a specific initial marking, the behaviour of the folded CPN model is equivalent to the behaviour of the unfolded CPN model. We will focus on sizes of the state spaces, boundedness properties, and paths in the state spaces. We have used the state space tool of Design/CPN for the analysis to be presented in this paper.

5.1 Model Similarities and Statistical Information of the State Space

Before we go into details with how the equivalence has been validated, we will briefly mention a few important issues related to how the folded CPN model has been obtained from the unfolded CPN models.

As mentioned in Sect. 4, when we have a transition in the folded CPN model, then this transition is the folding of a number of transitions in a unfolded CPN model. In other words, when we have a binding of a transition in the unfolded model, exactly one binding of a transition exists in the folded CPN model. The only exception from the correspondence between transitions in the folded and the unfolded CPN models is the transition Distribute Tokens depicted in Fig. 9 in Sect. 4. This transition has no counterpart in the unfolded CPN model. However, it is the first

transition to occur in the initial marking of the folded CPN model, and it will only be enabled once. That means that the impact on the state space is the addition of exactly one node and one arc to the state space for the folded CPN model. However, to avoid this extra node we first let the transition Distribute Tokens occur, and then start to generate the state space. That implies that we have the exact same number of nodes and arcs in the two state spaces. When we refer to the state space of the folded CPN model in the following, we will refer to the state space without the first node.

The fact that the folded CPN model has been constructed in the way mentioned above, makes it easier to check the equivalence of the folded and unfolded CPN models. The reason is that the number of nodes and arcs in the state space for each of the two models are equal, i.e. when there is one node in one of the state spaces, then there will be a similar node in the other state space, and similar for arcs. Therefore, ignoring this first node and arc from the state space makes the state spaces equivalent for a given initial marking of the two CPN models.

As an example, consider the TINL depicted in Fig. 3 in Sect. 2. We have generated the state space for both the unfolded and the folded CPN models. By generating the state space report for this example we obtain the statistical information in Table 1. The state space for the folded CPN model contains exactly the same number of nodes as the unfolded CPN model, and likewise for arcs. This gives us strong evidence to believe that the structure of the state spaces are identical.

Unfolded CPN model		Folded CPN model	
Nodes:	13930	Nodes:	13930
Arcs:	29595	Arcs:	29595
Secs:	301	Secs:	642
Status:	Full		Full

Table 1: Statistics for the state space for the TINL in Fig. 3.

We have also generated the strongly connected component graphs (SCC-graphs) for both CPN models, and they have exactly the same number of nodes and arcs as the corresponding state spaces. This was expected because TINLs are acyclic directed graphs, and thus the state space of the corresponding CPN model is to be an acyclic graph.

The pairwise equivalent nodes in the two state spaces are expected to have the same number of input and output arcs. In other words, from a given marking in either of the two CPN models, the same number of binding elements can be concurrently enabled. We have checked that this is the case by counting the nodes with the same number of input and output arcs {(0 input arcs, 0 output arcs), (0 input arcs, 1 output arc), ..., (n input arcs, n output arcs)}. As expected, it turned out that we got the same numbers from both CPN models. This check gives us further reason to believe that the statical structure of the TINL represented in the folded CPN model is correct.

5.2 Boundedness Properties

In this section we will focus on the results or markings produced by the CPN models – rather than the structure of the state spaces. We will show that for representative examples the two models give the same results. We will focus on the output places of the CPN models because the

Place	Upper Bound	Lower Bound
PN1'O1	3	0
PN1'O7	2	0
PN1'O8	3	0
PN1'O9	6	0

Table 2: Unfolded CPN model: integer bounds.

Place	Upper Bound	Lower Bound
Top'O	14	0

Table 3: Folded CPN model: integer bounds.

marking of these places are the ones showing the actual results of interest to the person simulating the model. In addition, the markings of these places are highly correlated with the correctness of the CPN models because the markings are based on the forward propagation of probabilities from the initial parameters.

Integer Bounds

First we consider integer bounds. Integer bounds give information about the maximal and minimal number of tokens which may be located on the individual places within the reachable markings. We can use that to check if the number of tokens in the folded and unfolded CPN models have the same bounds.

Table 2 shows the upper and lower integer bounds for output places (O_i) of the unfolded CPN model for our example TINL. We are able to use integer bounds only because the markings of these places are monotonically increasing, i.e. tokens are added but no tokens are removed from these places. From the table we see that in total there can be up to 14 (2+3+3+6) tokens on the output places during a simulation (in our case it will be at the end of a simulation), while the lower bounds are 0 for all output places which means that they are empty initially.

Table 3 shows the bounds for the folded CPN model. This model has only one output place which is the folding of the four places of the unfolded CPN model. We notice that the upper bound is 14 like the sum of the upper bounds for the unfolded CPN model. Also for this model the lower bounds is 0. Now we know that the bounds of the output places in the two models are the same which means that we are more confident that the two models gives the same number of outputs when given the same input parameters.

Multi-set Bounds

We will now focus on the actual values of the tokens in the two models. The multi-set bounds give us information about the values which the tokens may carry. By definition, the upper multi-set bound of a place is the smallest multi-set which is larger than all reachable markings of the place. We consider the multi-set bounds for the output places. From the upper multi-set bounds we will be able to see which probability values may be calculated during executions of the two models.

Place	Best Upper Multi-set Bounds
PN1'O1	1‘(1,1,(ii ("56131")))++ 1‘(1,1,(ii ("65095")))++ 1‘(1,2,(ii ("56131")))++ 1‘(1,2,(ii ("65035")))++ 1‘(1,3,(ii ("65035")))
PN1'O7	1‘(7,1,(ii ("196500")))++ 1‘(7,2,(ii ("233750")))
PN1'O8	1‘(8,1,(ii ("45673"))) ++ 1‘(8,2,(ii ("51935")))++ 1‘(8,3,(ii ("51935")))
PN1'O9	1‘(9,1,(ii ("294572")))++ 1‘(9,1,(ii ("300891")))++ 1‘(9,2,(ii ("171766")))++ 1‘(9,2,(ii ("294572")))++ 1‘(9,2,(ii ("300849")))++ 1‘(9,3,(ii ("171766")))++ 1‘(9,3,(ii ("176807")))++ 1‘(9,3,(ii ("300849")))++ 1‘(9,4,(ii ("171766")))++ 1‘(9,4,(ii ("176807")))++ 1‘(9,4,(ii ("300849")))++ 1‘(9,5,(ii ("176807")))++ 1‘(9,6,(ii ("176807")))

Table 4: Unfolded CPN model: best upper multi-set bounds.

Place	Best Upper Multi-set Bounds
Top'O	1‘(1,1,(ii ("56131")))++ 1‘(1,1,(ii ("65095"))) ++ 1‘(1,2,(ii ("56131")))++ 1‘(1,2,(ii ("65035"))) ++ 1‘(1,3,(ii ("65035")))++ 1‘(7,1,(ii ("196500")))++ 1‘(7,2,(ii ("233750")))++ 1‘(8,1,(ii ("45673"))) ++ 1‘(8,2,(ii ("51935")))++ 1‘(8,3,(ii ("51935"))) ++ 1‘(9,1,(ii ("294572")))++ 1‘(9,1,(ii ("300891")))++ 1‘(9,2,(ii ("171766")))++ 1‘(9,2,(ii ("294572")))++ 1‘(9,2,(ii ("300849")))++ 1‘(9,3,(ii ("171766")))++ 1‘(9,3,(ii ("176807")))++ 1‘(9,3,(ii ("300849")))++ 1‘(9,4,(ii ("171766")))++ 1‘(9,4,(ii ("176807")))++ 1‘(9,4,(ii ("300849")))++ 1‘(9,5,(ii ("176807")))++ 1‘(9,6,(ii ("176807")))
Initial'Structure	1‘{NodeID = 2,PreSet = [],PostSet = [7]}++ 1‘{NodeID = 3,PreSet = [],PostSet = [1]}++ 1‘{NodeID = 4,PreSet = [],PostSet = [8]}++ 1‘{NodeID = 5,PreSet = [],PostSet = [7]}
Intermediate'Structure	1‘{NodeID = 1,PreSet = [3,7],PostSet = [9]}++ 1‘{NodeID = 7,PreSet = [2,5],PostSet = [8,1]}++ 1‘{NodeID = 8,PreSet = [7,4],PostSet = [9]}
Terminal'Structure	1‘{NodeID = 9,PreSet = [1,8],PostSet = []}

Table 5: Folded CPN model: best upper multi-set bounds.

Compare Tables 4 and 5 containing the multi-set bounds for the output places in the two models for our example TINL. Notice that e.g. the place PN1'O7 in the unfolded CPN model contains two tokens with node id 7, sequence numbers 1 and 2, and probabilities 196500 and 233750. From the upper multi-set bounds for the place Top'O in the folded CPN model we see that these tokens will also be present in this model. By comparing the rest of the upper multi-set bounds we see that they are equal as well.

We have also included multi-set bounds for the places Structure in the folded CPN model (see Fig. 8 for details). From these bounds we see that the tokens describing the structure of the TINL are indeed as expected. This can be observed by comparing preset and postset of the individual nodes with the structure of the corresponding unfolded CPN model in Fig. 4.

5.3 Equivalent Paths in State Spaces

In addition to checking the equivalence of the behaviour of the two models only by means of the above mentioned techniques, we have tried to compare paths in the state spaces of the two models. We have checked that, whenever one of the models can make a step (let a transition occur) then the other model must also be able to make a step. This was done by exploring paths through the state space. We considered the arcs on the path in the state space of the unfolded CPN model. For every binding of a transition on the path in this CPN model, we checked if the corresponding folded transition in the folded CPN model was enabled. This was indeed the case for the paths that we followed. In addition, we checked that for every node on the path, the same number of successor nodes existed in both CPN models. From this comparison we get even more confident in the equivalence of the two CPN models.

However, we plan to do this more consequently than what we have done so far. We will define a mapping M_{unfold} from states in the folded CPN model into states in the unfolded CPN model. This mapping should be based on the equivalence between markings in the folded and unfolded CPN models. A similar mapping BE_{unfold} should be defined for mapping binding elements in the folded CPN model into binding elements in the unfolded CPN model. Then the two mappings M_{unfold} and BE_{unfold} should be applied to the state space of the folded CPN model to show that resulting state space is identical to the state space of the unfolded CPN model (modulo the extra initialisation marking and arc in the folded CPN model).

6 Conclusion and Future Work

When using CP-nets for creating models one have to be careful to choose the right level of folding. This paper shows that it is sometimes useful to have co-existing models with different degrees of folding.

One CPN model with a relatively extensive net structure to be used for understanding the model and for reference to the modelled system. This may require that the level of folding is kept so low that a new CPN model has to be created for each instance of the problem. However, if the model can be generated automatically as for the models presented in this paper, it may not be a problem.

The second model should focus on modelling the entire class of problems. This model is typically a folded version of the other one, which implies that the one and only model can be used for any instance of the problem. That means that the general CPN model will typically have relatively little net structure compared to the specific models.

A method for validating the equivalence between the folded and unfolded models has successfully been applied to several different TINLs with different time delays. Therefore, we believe in that the behaviour of the models are equivalent. However, we have not proved or verified that they are equivalent.

Future work will first of all focus on verification of the equivalence of the behaviour of the models. In particular we want to get more confident that the structure of the state spaces from the different models are equivalent. We plan to establish a formal proof to prove the equivalence of the behaviour of the models.

We will also experiment with alternative graphical interfaces for the simulators of the CPN models. The current web-based graphical interface is only useful for non-interactive simulations. By being able to interact with the simulator during simulations it will be possible to access the current state of the TINL represented by the CPN model during a simulation. This may be useful if the simulator is integrated with the CAT-tool (using TCP/IP communication) which is used to create the TINL. That would make it possible to display the results in the CAT-tool during simulations.

Acknowledgements. This research was conducted at the C3I Center of George Mason University (GMU), VA, USA, with partial support provided by the U.S. Air Force Office for Scientific Research under Grant No. F49620-98-1-0179. The work in this paper proposing or-gates in influence nets for information assurance is based on work done by Insub Shin, GMU. Special thanks goes to Alexander H. Lewis and Lee W. Wagenhals, GMU, for valuable discussions and support during the project. Finally, we also want to thank Kurt Jensen and the anonymous referees for constructive critique and usefull suggestions on improving the paper.

References

[1] R.T. Clemen. *Making Hard Decisions: An Introduction to Decision Analysis*. Duxbury Press, 1996. 2nd edition.

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

[3] Shishir Gundavaram. *CGI Programming on the World Wide Web*. O'Reilly & Associates, Inc., 1996.

[4] Paul G. Hoel, Sidney C. Port, and Charles J. Stone. *Introduction to Probability Theory*. Houghton Mifflin, 1971.

[5] J.V. Jensen. *An Introduction to Bayesian Networks*. UCL Press, 1996.

[6] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997. 2nd corrected printing.

[7] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997. 2nd corrected printing.

[8] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 3, Practical Use*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[9] 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, December 1998.

[10] B. Lindstrøm. Web-Based Interfaces for Simulators of Colored Petri Net Models. *To appear in: International Journal on Software Tools for Technology Transfer*, 2001.

[11] J.A. Rosen and W.L. Smith. Influence Net Modelling with Causal Strengths: an Evolutionary Approach. In *Proc. Command and Control Research and Technology Symposium*, pages 699–708. Naval Post Graduate School, Monterey, CA, USA, 1996.

[12] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley And Sons Ltd., 2000.

[13] L.W. Wagenhals. *Course of Action Development and Evaluation using Discrete Event System Models of Influence Nets*. PhD Dissertation, GMU/C3I/SAL-212-TH. C3I Center, George Mason University, Fairfax, VA, USA, January 2000.

[14] L.W. Wagenhals, I. Shin, and A.H. Lewis. Creating Executable Models of Influence Nets with Colored Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):168–181, December 1998.

Operational Planning: A Use Case for Coloured Petri Nets and Design/CPN

Lin Zhang
Senior Research Scientist
Systems Simulation and Assessment Group
Information Technology Division
Defence Science and Technology Organisation
PO Box 1500, Salisbury, SA 5108
Australia

Abstract

Operational planning is one of the key Command and Control (C2) functions performed by commanders and staff in military Headquarters. Effective operational planning must take into account of all aspects of an operational environment that is often extremely complex and uncertain. Software tools for planning with embedded reasoning capabilities would help improve efficiency of a planning process and facilitate effective planning and decision-making by commanders and staff. This talk presents a case for the Coloured Petri Nets (CPN) formalism and the Design/CPN tool to be used as a major part of a solution to the challenging problems in operational planning.

There are two parts in this presentation. Part One reports the modelling work in support of planning processes. It focuses on the use of coloured Petri nets in the modelling and analysis of planning processes in operational Headquarters in order to support the development of efficient and doctrine-based standard operating procedures (SOP). The process models developed with Design/CPN are hierarchical. With animated interactive simulation, the models can be used in military education and training of planning processes. State space analysis of the models is conducted to identify critical staff resources in terms of both concurrency and resource usage. Analytical results are also used to identify all possible flows of a process, and to observe effects of different resource allocations.

The second part of this presentation reports the modelling work in support of planning products. It is mainly concerned with the use of CPN and Design/CPN in conjunction with probabilistic modelling techniques in supporting the development and analysis of military courses of action (COA). It outlines the concept of an integrated modelling environment (InMODE) for operational planning that is currently being developed in Australia. Design/CPN is used as part of the reasoning engine in InMODE, contributing towards automated sequencing of tasks determined by planners in terms of their individual attributes such as pre-conditions, effects, and military resources. State-space analysis of the task models helps determine the feasibility of COAs.

Efficient Data-Structures and Algorithms for a Coloured Petri Nets Simulator

Kjeld H. Mortensen
Department of Computer Science, University of Aarhus,
Aabogade 34, DK-8200 Aarhus N, Denmark
k.h.mortensen@daimi.au.dk

Abstract

In this paper we describe how efficient data-structures and algorithms are used to dramatically improve the performance of a simulator for Coloured Petri Nets compared with earlier versions.

We have improved the simulator with respect to three areas: Firstly we have improved the transition occurrence scheduler algorithm so that we use lazy calculation of event lists. We only keep track of disabled transitions which we have discovered during the search for an enabled transition, and use the locality principle for an occurring transition in order to minimise the changes of enabling status of other transitions. Secondly we have improved the data-structures which hold multi-sets for markings. A kind of weight-balanced trees, called BB-trees, are used instead of lists as in the original version of the simulator. Although this kind of trees are more difficult to maintain at run-time they are surprisingly efficient, even for small multi-sets. Thirdly we have improved the search for enabled binding elements. We use the first enabled binding element we find in a fair search and make it occur immediately instead of calculating all bindings and then randomly select one. The search is guided by a binding "recipe" which is specially generated and optimised for each individual transition.

The improved simulator is implemented in both the Design/CPN and CPN Tools software packages, and has been used in several industrial projects.

1 Introduction

The first version of the Design/CPN [3] tool was completed in 1989. It included, among other things, a simulation engine for Coloured Petri Nets (CP-nets or CPN). In this paper we refer to this simulator as the *old simulator*. In 1994 the Design/CPN simulator was redesigned from scratch and the results were documented in a masters thesis by Haagh and Hansen [5]. We refer to this simulator as the *new simulator*. After several years of further development and implementation the new simulator was made publicly available in early 2000 in a release of Design/CPN. In Design/CPN the user can choose between the old and new simulator. In the meantime a new user interface has been developed, *CPN Tools* [2], which exclusively uses the new simulator. It is the intention that also the performance and state space tools eventually will use the new simulator engine.

The old simulator was made with data-structures and algorithms which were natural to use given the expectations of how the users would create CPN models. It was not anticipated that users would create models with hundreds of transitions, and places containing thousands of tokens on each. Markings of places are implemented by means of lists, and they work well for small multi-sets. Unfortunately we experience a performance penalty when we have a place with thousands of tokens which is frequently updated. Although the old simulator performs well for many known models we encounter more and more models for which it does not. Apart from the problem with large multi-sets we also see performance problems with timed simulations. The reason is that the old simulator calculates all enabled binding elements and then selects one of them. This is

expensive when we need to advance the simulation clock because we recalculate enabling for all transitions in order to find the smallest time value at which some binding element is enabled.

The new simulator project was a consequence of the increasing user demand for faster simulation support. Hence, for the new simulator we designed sophisticated data-structures for markings of places and fast algorithms for checking enabled transitions. These are the main topic of this paper and we explain the details of such data-structures and algorithms. The description of the design and implementation in this paper is a compact and updated version of the thesis work by Haagh and Hansen in [5].

We use the classical view of discrete event simulation (DES) where the central component is a scheduler with an event queue. However, instead of using a generic DES design pattern we adapt the design directly to central properties of CP-nets, such as the locality principle. On one hand we get a specialised design but on the other we obtain a very efficient implementation.

We have chosen not to go in depth with describing the performance experiments with various data-structures and CPN models. This is described in detail in [5]. Although the experiments provide the bases for deciding which data-structure performs best, we only note here that the performance results are still valid for the present version of the simulator.

We have also chosen not to attempt to directly compare the performance of the new simulator with the old simulator. The reason is that we believe it is very hard to construct experiments which will produce useful performance measurements. We have indeed tested a lot of small models and a handful of industrial models. Most of these ran faster in the new simulator, and it is just a matter of adding more tokens to the models in order to get results more in favour of the new simulator. One model, however, has exhibited the same performance in both the old and new simulator [13]. In this model most places use list encoded colour sets for the old simulator in order to avoid large markings and poor simulation performance. Hence there are only a few tokens and many list operations while simulating. Surely the new simulator will get the advantage if this model would have been built without a particular simulator in mind.

We assume the reader is familiar with the concepts of CP-nets from [8] and [9]. It is also an advantage to be experienced with writing Standard ML (SML) expressions and functions, since we show several extracts of source code from the simulator implementation. The source code has been collected in appendices and can be skipped without loss of continuity. We refer to [6] for more information on SML.

The paper is divided into the following sections. We use the main part of the paper to describe the scheduler algorithm in Sect. 2, data-structures for multi-sets for place markings in Sect. 3, and the techniques for calculating enabled bindings in Sect. 4. Each of these sections is divided into subsections for simulation without and with time respectively. Subsequently we provide a short survey of related work in Sect. 5, and conclude the paper with Sect. 6. Note also a number of appendices with source code listings at the end of the paper right after the bibliography.

2 Transition Occurrence Scheduler Algorithm

The central component in a discrete event simulator is the scheduler. It selects events which should occur (be executed) in the next step. This is also the case for our CPN simulator, where events are transition occurrences. The scheduler we describe in the following is optimised for CP-nets. A limitation of the scheduler is that it only supports *interleaving semantics* and does not exploit concurrently occurring transitions. In practice this does not cause any further problems but rather simplifies the implementation drastically. If we, on the other hand, should implement *true concurrency* it is necessary to make a complicated calculation of a non-conflicting multi-set of binding elements for each simulation step. In the following we separately describe the two cases with untimed and timed CP-nets.

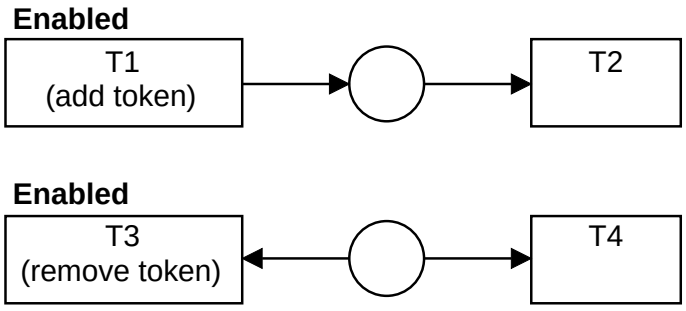


Figure 1: CP-net sketches to illustrate dependencies.

2.1 Scheduling without Time

The locality principle of CP-nets is essential to the scheduler of the new simulator. This principle states that an occurring transition will only affect the marking on immediate neighbour places, and hence the enabling of a limited set of neighbour transitions. For instance, consider the CP-nets sketched in Fig. 1. They illustrate the only two cases where transitions can change enabling status ($T2$ and $T4$) when tokens are added to ($T1$) respectively removed from ($T3$) a place. We assume that the places have some arbitrary marking and $T1$ and $T3$ are enabled. If transition $T2$ is disabled it may become enabled if $T1$ occurs, but if $T2$ is enabled it will remain so if $T1$ occurs. If transition $T4$ is enabled it may become disabled if $T3$ occurs, but if $T4$ is disabled it will remain so if $T3$ occurs. A dependency relation is used in the scheduler based on the locality principle.

In the following we describe how the new simulator schedules transitions for occurrence. The main idea is to check enabling of a transition only when absolutely necessary, because it is computationally relatively expensive to do so. This is the motivation for introducing a status for transitions, which can be *disabled* or *unknown*. When initialising the scheduler all transitions are given the status *unknown*, i.e., we do not know if the transitions are enabled or disabled. If we somehow discover that a transition is disabled, it will get the status *disabled*. All other transitions that we have not investigated will still have status *unknown*. Upon occurrence of a transition we update the status of neighbouring transitions while taking into account the before mentioned neighbourhood dependencies. We summarise these ideas in an algorithm given in Alg. 1.

T, set of transition instances in the given CP-net
$T_{disabled} \leftarrow \emptyset$, subset of T with elements having status *disabled*
$T_{unknowns} \leftarrow T$, subset of T with elements having status *unknown*
$step \leftarrow 0$, the simulation step counter

while $T_{unknowns} \neq \emptyset$ **do**
  $t_{candidate} \leftarrow \langle\langle$random element from $T_{unknowns}\rangle\rangle$
  $status \leftarrow \langle\langle$Try finding an enabled binding for $t_{candidate}$, make it occur
    and return *occurred*. Otherwise return enabling status.$\rangle\rangle$
  if $status = disabled$ **then**
    $\langle\langle$move $t_{candidate}$ from $T_{unknowns}$ to $T_{disabled}\rangle\rangle$
  else if $status = occurred$ **then**
    $step \leftarrow step + 1$
    $T_{dependents} \leftarrow (\{t_{candidate}\}\bullet)\bullet$
    $\langle\langle$move $T_{dependents} \cap T_{disabled}$ from $T_{disabled}$ to $T_{unknowns}\rangle\rangle$
  end if
end while

Algorithm 1: CPN simulation transition occurrence scheduling algorithm.

Notice that in the present version of Alg. 1 there are no stop criteria built in, but it can easily be extended for example so that the simulator will stop after a given number of steps. Another improvement which can be considered is where we calculate $T_{dependents}$. Here we take into account dependencies originating from double headed arcs. Actually they are not necessary and can safely be removed from the dependency set. Finally note that we do not calculate all enabled bindings, but rather choose the first enabled binding encountered in a fair manner. We elaborate on this later in Sect. 4.

The scheduler in automatic simulation mode in the old simulator is different from Alg. 1. In fact the old simulator does not even take into account the neighbourhood dependency relation of transitions. Basically the scheduler has a list of transitions and it searches this list cyclically for enabled transitions in a fixed order. The scheduler makes transitions occur immediately when they are determined to be enabled. An unfortunate property of this simple algorithm is a *shadowing effect* where an occurring transition higher on the list will always disable a transition lower in the list. This is typically the case where two transitions are in conflict. Shadowing is not a problem in the new simulator.

2.2 Scheduling with Time

Simulation with timed CP-nets is realised with a global clock (also called the model time) and time-stamps on tokens [9]. If there are no more enabled transitions at a particular time we increase the model time until a transition is enabled. Naively this involves calculating all colour enabled binding elements and then finding the one resulting in the smallest model time. In the following we describe a scheduler algorithm which is much more efficient in practice compared with the naive solution.

The main idea is that instead of calculating exactly how much the model time should be increased before we know that at least one transition is enabled, we only make a safe approximation. When asking a transition to attempt to find an enabled binding it may answer that it is not enabled at this time, and additionally propose when it *may be* enabled. We postpone the technical details of exactly how the transition calculates the approximate time to Sect. 4.2. Once we know the time proposal from all transitions we advance the model time to the smallest of these, and then we start the search for an enabled transition again. These ideas are summarised in an algorithm which can be found in Alg. 2.

The algorithm in Alg. 2 is an extension of Alg. 1 where we handle the extra time information. Notice that we introduce a priority queue, T_{maybes}, which holds transitions that are not enabled now, but instead we have estimated the time at which they *may be* enabled. Note also that the algorithm can both handle CP-nets with and without time, and can be simplified to Alg. 1 in case the given CP-net has no timed elements.

3 State Representation

A marking (or state) is a multi-set of token elements, i.e., the distribution of tokens on places at a given moment. Hence, in order to make a representation of a marking we need to have a data-structure for multi-sets. Colour sets are types on places which determine the element type of our multi-set data-structures. Hence our data-structures for multi-sets are also directly related to colour sets, and we need to have efficient data-structures to hold elements of a variety of types.

In the old simulator multi-sets are represented by means of SML lists. As we discussed in the introduction we cannot rely on lists with the increasing demand for large multi-sets from users. In this section we present an improved data-structure with logarithmic searching time complexity which is suitable for representing very large multi-sets.

Based on our experience with the old simulator and practical use of CP-nets for modelling systems we give a number of general requirements for the data-structures:

- Tokens are frequently added and removed from places during simulation. Hence the data-structure must be able to re-structure itself efficiently.

- We need to lookup and search for tokens on places. This means that we can only handle colour sets where equality (=-operator) is supported. In particular for sorted data-structures there must additionally be support for a <-operator.
- We need to support draws at random of tokens and generate random permutations from the data-structure. This is needed for supporting fair calculation of bindings. We also need to be able to mark individual tokens as reserved during the binding process, in order to ensure that the same token is never used several times for any binding.
- Efficient data-structures may be complex and hence we do not wish that the user access the *internal* representations directly. Instead we should provide a secondary *external* simple representation for the user.

Note that the SML type `real` does not support equality in the SML/NJ compiler we rely on and hence we cannot use this kind of colour set in the simulator. An alternative to `real` is the `IntInf` type, which is an unbounded integer type which supports the standard comparison operators.

T, set of transition instances in the given CP-net
$T_{disabled} \leftarrow \emptyset$, subset of T with elements having status *disabled*
$T_{unknowns} \leftarrow T$, subset of T with elements having status *unknown*
$T_{maybes} \leftarrow \emptyset$, priority queue, smallest time value has highest priority
$step \leftarrow 0$, the simulation step counter
$model\_time \leftarrow 0$, the simulation model time, never decreasing

while $T_{unknowns} \neq \emptyset$ **and** $T_{maybes} \neq \emptyset$ **do**
  while $T_{unknowns} \neq \emptyset$ **do**
    $t_{candidate} \leftarrow \langle\langle$random element from $T_{unknowns}\rangle\rangle$
    $status \leftarrow \langle\langle$Try finding an enabled binding for $t_{candidate}$, make it occur and return *occurred*. Otherwise return enabling status.$\rangle\rangle$
    if $status = disabled$ **then**
      $\langle\langle$move $t_{candidate}$ from $T_{unknowns}$ to $T_{disabled}\rangle\rangle$
    else if $status = occurred$ **then**
      $step \leftarrow step + 1$
      $T_{dependents} \leftarrow (\{t_{candidate}\}\bullet)\bullet$
      $\langle\langle$move $T_{dependents} \cap T_{disabled}$ from $T_{disabled}$ to $T_{unknowns}\rangle\rangle$
    else if $status = maybe\_enabled\_at(time\_proposal)$ **then**
      $T_{maybes} \leftarrow (time\_proposal, t_{candidate})$
      $\langle\langle$remove $t_{candidate}$ from $T_{unknowns}\rangle\rangle$
    end if
  end while

  if $T_{maybes} \neq \emptyset$ **then**
    $model_time \leftarrow \langle\langle$time proposal of highest priority element in $T_{maybes}\rangle\rangle$
    $\langle\langle$move trans. from T_{maybes} to $T_{unknowns}$ with time = $model\_time$ $\rangle\rangle$
  end if
end while

Algorithm 2: CPN simulation transition occurrence scheduling algorithm which handles models with or without time.

3.1 Multi-Sets without Time

We first treat the more simpler case of multi-sets without time. In Sect. 3.2 we extend the data-structure proposed in this section so that timed multi-sets also can be handled efficiently.

A number of standard data-structures has been investigated and empirically compared. These comparisons can be found in [5]. A few data-structures were not even considered for implementation. Hash tables were one of these which are otherwise a popular choice for many different applications. The biggest obstacle with these is to devise a hash function which is sufficiently general to cover a large variety of multi-sets. Another obstacle is that allocation of each hash table will take up a lot of memory. One solution could be to let the user specify the hash function, but it would require advanced SML experience and hence make our tools more complicated to use.

3.1.1 Bounded Balanced Trees

In the new simulator the choice of data-structure for representing multi-sets fell on a kind of trees called bounded balanced trees (abbreviated as BB-trees). The implementation of BB-trees is based on work by Adams [1], Mehlhorn [12], and Nievergelt and Reingold [16]. In [5] empirical data is provided which suggests that BB-trees has the most favourable time complexity of all the candidate data-structures. The experiments surprisingly suggest that BB-trees are competitive even for small multi-sets.

BB-trees are in the family of weight-balanced binary trees, which is different from the height-balanced tree family such as splay-trees and 2-4-trees. There is the following invariant for all subtrees in a BB-tree, BBT:

$$1/\alpha \leq \frac{|BBT_l|}{|BBT_r|} \leq \alpha$$

where BBT_l and BBT_r are the left and right siblings respectively, and α is called the weight ratio (and sometimes results in the name BB[α]-trees).

A useful property of the weight-balanced tree family (in contrast to height-balanced trees) is that they are easier to re-balance. This is important in our case because we expect that tokens are frequently added and subtracted from multi-sets, and this suggests a lot of re-balancing during simulation. Weight-balanced trees furthermore allows a trade-off between search time and rebalancing time by choosing the weight ratio α. Adams [1] use $\alpha = 5$ and we have adopted the same ratio in the implementation without further consideration.

When an element has been inserted or deleted in a BB-tree, the tree may be out of balance, i.e., the above mentioned invariant is temporarily broken. The invariant is reestablished immediately by means of rotations of nodes, so that larger subtrees are positioned higher in the tree than smaller subtrees. Two kinds of rotations are used, namely single rotations and double rotations. These are illustrated in Fig. 2. It is important to note that a rotation preserves the order of the elements.

Functional languages, such as SML as we use here, make very elegant implementations of trees such as BB-trees. Extracts of the source code from the new simulator can be viewed in Appendix A.

3.1.2 Special Structures for Small Colour Sets

Small colour sets with only a few different elements can be treated as special cases with very efficient data-structures. Colour sets such as `unit` and `bool` are represented with special data-structures. In these cases we can do with an integer counter for each colour in the multi-set. For `unit` we only need one counter and two for `bool`. Hence searching is reduced to comparison of integers, and removal of elements is reduced to integer subtraction (similar for addition of elements).

This reduction is welcome because both `unit` and `bool` are colour sets which are used relatively often in practice. It results in even faster simulations and less usage of run-time memory.

3.2 Multi-Sets with Time

Representation of multi-sets and markings on places with timed colour sets requires that we handle time-stamps on tokens. We discuss this issue in the following.

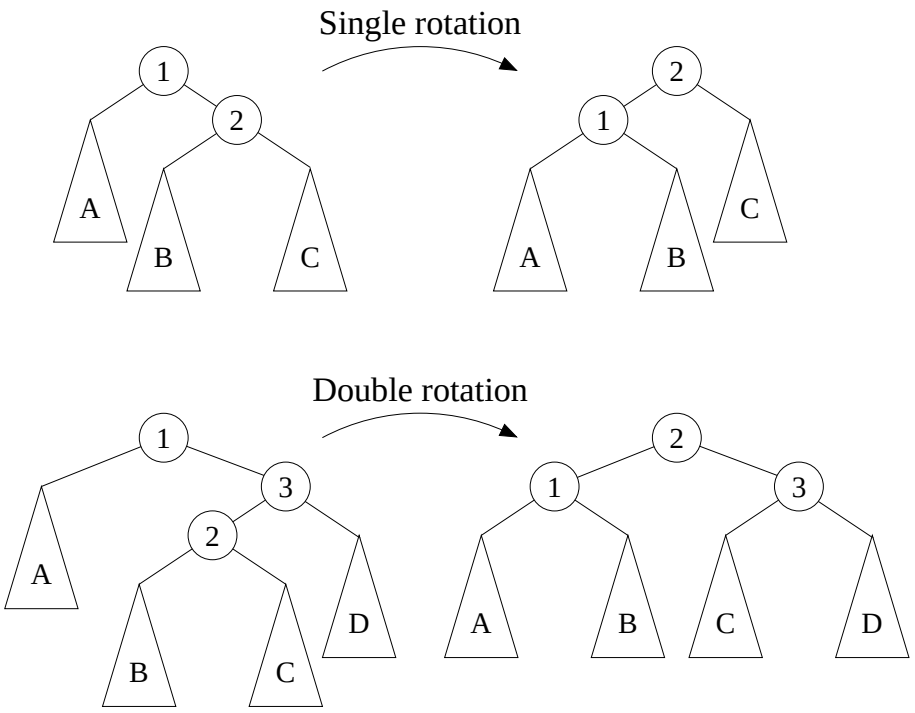


Figure 2: Rotation of nodes in BB-trees. Circles represent elements and triangles represent subtrees.

The first observation to make for timed CP-nets is that a state at a given moment is a marking together with the model time. Tokens with time-stamps greater than the model time cannot participate in the calculation of enabled binding elements. They are in some sense hidden from the simulator and waiting in a queue. When we need to estimate at what time a transition can be enabled we need to find tokens with the smallest time-stamp (see details in Sect. 4.2). This suggests that we need an additional data-structure which can hold the tokens with time-stamps greater than the model time. A priority queue is a useful structure for this case because we can lookup the element with the highest priority (lowest time-stamp) in constant time. Tokens held in the priority queue are called *waiting tokens*.

Tokens with time-stamps less than or equal to the model time are called *ready tokens*. For these we simply use the BB-tree representation from before. Note that we are only interested in the ready tokens when we search for an enabled binding, and hence can ignore waiting tokens in this case.

In summary the marking of a timed place is, in the new simulator, represented by two data-structures. One for ready tokens (BB-trees) and one for waiting tokens (priority queue). When model time progresses, we move tokens from the waiting structure to the ready structure, so that only tokens with time-stamps greater than the model time is left in waiting.

As a new feature (i.e., not present in the old simulator) we can also handle non-zero time-stamps on input arc expressions. Consider the example in Fig. 3. For instance, if `time_exp` is a positive value, we need to move more tokens from waiting to ready of the place. We in some sense look into the future of the marking of the place. This is simple and efficient to implement in the case where `time_exp` is a constant, because we only need to add the model time and `time_exp`, and then move tokens from waiting in the usual fashion. If a place has more output arcs with non-zero constant time-stamps we simply use the largest value for reasons of efficiency and simplicity. This is a safe approximation to make and the implementation is not further complicated because the binding algorithm already takes into account the time-stamps of tokens. In the general case where

`time_exp` is not constant we have not yet found an efficient implementation. This is left to future work.

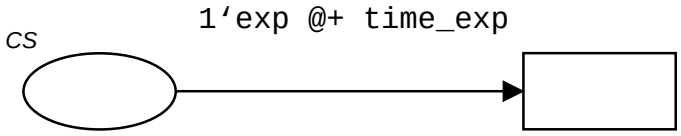


Figure 3: CP-net where the arc has an expression with a time-stamp.

4 Calculation of Enabled Transition Bindings

In this section we describe how the new simulator finds enabled binding elements. In Sect. 2 we described that the simulator is using interleaving semantics. This simplifies the implementation because we then know that at most one transition at any moment will access the data-structures of the places.

The implementation of the binding calculation is directed by a number of design criteria which are listed in the following:

- The first enabled binding we find will immediately be used for occurrence. For this to work we also need to ensure that the search algorithm is, in some sense, fair so that every enabled binding has a non-zero probability of being used.
- Variables in arc expressions and guard of a transition are grouped so that variables in different groups can be bound independently. For instance, two variables in the same arc expression are dependent with respect to binding and therefore grouped. Taking the transitive closure of this dependency relation induces independent groups of variables which, as we explain later, may result in significantly fewer binding combinations to search for.
- Instead of making a single general binding algorithm which can handle all kinds of transitions during simulation, we generate an individually optimised *binding recipe* (procedure for binding variables) for each transition. As an additional optimisation we only generate a binding recipe for each of the groups mentioned above which reduces the complexity of optimising recipes.

In the following we elaborate on the criteria above, and also divide the description into two sections which cover the case without and with time respectively.

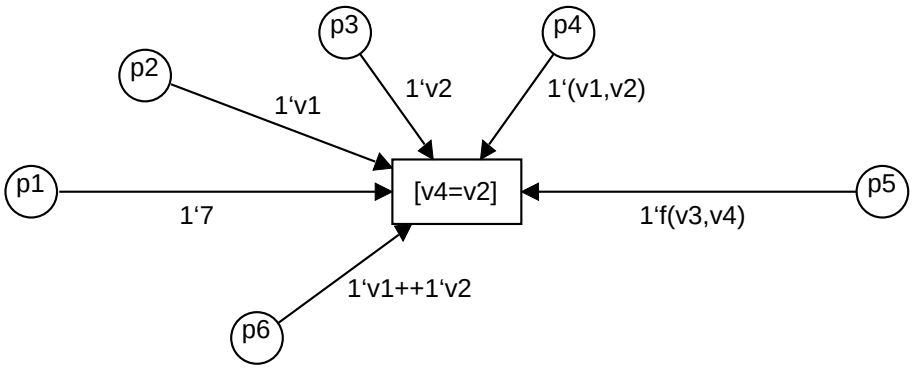


Figure 4: CP-net with different expression classes.

4.1 Binding Calculation without Time

The central task of the binding calculation is to bind each variable of a transition to a value. Our approach is to make a dependency analysis of expressions and variables for a transition, and then generate a sequence of binding operators which constitute a recipe (or procedure) for making a complete binding. The goal is that we, for each transition, can take such a sequence and automatically translate (compile) it into SML source code. This is a very powerful approach and permits us to make optimisations analogous to those used in compilers.

4.1.1 Binding Language

The analysis of arc and guard expressions induces a classification. The classification is dependent on the expression language used, which in this case is SML. Consider a number of different expression classes of Fig. 4. Each class requires a different technique for binding variables. We identify these based on the mentioned example:

- On the arc from the place $p1$ there is a constant value expression. In this case we only need to check that the value exists on $p1$.
- The expressions on the arcs from $p2$, $p3$, and $p4$ are all SML patterns. The variables can be bound by pattern matching with a random token from the place markings.
- With the expression on the arc from $p4$ we could alternatively already have bound one of the variables, say $v1$, by means of another expression. In this case we need to bind $v2$ by looking up tokens by means of the known value of $v1$. This is also called key-lookup.
- On the arc from $p5$ there is a function f with two variables as argument. In this case we must bind the two variables and subsequently test if the token $f(v3, v4)$ exists on the place.
- On the arc from place $p6$ we have an expression which is dividable in the sense that we can split the arc into two, one for each addend.
- The guard can be used to bind either $v4$ or $v2$ if one of them is already bound. If both variables are bound we can use the guard to test if the binding is enabled.

We can use the above observations to identify six different binding operators, which are summarised in Table 1. These are the binding operators we use in the new simulator in order to create recipes for binding transitions.

Operator	Purpose
$B_P\ pat$	Bind variables in the pattern *pat* by choosing a random token on the place.
$B_K\ pat$	Bind a subset of variables in the pattern *pat* with key-lookup by means of the values of the already bound variables.
$B_C\ pat$	Bind a subset of variables in the pattern *pat* in case they are of a small colour set (unit, bool, enumeration, index, and subsets), simply by an exhaustive search through all values of the colour set.
$T_A\ exp$	Assuming that all variables in the expression *exp* are bound, we test if the value of *exp* exists on the place.
$B_G\ pat = exp$	In the guard: Bind variables in pattern *pat* by means of the expression *exp*.
$T_G\ exp$	In the guard: Assuming that all variables in the expression *exp* are bound, we test if the guard is true for the given binding.

Table 1: Binding operators and their purpose.

Table 1 does actually cover all kinds of expressions we use, however this is not so obvious from the table. We have included two figures which makes this more clear. The figures illustrate the decision trees we use in order to determine the appropriate binding operation for a given expression. Figure 5 depicts the decision tree for all arc expression classes while Fig. 6 depicts the decision tree for all guard expression classes. In the next section we describe how to create sequences of operators in order to make a complete binding recipe.

4.1.2 Binding Operator Sequence

For a given transition we can usually construct several sequences of binding operators which completely binds all variables. Some sequences are more efficient than others. In this section we describe how we can identify an efficient binding sequence. It is, however, an open question how to find the optimal sequence.

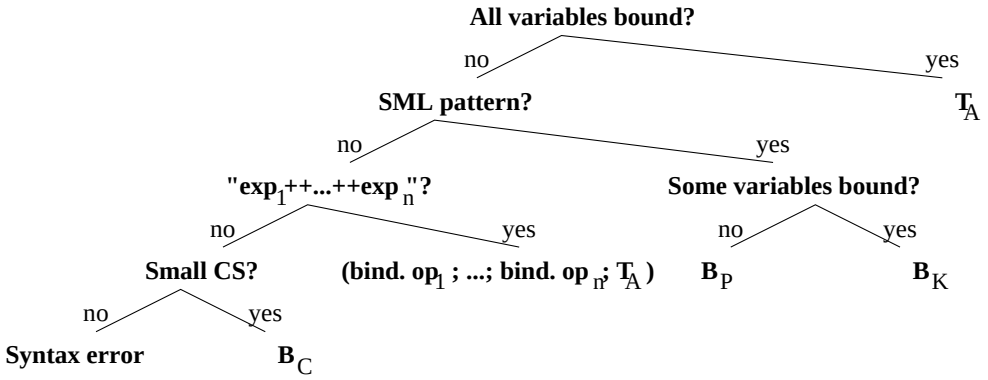


Figure 5: Decision tree for the class of arc expressions. Given an expression we answer the questions in order to find the appropriate binding operator.

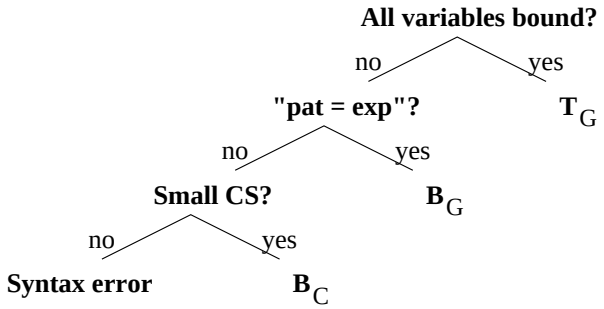


Figure 6: Decision tree for the class of guard expressions. Given an expression we answer the questions in order to find the appropriate binding operator.

The idea is to make a heuristic based on our experience with binding operators which are more successful than others. We can control this by assigning a weight to each binding operator and then consider the binding sequence problem as an optimisation problem. The problem is then reduced to finding the binding sequence which has the lowest sum of weights. The weights we have good experience with so far are as follows (a weight of 1000 is not meant to be exactly a thousand times worse than a weight of 1 but rather suggests that we wish to emphasise that B_C usually should appear last in a binding sequence):

Operator	Weight
$B_P\ pat$	100
$B_K\ pat$	20
$B_C\ pat$	1000
$T_A\ exp$	10
$B_G\ pat = exp$	1
$T_G\ exp$	2

For finding all the binding sequences we are interested in we surprisingly discovered that 1-safe P/T Nets [15] can be applied. The P/T Net is constructed as follows:

- For each input arc and guard expression we have a marked place, and for each variable we have an empty place in the P/T Net. A marked expression place means that the expression has not been used yet for binding, and a marked variable place means that the variable has been bound.
- We add a transition for each combination of binding operators, patterns, and expressions, and then make arcs to/from the relevant places if the occurrence of the transition results in an expression being used or a variable being bound.

Based on the resulting P/T Net we construct its full state space. We can then extract all possible binding sequences by selecting those paths in the state space which starts in the initial marking (no expressions used and no variables bound), and ends in a state where all variables are bound and all expressions used.

We provide an example on how to find the binding sequence with the smallest weight in the next section after we have described variable groups.

4.1.3 Variable Groups

We can use variable groups in order to split a binding sequence into independent sequences so that we can reduce the binding search space significantly. We make a relation between variables so that groups of variables can be bound independently. Assume that we have two independent variables, $v1$ and $v2$. Without grouping we need, in general, to check all possible values for $v2$, for each possible value of $v1$. This means we need to check $|v1||v2|$[1] combinations in the worst case. If we can bind the two variables independently of each other we simply just bind, say, $v1$ first and then $v2$. In this case we only need to check $|v1| + |v2|$ combinations in the worst case. We explain the variable grouping relation in the following.

The main idea of the relation between variables is to let two variables be in relation if they appear in the same expression. Making the transitive closure of the relation (taking into account relations across different expressions) results in an equivalence relation which partitions the set of variables. An equivalence class of variables induces a group of expressions where these variables occur. Each group of expressions can be bound independently by a binding sequence. Hence we apply the binding sequence construction method described in Sect. 4.1.2 on each group of expressions.

Consider the example in Fig. 7. We observe that the transition of the CPN model has two variable groups, namely $\{x1, y1\}$ and $\{x2, y2, z2\}$. The example in the figure considers the former group, and hence the P/T Net and state space depicted in the figure is for this group. From the state space we can see that two binding sequences are possible: One with weight 110 and the other with weight 120. Since we search for the binding sequence with the lowest weight we choose binding sequence 1 for the transition. This is also the choice we would have made should we construct the binding sequence manually.

Notice that without variable groups in the example of Fig. 7 we would, in the worst case, need to search through all binding combinations for $x1$, $y1$, $x2$, $y2$, and $z2$, and for each check

[1] By $|variable|$ we here mean the size of the domain of *variable*, i.e., the number of different values that *variable* can be bound to.

if the binding is enabled. In this case we need to search through $2^5 = 32$ bindings. If we take into account the variable groups we should only check $2^2 + 2^3 = 12$ combinations, and with the resulting binding sequence we only need to search $2^2 + 2^2 = 8$ combinations. Imagine that the colour set `B` instead was integers from 1 to 1000. This would result in a combinatorial explosion when searching for bindings without variable groups and binding sequences.

We use binding sequences to automatically generate an implementation of the binding function of each transition. In Appendix B we have extracted some of the code generated for the example from Fig. 7.

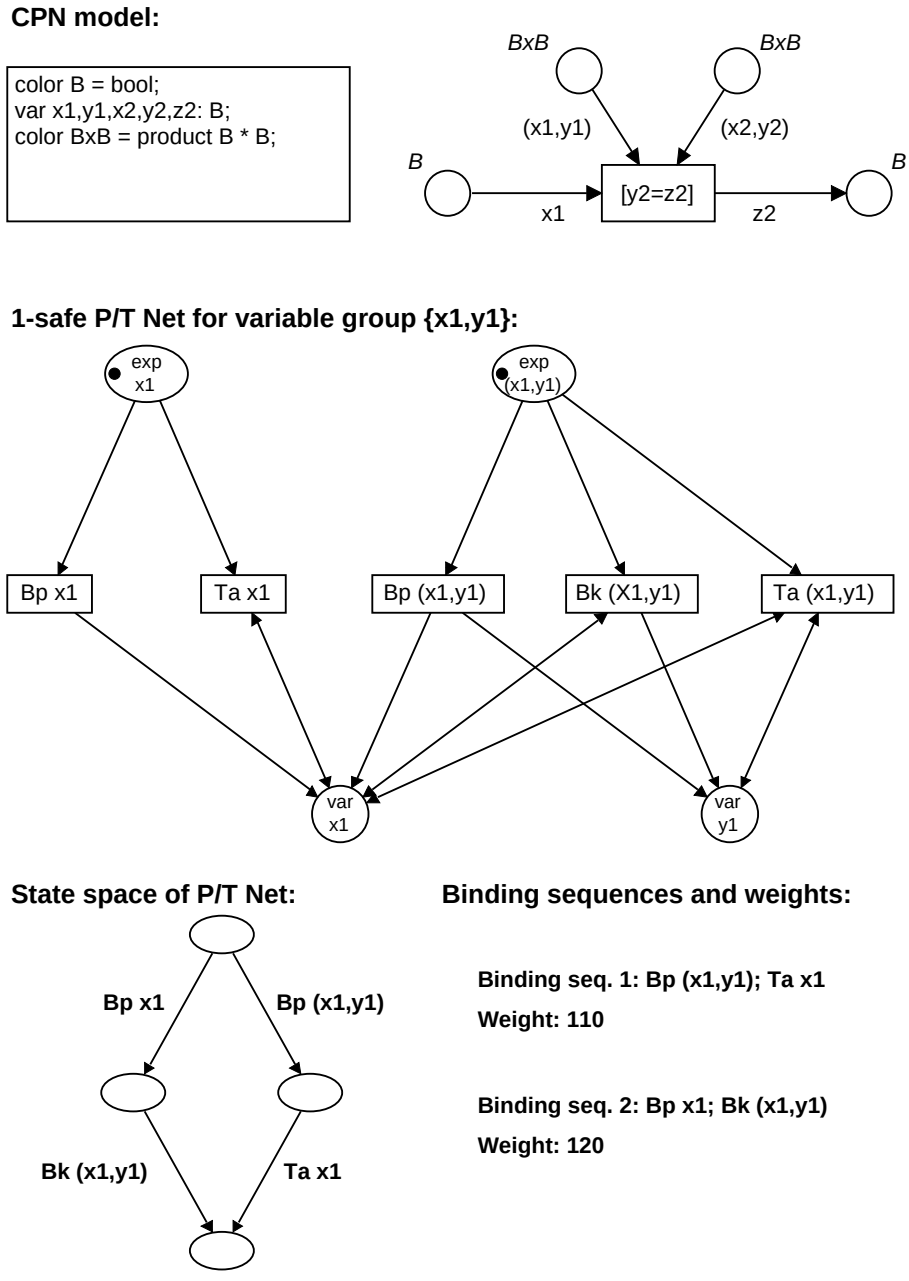


Figure 7: CPN model, the P/T-net for determining binding sequences, the state space, and the resulting sequences with weights.

4.2 Binding Calculation with Time

Binding variables for timed CP-nets is just as binding variables for CP-nets without time. The only difference is when we cannot find a binding. In this case we need to calculate a time value at which the transition *may be* enabled. This time value is needed by the scheduler algorithm as explained in Sect. 2.

If we wish to calculate the exact time value at which the transition is enabled we need in the worst case to calculate all bindings in order to find the combination of tokens and time-stamps which results in the smallest increase of the model time. This is contradicting one of our design goals for the binding algorithm that we only wish to use the first enabled binding we discover and then make it occur in the next step.

Instead we present an approximation for solving this problem. Observe that we can make a guess at the model time at which a transition is enabled without causing any harm, as long as it is smaller than the actual time at which it really is enabled. Recall from Sect. 3.2 that each timed place has a structure with waiting tokens with time-stamps greater than the model time. Consider a transition and lookup the token in the waiting structure on each input place which has the lowest time-stamp (highest priority). As we described earlier, this can be done in constant time for each place. We then find the lowest time-stamp among the set of time-stamps just found, and use this value as an approximation for the next model time at which the transition is potentially enabled. In Sect. 2.2 we explained how this approximation value is used in the scheduler.

In the old simulator all enabled binding elements are calculated in order to determine accurately the smallest value of the model time at which there is an enabled transition. This can for some CP-nets be rather time consuming, while for the new simulator we only make a fast approximation based on token time-stamps on input places. Hence the new simulation scheduler is a big improvement for timed CP-nets.

5 Related Approaches

In this section we make a short survey of related work. There are several papers on simulation data-structures and algorithms for various kinds of high-level Petri Nets. Mäkelä made recently a query on the PetriNets Mailing List [17] which resulted in several references. We discuss some of these in the following.

Mäkelä [11] calculates enabled binding elements by means of a unification technique. The kind of Petri Nets used is Algebraic System Nets which have similar challenges when searching for enabled bindings as with the case of CP-nets. Unification is in some sense a more systematic approach than ours which is based on heuristics on finding optimised binding operator sequences.

Sanders [19] views the problem of finding enabled binding elements as a constraint satisfaction problem. The kind of Petri Nets used is CP-nets, but expressions on input arcs are unfolded to elementary multi-set expressions on the form $n`exp$. This is not possible for arbitrary SML functions. In our tools we permit arbitrary SML functions as arc expressions We think that the required unfolding to elementary multi-set expressions is too restrictive for practical purposes.

Gaeta [4] has studied algorithms for Stochastic Well-Formed Nets. Among other techniques, he uses a heuristics for determining if a transition is disabled. Tokens are counted on input places and compared with the number of tokens removed from input arcs. The transition is certainly disabled if there are not a sufficient amount of tokens on the place compared with the number of tokens to be removed via the arc. We have developed the same technique, and we find it to be a very efficient technique with a simple implementation.

Ilie et al. [7] use, among other things, caching techniques on enabled binding elements for simulating Well-Formed Nets. We do not use any caching techniques in our work, although the technique could be used with CP-nets. We do not use caching now but have considered to do it. The drawback with caching is that it is very difficult to implement without compromising fair choice of bindings.

Reinke [18] translates CP-nets to the functional programming language called Haskell. How-

ever, there is no focus on optimised algorithms and data-structures but rather a demonstration of Haskell used as both an inscription language and a language for simulator implementation. We use SML for the same purpose. Kummer et al. [10] use Java both for inscriptions and simulator implementation.

6 Conclusion

This work demonstrates that the application of sophisticated data-structures for place markings and algorithms for transition binding and scheduling have significantly improved the performance of our CPN simulation tools. We have implemented and released the simulator together with Design/CPN and CPN Tools.

The simulator has been applied successfully in an industrial project. It is a project accomplished together a Danish security company, Dalcotech, where we have used Design/CPN with the new simulator to automatically generate an implementation of an access control system based on the CPN model of the system [14]. In the project it was a great advantage to use a fast running simulator because the automatically generated implementation was required to run on hardware with limited processor power.

Future Work

An ongoing project is a re-design project similar to the one described in this paper. A new state space tool is being designed where we expect that sophisticated algorithms and data-structures will result in significant performance improvements. The state space tool will use the new simulator and even get a new interface when we build it into the CPN Tools software package.

We are also working on improving the syntax check and simulation interface for the new simulator in the CPN Tools software package. In particular we are currently working on incremental syntax checking of models. The idea is that syntax checking should be transparent to the user so that the tool in principle always is ready to simulate immediately after the user has made changes to a CPN model. Incremental syntax checking has already been used in CPN Tools by students in a course where CPN Tools is used, and it is clear that the immediate feedback when editing a model is important for their success as novices.

References

[1] S. Adams. Functional Pearls, Efficient Sets — A Balancing Act. *Journal of Functional Programming*, 3(4):553–561, 1992.

[2] M. Beaudouin-Lafon, W.E. Mackay, M. Jensen, P. Andersen, P. Janecek, M. Lassen, K. Lund, K. Mortensen, S. Munck, A. Ratzer, K. Ravn, S. Christensen, and K. Jensen. CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets. In M. Koutny and J.-M. Colom, editors, *22nd International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Newcastle upon Tyne, United Kingdom, June 2001. Springer-Verlag. To appear.

[3] Design/CPN Online. WWW Site. URL: www.daimi.au.dk/designCPN.

[4] R. Gaeta. Efficient Discrete-Event Simulation of Colored Petri Nets. *IEEE Transactions on Software Engineering*, 22(9):629–639, September 1996.

[5] T.B. Haagh and T.R. Hansen. Optimising a Coloured Petri Net Simulator. Master's thesis, University of Aarhus, Department of Computer Science, Denmark, 1994. www.daimi.au.dk/CPnets/publ/thesis/HanHaa1994.pdf.

[6] M.R. Hansen and H. Rischel. *Introduction to Programming Using SML*. Addison-Wesley, 1999.

[7] J.-M. Ilié and O. Rojas. On Well-formed Nets and Optimizations in Enabling Tests. In M.A. Marsan, editor, *14th International Conference Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 300–318, Chicago, Illinois, USA, June 1993. Springer-Verlag.

[8] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 1992.

[9] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods*. Monographs in Theoretical Computer Science. Springer-Verlag, 1994.

[10] O. Kummer. Simulating Synchronous Channels and Net Instances. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, *Forschungsbericht: 5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pages 73–78. Universität Dortmund, Fachbereich Informatik, 1998. Published as Forschungsbericht: 5. Workshop Algorithmen und Werkzeuge für Petrinetze, number 694.

[11] M. Mäkelä. Optimising Enabling Tests and Unfoldings of Algebraic System Nets. In M. Koutny and J.-M. Colom, editors, *22nd International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Newcastle upon Tyne, United Kingdom, June 2001. Springer-Verlag. To appear.

[12] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1984.

[13] K. H. Mortensen, S. Christensen, L.M. Kristensen, and J.S. Thomasen. Capacity Planning of Web Servers using Timed Hierarchical Coloured Petri Nets. In *HP Openview University Association (HP-OVUA'99) 6th Plenary Workshop*, Bologna, Italy, 1999.

[14] K.H. Mortensen. Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System. In M. Nielsen and D. Simpson, editors, *21st International Conference on Application and Theory of Petri Nets*, volume 1825 of *Lecture Notes in Computer Science*, pages 367–386, Aarhus, Denmark, June 2000. Springer-Verlag.

[15] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[16] J. Nievergelt and E.M. Reingold. Binary Search Tree of Bounded Balance. *Siam Journal of Computing*, 2:33–43, 1973.

[17] Petri Nets World. WWW Site. URL: www.daimi.au.dk/PetriNets.

[18] C. Reinke. Haskell-Coloured Petri Nets. In *Implementation of Functional Languages, 11th International Workshop, IFL'99*, volume 1868 of *Lecture Notes in Computer Science*, pages 165–180, Lochem, The Netherlands, September 1999. Springer-Verlag.

[19] M.J. Sanders. Efficient Computation of Enabled Transition Bindings in High-Level Petri Nets. In *Proceedings of the 2000 IEEE International Conference on Systems, Man and Cybernetics*, pages 3153–3158, Nashville, TN, USA, October 2000.

A SML Listing for Multi-Set Data-Structures

In the following we list some SML fragments which presents a basic implementation of multi-sets with BB-trees. The background for the implementation can be found in Sect. 3. The listings have been edited for presentation purposes. Code has been left out in locations where "..." is used.

First we declare a recursive BB-tree data-type, and a useful function for creating new nodes:

```
datatype 'a tree = TreeNil
                 | TreeNode of {value: 'a, size : int,
                                left : 'a tree, right : 'a tree}
fun new(v,l,r) = TreeNode{value= v, size= 1+(size l)+(size r),
                          left = l, right= r}
```

Single and double clockwise rotation has simple functions. Similarly for counterclockwise rotation (not listed here):

```
fun single_rotate_cw
    (y, TreeNode{value=x,left=s1,right=s2,...}, s3)
  = new(x, s1, new(y,s2,s3))

fun double_rotate_cw
    (z, TreeNode{value=x,
                 left =s1,
                 right=TreeNode{value=y,left=s2,right=s3,...},
                 ...},
     s4)
  = new(y, new(x,s1,s2), new(z,s3,s4))
```

While inserting or deleting elements in the BB-tree we may find that the tree is out of balance and we then call one of the above rotation functions. The value `ratio` is the weight ratio α as we described in Sect. 3.1.1.

```
fun balance (p as (v,l,r)) =
    let
      val ln = size l
      val rn = size r
    in
      if ln+rn < 2 then
        new p
      else if rn > ratio * ln then
        let
          val (rl,rr) = sons r
        in
          if size rl < size rr
          then single_rotate_ccw p
          else double_rotate_ccw p
        end
      else if ln > ratio * rn then
        let
          val (ll,lr) = sons l
        in
          if size lr < size ll
          then single_rotate_cw p
          else double_rotate_cw p
          end
      else
        new p
    end
```

The `balance` function is called immediately after each addition or deletion of an element in the tree.

B SML Source Code for a Small Example Model

The following source code listing is the simulation code automatically generated for the example model in Fig. 7. The idea is to give the reader an impression of the structure of the automatically

generated code. The listing has been edited for presentation purposes. Code has been left out in locations where "..." is used. We have also added extra comments with the usual SML notation "`(* ... *)`".

```
(* colour set B = bool *)
structure B = CPN'ColorSets.BoolCS (val arg= NONE);
type B = B.cs;
(* BB−tree for B *)
structure CPN'B_pims1 = CPN'BoolPIMS(structure cs = B);

(* colour set BxB = product B * B *)
type BxB = {1:B,2:B}
structure BxB = struct
 type cs = BxB
  ...
end;
(* BB−tree for BxB *)
structure CPN'BxB_pims21 =
 CPN'MakeTreeListPIMS(structure cs = BxB;
                      val cmp:BxB * BxB -> order = ... );

(* Places and initial markings *)
structure CPN'place4 =
 CPN'Place.MakePlace (structure ims = CPN'BxB_pims21; val no_of_inst = 1);
val _ = (CPN'place4.init_mark:= ([]);
         CPN'Place.init_mark_funs::= CPN'place4.set_init_mark;
         CPN'place4.set_init_mark());
 ...

(* The transition *)
fun CPN'transition8 (CPN'mode,CPN'inst) =
 let
  val CPN'id = "8"
  (* Storage for binding of group {x2,y2,z2} *)
  val CPN'bh1 = ref(nil: {z2: B,y2: B,x2: B} list )
  (* Storage for binding of group {x1,y1} *)
  val CPN'bh2 = ref(nil: {y1: B,x1: B} list )
  (* First check if there are enough tokens on the input places *)
  val (CPN'enough_tokens, CPN'answer) =
      (CPN'Sim.each_place
       (1 <= CPN'place6.init(CPN'inst),
        CPN'Sim.each_place(1 <= CPN'place7.init(CPN'inst),
                           CPN'Sim.each_place(1 <= CPN'place4.init(CPN'inst),
                                              (true,CPN'Sim.is_disabled)))))

  (* Function for searching for an enabled binding *)
  fun CPN'bindfun () =
   let
    fun CPN'bf3() = ()
    fun CPN'bf2() =
     let (* B_p (x1,y1) binding operator *)
      val _ = CPN'BxB_pims21.init_res(CPN'place6.mark CPN'inst)
      fun CPN'bf() =
       let
        (* Pick random token from place *)
        val (x1,y1) =
            CPN'BxB_pims21.random_res BindFatalFailure
                                      (CPN'place6.mark CPN'inst)
       in
```

```
          ((* T_a x1 *)
           (if CPN'B_pims1.member (!(CPN'place7.mark CPN'inst),x1) then
               (CPN'bh2::= {y1=y1,x1=x1}; (* binding found *)
                CPN'bf3())
            else raise BindFailure
           ) handle BindFailure => CPN'bf()
          )
         end handle Bind => CPN'bf()
        in
         CPN'bf()
        end
       fun CPN'bf1() =
        let (* B_p (x2,y2) binding operator *)
         val _ = CPN'BxB_pims21.init_res(CPN'place4.mark CPN'inst)
         fun CPN'bf() =
          let
           (* Pick random token from place *)
           val (x2,y2) =
               CPN'BxB_pims21.random_res BindFatalFailure
                                         (CPN'place4.mark CPN'inst)
          in
           ((* B_g z2=y2 binding operator *)
            (let
              val z2 = y2
             in
              (CPN'bh1::= {z2=z2,y2=y2,x2=x2}; (* binding found *)
               CPN'bf2())
             end handle Bind => raise BindFailure
            ) handle BindFailure => CPN'bf())
         end handle Bind => CPN'bf()
        in
         CPN'bf()
        end
      in
       CPN'bf1()
      end

     (* Function for making an enabled binding occur *)
      fun CPN'occfun ({z2,y2,x2},{y1,x1}) =
       let
        (* Remove tokens from input places *)
        val _ = (CPN'BxB_pims21.delete(CPN'place6.mark CPN'inst,(x1,y1));
                 CPN'B_pims1.delete(CPN'place7.mark CPN'inst,x1);
                 CPN'BxB_pims21.delete(CPN'place4.mark CPN'inst,(x2,y2)));
        (* Add tokens to output places *)
        val _ = (CPN'B_pims1.insert(CPN'place5.mark CPN'inst,z2));
       in
        (CPN'Sim.is_executed,
         ...)
       end

     in (* CPN'transition8 *)
      if CPN'enough_tokens then
       (CPN'bindfun();
        CPN'occfun (CPN'hd(!CPN'bh1),CPN'hd(!CPN'bh2))
       ) handle BindFatalFailure => (CPN'answer,nil)
      else (CPN'answer,nil)
     end
```

Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN

Guy Gallasch and Lars Michael Kristensen

Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
Email: {galgy002@students.unisa.edu.au,lars.kristensen@unisa.edu.au}

Abstract. In this paper the development of Comms/CPN is presented. Comms/CPN is a Standard ML library that augments Design/CPN with the necessary infrastructure to establish communication between CPN models and external processes. Comms/CPN is potentially beneficial in a number of areas such as allowing external visualisations of simulations, providing CPN models with their own Graphical User Interface, and allowing CPN models to interact with the physical environment. Comms/CPN has been successfully applied for providing external visualisation of the simulation of a CPN model within the area of avionics mission systems.

1 Introduction

Coloured Petri Nets (CPNs) [11,13], when constructed and simulated using the Design/CPN tool [21], are restricted in their ability to interact with external processes. Extending the Design/CPN tool by providing a communication infrastructure allows communication to be established between CPN models and external processes. The Comms/CPN library [6] presented in this paper has been developed to extend Design/CPN with such external communication facilities.

The motivation behind developing external communication facilities comes from the desire to visualise the simulation of CPN models. As demonstrated in [2,23], it is often beneficial to extend CPN models with application specific graphics. The behavior of the system under consideration can be visualised using different kinds of graphical feedback. This provides a view of system behavior useful for system developers and analysts, and can also be useful for conveying knowledge and results about CPN models to people not familiar with CPN modelling and analysis.

Currently, Design/CPN provides visualisation capabilities in two forms. The first is the token game which displays the simulation of a CPN model in a very detailed fashion. The second is by using high-level application specific graphics that can be added on top of a CPN model. Mimic/CPN [1] and the Message Sequence Chart library [18] provide such high-level graphics. However these methods are not always satisfactory. The token game is often too detailed, and the application specific graphics are sometimes limited in capability and capacity, and are tied to the Graphical User Interface (GUI) of Design/CPN. It is therefore of interest to conduct visualisation using an external application. External applications can be developed with greater graphical capabilities than those of Design/CPN, and there is the potential to execute the external application on a remote machine.

The Comms/CPN library has been developed to allow communication between Design/CPN and external processes via TCP/IP [4]. The main benefits that the Comms/CPN infrastructure will provide in the context of visualisation are:

- The infrastructure makes it possible to visualise the behavior of CPN models and control their simulation independently of the Design/CPN GUI.
- The infrastructure provides flexibility, since other graphical libraries and packages are likely to provide better support for visualisation than Design/CPN.
- The infrastructure makes it possible to do the visualisation on remote machines provided they support TCP/IP communication.

It should be stressed that Comms/CPN is not limited to use in external visualisation. Comms/CPN has the potential to be beneficial in many other areas. As an example, it could be used to provide CPN models with their own GUI. The Design/CPN simulator is built on the Standard ML (SML) [20,27] compiler and Comms/CPN is also implemented in SML. This means that the Design/CPN GUI could be separated from the simulator, and a GUI specific to the CPN model could be used instead. An example where this may be useful is when applying CPN models in decision making processes, as shown in [15]. Using Comms/CPN, it is also conceivable that CPN models could interact with the physical environment. Examples include temperature and light sensors, keypads, and displays (although such experiments have not yet been conducted). Situations may also arise where computationally expensive algorithms and procedures are needed within Design/CPN. With Comms/CPN, these can be implemented and executed on remote machines, and the results can be sent back to the CPN model. An example of this can be found in [17] where the condensed state space tool of Design/CPN [12] relied on the GAP programming enviroment [7] for efficient manipulation of algebraic groups. More generally, Comms/CPN makes it possible to integrate Design/CPN and external applications via TCP/IP.

Comms/CPN was developed as a Practical Industrial Experience project in Computer Systems Engineering at the University of South Australia. The development is part of a research project being undertaken by the Air Operations Division (AOD) within the Australian Defence Science and Technology Organisation (DSTO) [24] and the Computer Systems Engineering Centre (CSEC) [3] at the University of South Australia. It involves the modelling and analysis of Avionics Mission Systems (AMS) for testing and evaluation. Part of this research involves providing visualisation of the simulation of CPN models by extending them with application specific graphics. The external communication facilities provided by Comms/CPN allow this visualisation to take place using an external visualisation package. The external visualisation package itself is currently in the process of being developed, but a prototype demonstrating a proof-of-concept exists.

The development of Comms/CPN is based upon previous work, in particular the Master's thesis [19]. The work done in [19] however has some drawbacks as it was primarily an encapsulation of TCP/IP. Only one connection could be opened, and this connection is made to a location fixed at compile time, i.e. it could not be changed without re-switching the CPN model. Comms/CPN extends the work presented in [19] in several ways. Firstly, Comms/CPN allows dynamic creation of connections (also during the simulation of a CPN model), the external process to which connections are being made is not fixed, and multiple simultaneous connections are supported. Secondly, Comms/CPN implements a protocol on top of TCP/IP for passing messages between Design/CPN and the external application. It is planned to make Comms/CPN available for public use via the Design/CPN home page [21].

This paper is organised as follows. Section 2 provides a description of the design and requirements of the Comms/CPN library. Section 3 describes the implementation of the

Comms/CPN library. An example of the use of Comms/CPN for visualisation of the simulation of an AMS CPN model is presented in Section 4. Section 5 sums up the conclusions and outlines future work in further developing Comms/CPN. The reader is assumed to be familiar with CPN models and the Design/CPN tool.

2 Design Overview and Requirements

Comms/CPN is designed to act as an interface between CPN models and TCP/IP. Figure 1 shows the overall architecture of Comms/CPN and how it relates to Design/CPN and TCP/IP. Comms/CPN consists of three main modules, organised as layers. The *Communication Layer* contains the interface to the underlying transport protocol, in this case TCP/IP, and contains all TCP/IP and socket related primitive functions. The *Messaging Layer* is responsible for transforming the reliable byte stream service provided by the transport layer into a service suitable for passing *messages* between Design/CPN and external applications. The *Connection Management Layer* allows users to open, close, send to, and receive from multiple connections. The Connection Management Layer is the layer that the CPN model will normally interface to. When relating this to the Open Systems Interconnectivity (OSI) model [26], Comms/CPN can be viewed as the session layer. The Communication Layer provides an interface to TCP/IP, the transport layer of the OSI model. The Connection Management Layer provides an interface to Design/CPN, the presentation layer.

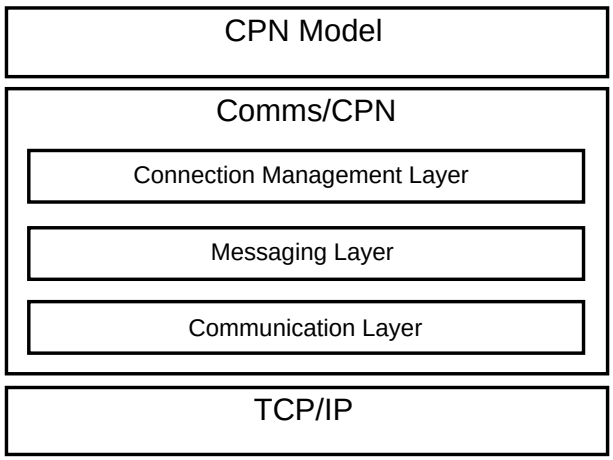


Fig. 1. Overall Architecture of Comms/CPN.

The design of all three layers has been based on five functional requirements representing the services expected of the library. They are that Comms/CPN shall provide means for CPN models to open connections to external processes, accept incoming connections requested by external processes, send data to external processes, receive data from external processes, and close connections to external processes. Comms/CPN has been designed as an SML library to reflect the non-functional requirements of allowing easy integration with the Design/CPN tool, and so that the communication facilities are separate from the Design/CPN GUI. This allows the communication facilities to be accessed using library functions that can be included in code segments of transitions, auxiliary boxes, or in the top loop of the Design/CPN simulator.

During the design and requirements processes of this architecture, three key design issues were identified. These are the *Fundamental Method of Communication*, *Connections and Connection Management*, and *Data Transfer*. The design and requirements reflect a desire to make Comms/CPN applicable to the widest range of applications possible. In the following subsections we discuss each of these design issues in detail.

2.1 Fundamental Method of Communication

The design decision was made for the underlying communication protocol to be TCP/IP. This comes from the requirement that data transmissions must be error free and in-order, and TCP/IP is a transport protocol that achieves this. TCP/IP is also desirable as it is a standard protocol, and most devices implement a TCP/IP stack and most programming environments (including SML) provide an interface to it through TCP/IP sockets.

Master thesis [19] examined the use of TCP/IP to communicate between Design/CPN and external processes. It summarised the ideas and concepts from various papers and publications. Of particular importance was [14], focussing on interaction between Design/CPN and Java processes. Two solutions were presented in [19] as to how TCP/IP communication between Design/CPN and external processes could be realised. The first was called the *Pure TCP Solution* in which Design/CPN connected directly to external applications via TCP/IP using functions in a communication library. The second was called the *Messenger Solution* in which library functions were used to communicate (via Unix pipes) with an external messenger subprocess written in Java. The Java subprocess then used TCP/IP to communicate with external processes.

Both the Pure TCP and Messenger solutions would provide usable communication facilities. The fundamental design choice for Comms/CPN was made to choose the Pure TCP solution for the design of Comms/CPN. Pure TCP provides for easier and tighter integration with Design/CPN, and can be used wherever Design/CPN is used. The Messenger solution (following the suggested implementation in [19]) does not provide for easy or tight integration because it uses a programming language other than SML. The Communication Layer reflects this decision. This layer provides an interface to TCP/IP for the Comms/CPN library. The primitives provided in this layer are used by the Connection Management Layer to establish TCP/IP connections to external processes, send and receive data in the form of streams of bytes, and to close TCP/IP connections to external processes. Moreover, it is possible to implement an architecture similar to the the Messenger solution of [19] based purely on Comms/CPN.

2.2 Connections and Connection Management

A connection represents a communication channel between a CPN model and an external process. It is the Connection Management Layer that manages connections between CPN models and external processes by creating, storing, and removing connection information. Non functional requirements of the library state that the library must be capable of handling multiple connections, and that these connections can be established dynamically during the simulation of a CPN model. Also, requirements state that the library must provide a mechanism for identifying connections and abstracting from low level socket identifiers. The design of the Connection Management Layer reflects these requirements.

Connection Identification. The connection identification strategy adopted in the design of the Connection Management Layer is to assign a unique string to each connection as it is made. String identifiers offer the advantage of being more human-readable and recognisable than an integer or a low level socket identifier. A string can be provided by the user (to further aid in recognisability) or it could be provided internally by Comms/CPN. Strings can easily be used as tokens within a CPN model to pass connection identifiers around during simulation of the CPN model.

Connection Attributes. When a connection is created, it is necessary for information about this connection to be recorded. These *connection attributes* must allow the connection to be identified and used. In order to identify the connection, the unique string identifier must be stored, and in order to use the connection, the low level TCP/IP socket identifier must be stored. The unique string identifier allows connections to be identified within CPN models, and the low level TCP/IP socket is needed in order to send and receive data. Without recording these two pieces of information, the establishment of connections becomes useless as there is no way to identify them or to use them. Before a connection is established, a check is made to ensure that the given unique string identifier is in fact unique. If not, the connection is not established. Multiple connections can be open simultaneously, so a data structure is needed to store the connection attributes of more than one connection. The Connection Management Layer contains a mechanism to do this, called the *Connection Storage Mechanism*, and a data structure in which the attributes are stored. The data structure must allow new information to be stored, existing information to be retrieved, and old information to be removed.

2.3 Data Transfer

A non functional requirement of Comms/CPN is that the library must have the capability to send and receive all types of data, including user defined types (colour sets). This is important in increasing the overall usefulness of Comms/CPN. TCP/IP dictates that data must be in the form of a sequence of bytes for transmission across a network, so data objects must be converted into this form for transmission.

The Messaging Layer of Comms/CPN within the Connection Management Layer provides a solution. Generic send and receive functions that send and receive sequences of bytes, regardless of the type of the data objects being transmitted, can be written and included in the Connection Management Layer. This provides a way for users to send and receive sequences of bytes without having the responsibility of implementing the actual sending and receiving functions themselves.

In order to convert data objects to and from sequences of bytes, encoding and decoding is necessary. An encoding function converts a data object into a form suitable for transmission via TCP/IP, and a decoding function converts a sequence of bytes into a data object. SML, being a functional programming language, allows functions to be passed as parameters. In this way, encoding and decoding functions can be written for any data type desired, and can then be passed as parameters to the generic send and receive functions. The generic send function applies the encoding function to a data object in order to convert it to a suitable form for transmission. Similarly, the generic receive function applies the decoding function to a received sequence of bytes to form data objects. In this way, any type of data can be sent or received, provided the corresponding encoding and decoding functions have been written. Encoding and decoding functions for commonly used data types are supplied with the library, e.g. for strings and integers.

Another non functional requirement is to ensure that data sent and received has a consistent format regardless of its type. The virtual byte stream service provided by TCP/IP allows for the transmission and reception of sequences of bytes. However, when dealing with many different types of data (including user defined types) this is not always adequate. A more structured approach is required to delineate items of data from the virtual byte stream to provide a better service than just a stream of bytes. Data needs to be packetised for transmission so that when an item of data is sent, the receiver knows when all of it has arrived. The solution to achieve this atomicity is to segment the stream of bytes, and to provide each segment with a header that describes it. A segment of bytes (payload) together with its header make up a data *packet*. A message is one or more of these packets.

The packet format chosen for COMMS/CPN consists of a one byte header and a maximum of 127 bytes of payload data. This is illustrated in Figure 2. Seven bits of the header indicate the length of the payload data attached to it (i.e. $2^7 - 1 = 127$ bytes) and the remaining bit indicates whether this is the last packet in the transmission of the data item, in the case where a data item is greater than 127 bits in length. In this way, the header allows variable length data packets to be handled. It must be stressed that the maximum of 127 bytes of payload data can easily be changed by choosing a different sized header. What is important is that the peer entity in the external process (with which communication is taking place) implements the same protocol at the Messaging Layer.

The choice of a one-byte header is somewhat arbitrary, as there does not appear to have been any studies conducted regarding ideal packet length when transferring data between DESIGN/CPN and external processes. It should be mentioned that other segmentation and assembly protocols could be used to achieve the same service.

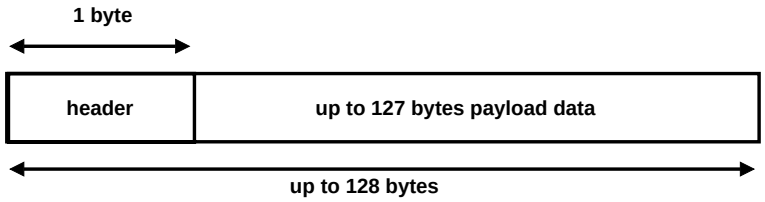


Fig. 2. Packet format for transmission of data.

3 Implementation

This section describes the implementation of the design into a working STANDARD ML [8] library. The implementation consists of SML library files, one for each of the layers described previously in Sect. 2.

3.1 The Communication Layer

The Communication Layer is implemented with TCP/IP as the underlying transport protocol. It is designed to encapsulate the TCP/IP protocol and to provide users of this layer with a shielded interface to the network functions provided by TCP/IP. The SML/NJ standard library [25] contains a structure called *Socket* in which primitive operations on sockets are available. The Communication Layer is based on this library.

Figure 3 lists the *COMMS_LAYER* SML signature. This signature is implemented by the *C*ommsLayer structure constituting the Communication Layer. We describe each of the primitives provided by the Communication Layer in more detail below.

```
signature COMMS_LAYER =
  sig
    type channel
    exception BadAddr of string

    val connect : string * int -> channel
    val accept : int -> channel

    val send : channel * Word8Vector.vector -> unit
    val receive : channel * int -> Word8Vector.vector

    val disconnect : channel -> unit
  end;
```

Fig. 3. SML signature for Communication Layer.

Most of the implementation of the Communication Layer comes directly from [25]. The only new datatype introduced is called *channel*. Its purpose is to allow the Connection Management Layer to map string identifiers to TCP/IP sockets without using any TCP/IP related code. Below we give a brief description of each of the primitives.

connect Creates a connection, acting as a client, to an external process. The first argument is used to specify the hostname of the external application, and the second argument is used to specify the port number.

accept Waits for an incoming connection on the port specified as the argument. The primitive blocks until an external application connects, and the connection is then established.

send Sends the sequence of bytes specified as the second argument on the channel specified as the first argument.

receive Receives the number of bytes specified as the second argument on the channel specified as the first argument. The primitive will block until the specified number of bytes have been received on the channel.

disconnect Closes the connection specified as the argument.

3.2 Messaging Layer

The Messaging Layer is implemented on top of the Communication Layer. Figure 4 lists the *MESSAGING_LAYER* SML signature. This signature is implemented by the *M*essagingLayer structure constituting the Messaging Layer. The *send* function implements the transmission of messages (specified as a sequence of bytes) according to the protocol discussed in Sect. 2.3. The data provided is segmented and appropriate headers are added to each segment. This forms packets of data that are sent to the external process using the *send* function from the Communication Layer. The *receive* function implements the reception of messages. It reads one byte (the header byte) and the corresponding number of payload bytes from the channel using the *receive* function from the Communication Layer. The *InvalidDataExn* exception will be raised if received data does have the format specified in Fig. 2.

```
signature MESSAGING_LAYER =
  sig
    type channel
    exception InvalidDataExn of string

    val send : channel * Word8Vector.vector -> unit
    val receive : channel -> Word8Vector.vector
  end
```

Fig. 4. SML signature for Messaging Layer.

3.3 Connection Management Layer

The Connection Management Layer builds on top of the Communication and Messaging Layers by providing the ability and interface to communicate with multiple external processes. The connection storage mechanism is implemented in this layer. The Connection Management Layer is implemented independently of TCP/IP and sockets. Instead it uses the services provided the Communication Layer and the Messaging Layer to interact indirectly with TCP/IP. It is the functions in this layer that would normally be used in a CPN model.

Figure 5 lists the *CONN_MANAGEMENT_LAYER* SML signature. This signature is implemented by the *C*onnManagementLayer structure constituting the Connection Management Layer. The signature specifies the type *Connnection* used to identity connections. The type has been implemented as strings. We describe each of the primitives provided by the Communication Layer in more detail below.

```
signature CONN_MANAGEMENT_LAYER =
  sig
    type Connection
    exception ElementMissingExn of string
    exception DupConnNameExn of string

    val openConnection : Connection * string * int -> unit
    val acceptConnection : Connection * int -> unit

    val send : Connection * 'a * ('a -> Word8Vector.vector) -> unit
    val receive : Connection * (Word8Vector.vector -> 'a) -> 'a

    val closeConnection : Connection -> unit
  end
```

Fig. 5. SML signature for Connection Management Layer.

openConnection Allows users to connect to external processes as a client. It takes three input parameters. The first of these is the unique string identifier (of type *Connection*) to be associated with the new connection. The second and third are the host name and port number that make up the address of an external process. The function first checks to ensure the string identifier given is unique, by searching the existing connections. A *DupConnNameExn* exception is raised if this is not the case. The function then attempts to create a connection to the external process by using the primitives from the Commu-

nication Layer. If successful, the appropriate information is stored and added to the list of connections. The return type of this function is type unit.

acceptConnection Provides server behaviour, and allows external processes to connect to Design/CPN. This function takes a *Connection* (string identifier) and a port number as input. The function checks that the given string identifier is unique, and then listens on the given port for incoming connection requests. This causes Design/CPN to *block* until an incoming connection request is received. When this happens, a connection is established with the external process requesting the connection.

send Allow users to send any type of data to external processes. The function is polymorphic, in the sense that the data passed to it for sending can be of any type, including user defined types. Three parameters are passed to this function as input. The first is a string identifier for the connection, the second is the data to send, and the third is a function to encode the data to send. The purpose of the encoding function is to encode the data to send into a sequence of bytes. This allows the data to be of any type, provided an encoding function exists for that type. The *send* function retrieves the connection corresponding to the given string identifier. It then invokes the *send* primitive at the Messaging Layer. The return type of this function is type unit.

receive Allows users to receive any type of data from an external process. The *receive* function is polymorphic in the same way as the *send* function. The parameters to this function are a string identifier and a decoding function, to decode the received byte vector into the appropriate data type. The function begins by retrieving the connection from which data will be received. It then invokes the *receive* from the Messaging Layer to receive the data. The payload data (which was stored in the correct order when it was read) is then passed to the decoding function. The resulting decoded data is then returned.

closeConnection Allows users to close a connection. The string identifier of the connection to be closed is passed to this function as the argument. A search of the connections is conducted to ensure that a connection exists with that string identifier. If the connection does not exist, an *ElementMissingExn* exception is raised. The connection itself is closed by calling the *disconnect* function from the Communication Layer. The stored connection information is then removed from the list of connections. The return type of this function is type unit.

4 Application of Comms/CPN

An external communication infrastructure was required as part of the research project on modelling and analysis of avionics mission systems mentioned in the introduction. Comms/CPN was developed for the purpose of providing external visualisation of the simulation of an Avionics Mission System (AMS) CPN model.

Figure 6 illustrates the architecture of the visualisation facilities, and how Comms/CPN fits into this architecture. The idea is that Comms/CPN will provide the necessary communication infrastructure to allow data to be sent from the CPN model to an external animation package which will then interpret this data and update the animation as necessary. The architecture consists of three applications (processes). The Design/CPN GUI (left), the SML process (middle), and the external Visualisation Package (right). The Design/CPN GUI and the simulator part of the SML process communicates (in the usual way) for visualising the token game during simulation in the Design/CPN GUI. This communication is done via TCP/IP using the DMO module of the Design/CPN simulator. In addition to this, the

simulator part of the SML process now also communicates with the external Visualisation Package via Comms/CPN.

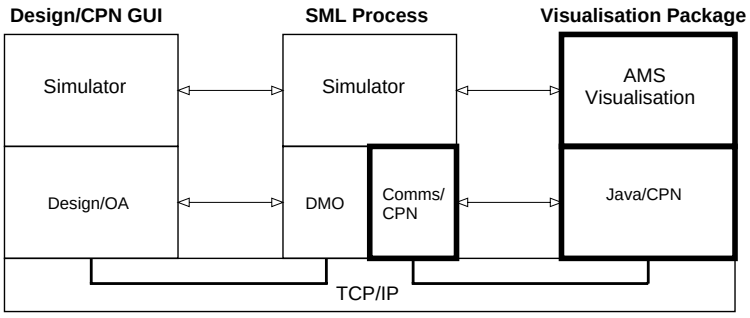


Fig. 6. Comms/CPN in the context of external visualisation.

The external visualisation package is currently under development. It is being implemented in Java [10] and consists of two main modules. The *AMS Visualisation* module is the module that provides the visualisation facilities. This module has been implemented using the Java Swing library [9]. The Java/CPN module is the peer module of Comms/CPN at the Java side. The Java/CPN module contains primitives similar to those in Comms/CPN to enable communication, and implements the protocol described in Sect. 2.3. We describe the Java/CPN module, the AMS visualisation module, and how it interacts with the AMS CPN model in more detail in the following subsections.

4.1 Java/CPN

The purpose of Java/CPN is to allow Java processes to communicate with Design/CPN through Comms/CPN. The current implementation of Java/CPN is the minimal implementation necessary to enable communication. It incorporates the equivalent functionality of the Messaging and Communication layers from Comms/CPN. The Communication Layer functionality from Comms/CPN and TCP/IP is already encapsulated in the *Socket* objects provided by Java through the use of Socket methods and the input and output streams available from the socket itself.

No connection management has been implemented within Java/CPN as this is a minimal implementation, however the important thing is that it implements the same protocol as the Messaging Layer from Comms/CPN as described in Sect. 2.3 . The interface of Java/CPN is shown in Figure 7. As in Comms/CPN, generic send and receive functions have been provided at the level of the Messaging Layer, meaning that sequences of bytes are passed to the *send* method and returned from the *receive* method. The *connect*, *accept*, and *disconnect* methods have been provided at the level of the Communication Layer from Comms/CPN. The deliberate attempt was made to make the interface as close to that of Comms/CPN as possible. We describe each of the methods within Java/CPN in more detail below.

The *connect* method acts in the same way as the *connect* method from the Communication Layer of Comms/CPN. It takes a host name and port number as arguments, and attempts to establish a connection as a client to the given port on the given host. This method does not return a value. Once the connection has been established (i.e. the socket opened) input

```
import java.util.*;
import java.net.*;
import java.io.*;

public interface JavaCPNInterface
{
  public void connect(String hostName, int port);

  public void accept(int port);

  public void send(ByteArrayInputStream sendBytes) throws SocketException;

  public ByteArrayOutputStream receive() throws SocketException;

  public void disconnect();
}
```

Fig. 7. Interface to Java/CPN.

and output streams are extracted from the socket to enable the transmission and reception of bytes.

The *accept* method also acts in the same way as the *accept* method from the Communication Layer of Comms/CPN. It takes a port number as an argument and, acting as a server, listens on the given port number for an incoming connection request. When received, it establishes the connection. Again, once the connection has been established, input and output streams are extracted from the socket to enable the transmission and reception of bytes. This method does not return a value.

The *send* method takes a *ByteArrayInputStream* object (a Java object for holding sequences of bytes, acting as input) as the argument. The segmentation into packets occurs in a similar way to that which occurs in the Messaging Layer of Comms/CPN. Bytes are read from the *ByteArrayInputStream* object, a maximum of 127 at a time, and a header added as described in Sect. 2.3. The data packets formed are then transmitted to the external process through methods acting on the output stream of the socket. The *send* method does not return a value.

The *receive* method has no arguments. It uses methods that act on the input stream of the socket to firstly receive a header byte, and then receive the number of payload bytes specified in the header, from the external process. The payload bytes are stored in a *ByteArrayOutputStream* object (a Java object for storing bytes as output) as each segment of payload data is received. This process is repeated until all data has been received for the current implementation. The receive method returns the *ByteArrayOutputStream* object.

The *disconnect* method has no arguments, and returns no value. It acts in the same way as the *disconnect* function from the Communication Layer of Comms/CPN, except that it also closes the input and output streams from the socket before the socket itself is closed.

Methods external to the Java/CPN class must be used to convert from data (i.e. a string) into a *ByteArrayInputStream* object, and from a *ByteArrayOutputStream* object back into data. This is akin to the encoding and decoding functions passed into the send and receive functions of the Connection Management Layer in Comms/CPN.

4.2 Visualisation of Avionics Mission Systems

An Avionics Mission System (AMS) consists of a number of subcomponents connected via a serial data bus. The serial data bus (SDB) is controlled by the Mission Control Computer (MCC), and subcomponents communicate by the exchange of data across the SDB. An initial CPN model of a generic AMS [16,22] has been constructed, capturing the AMS at a high level of abstraction, including communication between subcomponents. In this section we show how COMMS/CPN can be used to visualise this communication.

A snapshot from a prototype display of the visualisation package is shown in Figure 8. The display shows the various subcomponent of the AMS connected to the SDB. Each time two components communicate via the SDB, the external visualisation package will show this communication by highlighting the two subcomponents and the SDB. The simulation will then block until the user clicks on the *Continue* button.

Fig. 8. Snapshot from the external visualisation package.

In order to provide external communication facilities, the COMMS/CPN library must be included in the CPN model. This consists of loading a number of SML files using the SML *use* command. For the AMS CPN model, the visualisation is done using code segments attached to the transitions. Opening and closing the connection to the external Visualisation Package is done by evaluating SML code in auxiliary boxes.

Of particular interest in providing visualisation of SDB communication is the *SerialDataBus* subpage of the AMS CPN model. This page is shown in Figure 9. Each subcomponent of the AMS has a unique address associated with it, and the *Transmit* transition on this subpage models the actual transmission of messages across the SDB. Two auxiliary boxes containing COMMS/CPN primitives have been added to the top left of this page. When evaluated, the first opens a connection using the *openConnection* primitive in the *ConnManagementLayer* structure, and the second closes the connection using the *closeConnection* primitive in the *ConnManagementLayer* structure which constitutes the Connection Management Layer of COMMS/CPN.

A code segment has been attached to the *Transmit* transition. This code segment calls the function shown in Figure 10. The purpose of this function is to take the addresses of the sender and receiver, map them to integers, transmit these two integers to the external visualisation package, and then await a response before continuing the simulation. The external visualisation package interprets the two integers as the corresponding sender and destination and updates the animation to reflect this data transfer. The COMMS/CPN *send* primitive is

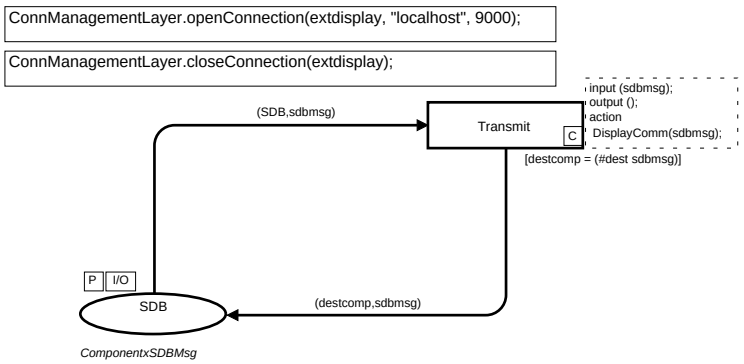


Fig. 9. The SerialDataBus page of the AMS CPN model.

used to send the two integers, and the *receive* primitive is used to receive the response from the visualisation package caused by the user clicking on the *Continue* button.

```
val extdisplay = "extdisplay"; (* --- name of connection --- *)

(* --- map subcomponent to an identifier --- *)
fun ComponentOpcode (MCC _) = "1"
  | ComponentOpcode (DISPLAYPROC (DISPLAY HUD)) = "2"
  | ComponentOpcode (DISPLAYPROC (DISPLAY MPD)) = "3"
  | ComponentOpcode (SENSOR ADC) = "4"
  | ComponentOpcode (SENSOR RADAR) = "5"
  | ComponentOpcode (SENSOR RALT) = "6"
  | ComponentOpcode (SENSOR WRW) = "7"
  | ComponentOpcode (SENSOR INS) = "8";

(* --- update the external display and wait for Continue --- *)
fun DisplayComm ({src,dest,...} : SDBMsg) =
    let
        val (src', dest') = (ComponentOpcode src, ComponentOpcode dest)
    in
        ConnManagementLayer.send(extdisplay,src'^","^dest',stringEncode);
        ConnManagementLayer.receive(extdisplay, stringDecode);
        ()
    end;
```

Fig. 10. SML functions for visualising SDB communication.

5 Conclusions and Future Work

Comms/CPN originated from a desire to provide visualisation of the simulation of AMS CPN models. Existing methods of visualisation were not satisfactory in this case, either through being too detailed or by having limited capability and being tied to the Design/CPN GUI. By developing an external visualisation package, access to greater graphical capabilities becomes

possible and visualisation is no longer tied to the DESIGN/CPN GUI. COMMS/CPN provides the necessary communication infrastructure to allow the external visualisation to take place.

The functional and non functional requirements of this library were considered and it was determined that five main functions must be provided, i.e. opening and accepting connections, sending data to and receiving data from external processes, and closing connections. It was also determined that the library must support multiple simultaneous connections and allow dynamic creation of connections. From these requirements, the library was designed. Three areas of design were considered. They were the fundamental method of communication, connection management, and data transfer. The architecture of COMMS/CPN was defined to consist of three layers, sitting between DESIGN/CPN and TCP/IP. The Communication Layer provides the interface to TCP/IP, the Messaging Layer introduces message passing scheme, and the Connection Management Layer provides the interface to DESIGN/CPN.

The current implementation of the library poses some difficulties when it comes to accepting incoming connection requests and receiving data. When listening for an incoming connection request, DESIGN/CPN blocks causing the simulation of the CPN model to block also. The same situation occurs when a receive operation is called but there is no data to receive. This blocking property is unfortunate if DESIGN/CPN is performing a simulation, because it causes the entire simulation to block (as DESIGN/CPN is purely single threaded.)

One possible area for investigation in the future is to provide non-blocking options for both the receive and accept operations. This is one area where using a messenger subprocess would have provided a relatively simple solution, as discussed Sect. 2. There is a possibility that in the future, COMMS/CPN could be used in conjunction with an external subprocess, to form a hybrid Pure TCP and Messenger solution. A system call similar to the *select* call in the C programming language would also provide a solution. Using Concurrent ML (CML) [5] instead of SML as the programming language for the DESIGN/CPN simulator and for COMMS/CPN would eliminate the blocking issues and so would also provide a solution.

Currently, the library only facilitates each connection to be connected to a single external process. To connect to more than one external process, multiple connections are used. It may be possible to extend the library functions to allow *multicasting* whereby more than one external process can receive the same data from a single connection. In this way multiple external processes will receive exactly the same data. Such multicasting would be useful when using this library for the purposes of visualisation of DESIGN/CPN simulations because it would allow exactly the same visualisation to be seen on different remote machines.

When a connection is created, the current implementation requires the user to provide the unique identifier. Future implementations of this library may give DESIGN/CPN the ability to provide this unique identifier itself, and to return this automatically generated identifier to the user for subsequent use.

Another area of future development would involve the creation of communication modules like JAVA/CPN for other programming languages, such as C/CPN, PERL/CPN and so on. Another issue to consider as part of future work is to make COMMS/CPN and JAVA/CPN libraries more fault tolerant.

Acknowledgments. The work presented in this paper was supported by the Australian Defence Science and Technology Organisation (DSTO) under contract no. 687237, and by a Divisional Small Grant from the University of South Australia. The authors also acknowledge valuable comments and feedback from Prof. Jonathan Billington.

References

1. Animation by Mimic/CPN. `http://www.daimi.au.dk/designCPN/libs/mimic/`.
2. C. Capellmann, S. Christensen, and U. Herzog. Visualising the Behaviour of Intelligent Networks. In *Services and Visualisation, Towards User-Friendly Design*, volume 1385 of *Lecture Notes in Computer Science*, pages 174–189. Springer-Verlag, 1998.
3. Computer Systems Engineering Centre. `http://www.unisa.edu.au/eie/csec`.
4. D. E. Comer. *Computer Networks and Internets*. Prentice-Hall International, Inc., 1997.
5. Concurrent ml. `http://cm.bell-labs.com/cm/cs/who/jhr/sml/cml/index.html`.
6. G. Gallasch and L. M. Kristensen. Comms/CPN library. `http://www.daimi.au.dk/designCPN/libs/commscpn/`.
7. The GAP Group, Aachen, St Andrews. *GAP – Groups, Algorithms, and Programming, Version 4.2*, 1999. (`http://www-gap.dcs.st-and.ac.uk/~gap`).
8. R. Harper. *Programming in Standard ML*. School of Computer Science, Carnegie Mellon University, `http://www.cs.cmu.edu/ rwh/introsml/`, 2000.
9. Java swing library. `http://java.sun.com/products/jfc/tsc/index.html`.
10. java.sun.com - The Source for Java(TM) Technology. `http://www.java.sun.com/`.
11. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
12. J. B. Jørgensen and L. M. Kristensen. *Design/CPN Condensed State Space Tool Manual*. Department of Computer Science, University of Aarhus, Denmark, 1996. Online: `http://www.daimi.au.dk/designCPN/`.
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. O. Kummer, D. Moldt, and F. Wienberg. A Framework for Interacting Design/CPN- and Java-Processes. In J. Kleijn and S. Donateli, editors, *Applications and Theory of Petri Nets*, volume 1639 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.
15. B. Lindstrøm. Web Based Interfaces for Simulation of Coloured Petri Net Models. In K. Jensen, editor, *Proceedings of Workshop on Practical Use of High-level Petri Nets*, pages 15–32. Department of Computer Science, University of Aarhus, Denmark, 2000. DAIMI PB-547. Available via http://www.daimi.au.dk/pn2000/proceedings/.
16. C. Douglass Locke, L. Lucas, and J. B. Goodenough. Generic Avionics Software Specification. Technical Report CMU/SEI-90-TR-8, Software Engineering Institute, Carnegie Mellon University, December 1990.
17. L. Lorentsen and L. M. Kristensen. Exploiting Stabilizers and Parallelism in State Space Generation with the Symmetry Method. In *Proceedings of International Conference on Application of Concurrency in System Design (ICACSD'2001)*, pages 211–220. IEEE Computer Society, 2001.
18. Design/CPN Message Sequence Charts library. `http://www.daimi.au.dk/designCPN/libs/mscharts/`.
19. S. Nimsgern and F. Vernet. Communication between Coloured Petri Net Simulations and External Processes. Master's thesis, Department of Computer Science, University of Aarhus, 2000.
20. Standard ML of New Jersey. `http://cm.bell-labs.com/cm/cs/what/smlnj/`.
21. Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.
22. Z. Qureshi, L. M. Kristensen, and J. Billington. Towards Modelling and Analysis of Avionics Mission Systems using Coloured Petri Nets and Design/CPN. Technical report, Defense Science and Technology Organisation, 2001. Divisional Dissussion Paper.
23. J. L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 400–419. Springer-Verlag, 1996.
24. Australien Defence Science and Technology Organisation. `http://www.dsto.defence.gov.au`.
25. SML/NJ library. `http://cm.bell-labs.com/cm/cs/what/smlnj/doc/smlnj-lib/index.html`.
26. W. Stallings. *Data and Computer Communications*. Prentice-Hall, 2000.
27. J. D. Ullman. *Elements of ML Programming*. Prentice-Hall, 1998.

Stochastic Well Formed Nets: an overview

Giuliana Franceschinis
DISTA - University of Eastern Piedmont, "A. Avogadro"
Corso Borsalino, 54, 15100 Alessandria, Italy
e-mail: giuliana@mfn.unipmn.it
tel: +39 0131 287445; fax: +39 0131 287440

Among the different proposals of high level extensions of Generalized Stochastic Petri Nets (GSPNs) [1] available in the literature, Stochastic Well-formed Nets (SWNs) [9] have gained a considerable success due to the possibility they offer of effectively coping with the state space explosion problem by exploiting the behavioural symmetries present in the model. The SWN formalism can be regarded as a *dialect* of Colored Petri Nets, however it adopts a particular syntax to specify color domains, arc functions and predicates that allow the *automatic* detection of the model symmetries and their exploitation in the model solution. In fact the main interest in SWN is due to the Symbolic Marking (SM) and Symbolic Firing notions that allow to build a reduced representation of the reachability graph called Symbolic Reachability Graph (SRG); the SRG nodes are no longer states but classes of states of the system. The SRG can be used for performance evaluation purposes [9] as well as for qualitative analysis (both basic properties like deadlock-freeness or reversibility [10], and temporal logic properties [16]). In some cases the SRG technique can be combined with orthogonal ones, like the Krocnecker algebra decomposition approaches [18].

The gain in state space reduction can be relevant if the models to be analysed are highly symmetric. However in practice it is often the case that a system behaves in a symmetric way in most situations, while in exceptional situations asymmetries arise. The SRG approach is such that an exceptional asymmetry may destroy any possibility of exploiting symmetries. For this reason an extension of the SRG technique (Extended SRG -ESRG) was proposed [17], able to exploit the symmetries whenever possible, and to deal with asymmetries only when they arise [6, 7].

The Symbolic Marking notion can be profitably used also in conjunction with discrete event simulation[14]: it allows to reduce the average event list size, the marking size and speed up the check for enabled transition instances.

Very important is the issue of *compositionality*. Even if the use of colours allows to reduce the model size, real system models (even rather abstract ones) can be handled only using a *divide and conquer* approach. A lot of literature exists in this broad field, and in particular for high level models. Concerning SWNs, some work has been done in different directions: [19], [3], [4].

Last but not least, tool support for designing and analysing SWN models is provided by the *GreatSPN* package[11] (*http://www.di.unito.it/∼greatspn*). It includes several analysis modules for SWN models, in particular a module for the construction of either the SRG or the ordinary RG and of the corresponding CTMC, and a module for symbolic or ordinary simulation. Moreover it has been recently integrated with a new tool, called *algebra*, for composition of SWN models.

SWNs are still evolving, active research is ongoing in different research groups involving both the development of new analysis algorithms for SWN models and of methodologies based on the SWN models supporting the design of systems. The practical interest of SWNs is witnessed by the several case studies present in the literature showing the applicability of the SWN formalism in different fields, e.g. control systems[2, 4], communication systems[13, 15], contact centers[12], fault tolerant systems[5, 8], etc., to study different aspects, from verification of properties, to performance and dependability analysis.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, 1995.

[2] C. Anglano, S. Donatelli, G. Franceschinis and O. Botti, "Performance prediction of a reconfigurable high voltage substation simulator: a case study using SWN" In *Proc. 7th International Workshop on Petri Nets and Performance Models*, St. Malo, France , June 1997.

[3] P. Ballarini, C. Donatelli, and G. Franceschinis. Parametric stochastic well-formed nets and compositional modelling. In *Proc. of the* 21^{th} *International Conference in Application Theory of Petri Nets (ICATPN 2000)*, volume 1825 of *Lecture Notes in Computer Science*, pages 43–62. Springer-Verlag, 2000.

[4] S. Bernardi, S. Donatelli, and A. Horáth. Compositionality in the GreatSPN tool and its use to the modelling of industrial applications. Accepted for publication on *Software Tools for Technology Transfer*.

[5] A. Bobbio, G. Franceschinis, L. Portinale, and R. Gaeta, "Dependability Assessment of an Industrial Programmable Logic Controller via Parametric Fault-Tree and High level Petri Net" In *Proc. 9th International Workshop on Petri Nets and Performance Models - PNPM01*. IEEE Computer Society, 2001.

[6] L. Capra, C. Dutheillet, G. Franceschinis and J.M. Ilie, "Towards Performance Analysis with Partially Symmetrical SWN", In *Proc. 7th International Symposium on Modeling, Analysis and Simulation*, College Park, MD, USA, October 1999.

[7] L. Capra, C. Dutheillet, G. Franceschinis and J.M. Iliè, "Exploiting Partial Symmetries for Markov Chain Aggregation ", In *Proc. of First workshop on Models for Time-Critical Systems* (MTCS 2000), Satellite workshop of CONCUR2000, Electronic Notes in Theoretical Computer Science, Volume 39, Issue 3

[8] L. Capra, R. Gaeta and O. Botti, "SWN Nets as a Framework for the Specification and the Analysis of FT Techniques Adopted in Electric Plant Automation", In Lecture Notes in Computer Science, Vol. 1630: Application

and Theory of Petri Nets 1999, 20th International Conference, ICATPN'99, Williamsburg, Virginia, USA, pages 168-187. Springer-Verlag, June 1999.

[9] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. "Stochastic well-formed coloured nets for symmetric modelling applications" *IEEE Transactions on Computers*, 42:1343–1360, 1993.

[10] G.Chiola, C.Dutheillet, G.Franceschinis, and S.Haddad, "A Symbolic Reachability Graph for Coloured Petri Nets", *Theoretical Computer Science B (Logic, semantics and theory of programming)*, Vol. 176, n. 1&2, April 1997, pp. 39-65.

[11] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. "GreatSPN 1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets" *Performance Evaluation*, 24:47–68, 1995.

[12] G. Franceschinis, C. Bertoncello, G. Bruno, G. Lungo Vaschetti, A. Pigozzi, "SWN models of a contact center: a case study" In *Proc. 9th International Workshop on Petri Nets and Performance Models - PNPM01.* IEEE Computer Society, 2001.

[13] G. Franceschinis, A. Fumagalli and A. Silinguelli "Stochastic Colored Petri Net Models for Rainbow Optical Networks" Special issue of Advances of Petri Nets on Communication Network Applications, Lecture Notes in Computer Science, Springer Verlag, LNCS 1605, April 1999.

[14] R. Gaeta, "Efficient discrete-event simulation of colored Petri nets" IEEE Transaction on Software Engineering, 22(9), September 1996.

[15] R.Gaeta and M.Ajmone Marsan, "SWN Analysis and Simulation of Large Knockout ATM Switches" volume 1420 of LNCS, Proc. of 19th International Conference on Application and Theory of Petri Nets. Springer-Verlag, June 1998.

[16] S. Haddad, J-M Ilie, and K. Ajami, A Model Checking Method for Partially Symmetric Systems, *Proc. of of FORTE XIII*, Pisa, Italy, October 2000.

[17] S. Haddad, J-M Ilie, M. Taghelit, and B. Zouari, "Symbolic marking graph and partial symmetries", In *Proc. of 16th Int. Conference on Application and Theory of Petri Nets, ICATPN '95*, pp. 238-257, Torino, Italy, June 1995.

[18] S. Haddad and P. Moreaux, "Evaluation of High Level Petri Nets by Means of Aggregation and Decomposition" In *Proc. of 6th International Workshop on Petri Nets and Performance Models*, N. Carolina, USA, pages 11-20. 1995.

[19] I. C. Rojas M., *Compositional construction and Analysis of Petri net Systems.* PhD thesis, University of Edinburgh, 1997.

Modelling and Analysis of the CES Protocol of H.245

Lin Liu and Jonathan Billington

Computer Systems Engineering Centre
University of South Australia
SCT Building, Mawson Lakes Campus, Mawson Lakes, SA 5095, Australia
liuly002@students.unisa.edu.au, jonathan.billington@unisa.edu.au

Abstract. This paper analyses part of ITU-T recommendation H.245, "Control protocol for multimedia communication". This is a pilot study of an ongoing project on modelling and analysing Internet multimedia communication standards with Coloured Petri Nets (CPNs). The Capability Exchange Signalling (CES) protocol of H.245 is modelled with CPNs. Analysis of the models shows that this protocol performs well in general, but some inadequacies also have been found. Firstly, this protocol could fail if the wrapping of the sequence numbers used by the protocol can happen, no matter whether the underlying medium of this protocol is reliable or not. Secondly, if the problem with sequence number wrap can be avoided, then, when the transport medium is unreliable, this protocol may be inefficient.

1 Introduction

Recommendation H.245 [8] is the control protocol for multimedia communication developed by the Telecommunication Standardization Sector of International Telecommunication Union (ITU-T) and has been used by a series of ITU-T multimedia system standards. These include recommendation H.310 [3] for broadband audiovisual communication, H.324 [4] for low bit-rate multimedia communication, and H.323 [6] for packet-based multimedia communication.

With the rapid growth of the Internet, techniques and standards for multimedia communications over packet-based networks are of increasing importance. At present, H.323 is the key recommendation for multimedia applications, e.g. IP (Internet Protocol) telephony and multimedia conferencing, over IP-based networks, including the Internet [9]. H.323 is a series of recommendations comprising, besides H.323 itself, H.225.0 (call setup and admission control) [5], H.245 (media channel and conference control) [8], and other related protocols [15]. The H.323 series of recommendations describes systems, logical components, messages and procedures that enable real-time multimedia calls to be established between two or more parties on a packet network. The packet network is not required to provide a guaranteed Quality of Service (QoS). The H.323 series also specifies the interoperation with multimedia systems over different networks, e.g. the Integrated Services Digital Network (ISDN) and the Public Switched Telephone Network (PSTN).

The research presented here is part of a research project [11] on modelling and analysing the H.323 standard with Coloured Petri Nets (CPNs) [10]. Previous work on modelling and analysing multimedia communication systems using Petri net techniques has concentrated on multimedia streams synchronization[1, 14] and QoS issues [16]. Little work has been done on protocols such as H.323 which are related to the general architecture, control and procedures of multimedia systems over the Internet. This proposed research is expected to expand the application domain of Petri net techniques into the *Internet* multimedia communication area and to investigate the analysis techniques which are most appropriate for the verification of Internet multimedia communication systems.

H.245 is chosen as the first standard to be modelled and analysed in this research, and some initial results have been presented in [13]. Although there is some ambiguity in the definition of H.245, compared with other recommendations of the H.323 series, H.245 is more mature and better described. Moreover, H.245 is one of the core protocols of the H.323 series. Hence we first investigate H.245 to lay the foundations for the study of other H.323 protocols.

In [13], an initial CPN model of the Capability Exchange Signalling (CES) protocol of H.245 was created and analysed, and it was assumed that the transport layer of the CES protocol is reliable. The major part of [13], however, is on the methodology for modelling and analysis of Internet multimedia

protocols. This paper aims to further model and analyse the CES protocol. A number of modifications are made to the initial model presented in [13], which is re-analysed when the underlying transport medium is reliable. Moreover, the CES protocol is also analysed under the assumption that the underlying transport medium may be unreliable. The properties of this protocol are investigated by state space analysis of the CPN models and some interesting results are obtained. The models and the analysis are discussed in detail.

This paper is organised as follows. Section 2 reviews the CES protocol. Sections 3 and 4 present the CPN models of the protocol and their analysis results when the underlying transport medium is reliable and unreliable respectively. Finally, Section 5 summarises the results and points out future directions of research.

2 The CES Protocol

Since different parties involved in a multimedia call may have different transmit and/or receive capabilities, they need to make their capabilities known to each other. Thus the multimedia streams sent by an end can be understood appropriately by its peer end(s). The CES protocol is used by a multimedia communication end to *inform* a peer end of its capabilities. It should be made clear that, although this protocol is named as capability exchange signalling protocol, it is not used by two ends to *exchange* and/or *negotiate* their capabilities. The initiator of the signalling sends out its capabilities and expects an acknowledgment from the responder side. Once the responder receives the capabilities, it is only expected to acknowledge the initiator whether it can accept them or not. Meanwhile, the responder side does not send its own capabilities to the initiator side. To be consistent with [8], however, we still use the terminology *exchange* in the following.

As defined in [8], the CES protocol consists of a set of capability exchange messages and procedures, and the CES Entities (CESEs). There is an outgoing CESE and an incoming CESE. For a particular capability exchange, an outgoing CESE is active at the initiator side and an incoming CESE is active at the responder side.

The H.245 protocols, including the CES protocol, are designed to be independent of the underlying transport medium [8], which means that they can operate over either a reliable or an unreliable transport layer. When these protocols are used in an H.323 system, however, they are required to run above a reliable transport layer, e.g. a TCP (Transmission Control Protocol) connection.

This section introduces the CES protocol and clarifies the inconsistencies found in the protocol definition. Sections 2.1 and 2.2 describe the CES protocol as it is defined in H.245 [8]. We use the same headings for these two sections (and their lower level headings) as those used in [8]. Section 2.3 lists the inconsistencies found in the CES protocol definition. All of the tables and figures in this section are taken from [8].

2.1 Communication between CESE and CESE User

2.1.1 Primitive Definition

Table 1 defines the CES primitives and their parameters. The CESE and the CESE user communicate using these primitives. The four TRANSFER primitives are used to transfer capabilities. The two REJECT primitives are used to reject a capability descriptor entry, and to terminate a current capability transfer. Primitives TRANSFER.request and TRANSFER.indication have the same parameters. PROTOID specifies the version of the recommendation in use. MUXCAP indicates the multiplexing capabilities of the outgoing end, and the multimedia receive and transmit capabilities are given by CAPTABLE and CAPDESCRIPTORS. The CAUSE parameter of a REJECT primitive indicates the reason for rejecting a CAPTABLE or CAPDESCRIPTORS parameter. The SOURCE parameter of the REJECT.indication indicates the source of the rejection, either USER or PROTOCOL.

2.1.2 CESE States

There are two states defined for an outgoing CESE: IDLE (ready to initiate a capability exchange); and AWAITING RESPONSE (waiting for a response from the peer CESE). Similarly, two states are defined for an incoming CESE: IDLE; and AWAITING RESPONSE, which indicates that the CESE is waiting for a response from its user.

Table 1. Primitives and parameters

Generic name	Type			
	request	indication	response	confirm
TRANSFER	PROTOID MUXCAP CAPTABLE CAPDESCRIPTORS	PROTOID MUXCAP CAPTABLE CAPDESCRIPTORS	- (Note 1)	-
REJECT	CAUSE	SOURCE CAUSE	not defined (Note 2)	not defined
NOTE 1 - "-" means no parameters. NOTE 2 - "not defined" means that this primitive is not defined.				

2.2 Peer-to-peer CESE Communication

2.2.1 Messages

There are four kinds of CES messages that may be transmitted for a capability exchange (Table 2). The TerminalCapabilitySet message is sent by the outgoing CESE to indicate the capabilities of the outgoing end. Messages TerminalCapabilitySetAck and TerminalCapabilitySetReject are the positive response (for accept) and negative response (for reject) from the incoming CESE respectively. TerminalCapabilitySetRelease is sent by the outgoing CESE to the incoming CESE to request termination of the current exchange when possible.

Table 2. CESE message names and fields

Function	Message	Direction	Field
transfer	TerminalCapabilitySet	O -> I	sequenceNumber protocolIdentifier multiplexCapability capabilityTable capabilityDescriptor
	TerminalCapabilitySetAck	O <- I	sequenceNumber
reject	TerminalCapabilitySetReject	O <- I	sequenceNumber cause
reset	TerminalCapabilitySetRelease	O -> I	-
NOTE - Direction: O - Outgoing, I - Incoming.			

The first three messages contain a transaction identifier called sequenceNumber. This message field ensures that the outgoing CESE can correlate a response to the corresponding TerminalCapabilitySet message, and inform its user of the expected response. The other fields of TerminalCapabilitySet are obtained from the parameters of the TRANSFER.request. The “cause” field of TerminalCapabilitySetReject has the same value as the parameter CAUSE of the primitive REJECT.request issued by the incoming CESE user.

2.2.2 CESE State Variables

A state variable, *out_SQ*, is defined for the outgoing CESE. It is used to indicate the most recent TerminalCapabilitySet message. Once a TRANSFER.request primitive is issued by the outgoing user, this variable is incremented by one (modulo 256) and is written into the sequenceNumber field of TerminalCapabilitySet before its transmission. Correspondingly, there is a state variable, *in_SQ*, at the incoming side to store the value of the sequenceNumber field of the most recently received TerminalCapabilitySet message. This value will be written into the sequenceNumber field of the TerminalCapabilitySetAck message or the TerminalCapabilitySetReject message before it is sent to the outgoing CESE.

2.2.3 CESE Procedures

In H.245, the CESE procedures are defined in SDL (Specification and Description Language) [7] as shown in Fig. 1. For each CESE, there are two SDL diagrams, one for each CESE major state (IDLE and AWAITING RESPONSE).

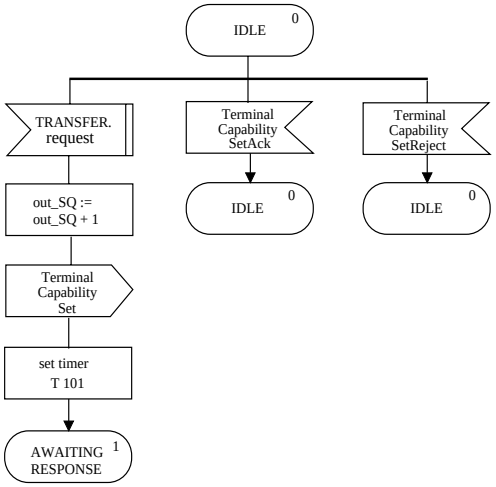
Referring to Fig. 1, a capability exchange is initiated when a TRANSFER.request is issued by the outgoing CESE user (Fig. 1.**a**). The CESE increments its state variable, *out_SQ*, sends a TerminalCapabilitySet message to the incoming CESE, starts a timer, and enters the AWAITING RESPONSE state. Upon receiving the TerminalCapabilitySet message, the incoming CESE copies the sequenceNumber field into variable *in_SQ* (Fig. 1.**c**), informs the user (TRANSFER.indication) and changes state from IDLE to AWAITING RESPONSE. The user accepts the capabilities by issuing a TRANSFER.response (Fig. 1.**d**), or rejects them (REJECT.request). Accordingly, a TerminalCapabilitySetAck message or a TerminalCapabilitySetReject message is sent to the peer CESE and the state of the incoming CESE becomes IDLE. At the outgoing side (Fig. 1.**b**), if the response message is received before the timer expires, the outgoing CESE notifies the user by submitting a TRANSFER.confirm or a REJECT.indication (depending on the message it receives), and returns to IDLE after resetting the timer. If the timer expires before any response is received, a TerminalCapabilitySetRelease message is sent to the peer CESE, a REJECT.indication primitive (with SOURCE set to PROTOCOL) is issued to the user, and the CESE's state returns to IDLE. When the TerminalCapabilitySetRelease message arrives at the incoming side, if the incoming CESE is in the AWAITING RESPONSE state (Fig. 1.**d**), it will submit a REJECT.indication (with its SOURCE parameter set to PROTOCOL) to the user to terminate the current exchange. If it is IDLE, i.e. the release message arrives after the incoming user has issued the response (Fig. 1.**c**), this message will be ignored.

We noted earlier that, all H.245 protocols are designed to operate over a transport layer that can be reliable or unreliable. In the CES protocol, the timer and sequence numbers are used to deal with the unreliability of the transport layer. It is possible that a response message is received by the outgoing CESE after the timer expires and the outgoing CESE is IDLE (Fig. 1.**a**), or a new TerminalCapabilitySet message has been sent out and the CESE is AWAITING RESPONSE (Fig. 1.**b**), waiting for the response to the most recently sent TerminalCapabilitySet message. Then this late response will be ignored (when the CESE is IDLE) or discarded (when the CESE is AWAITING RESPONSE). Also, it is possible for the incoming CESE to receive a new TerminalCapabilitySet message when it is in the AWAITING RESPONSE state (Fig. 1.**d**), waiting for the response from the user to the last TerminalCapabilitySet message. In this case, the incoming CESE will issue a REJECT.indication primitive (according to the text of H.245, its SOURCE parameter has the default value USER) to the user to terminate the current exchange. After the REJECT.indication, a TRANSFER.indication primitive is issued to the user to indicate the arrival of the new TerminalCapabilitySet message.

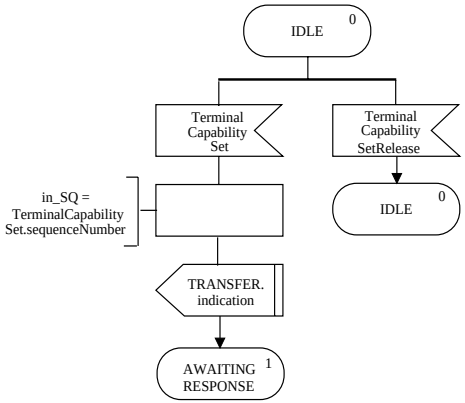
2.3 Inconsistencies in the CES Protocol Definition

The following inconsistencies were found in the CES protocol definition [8].

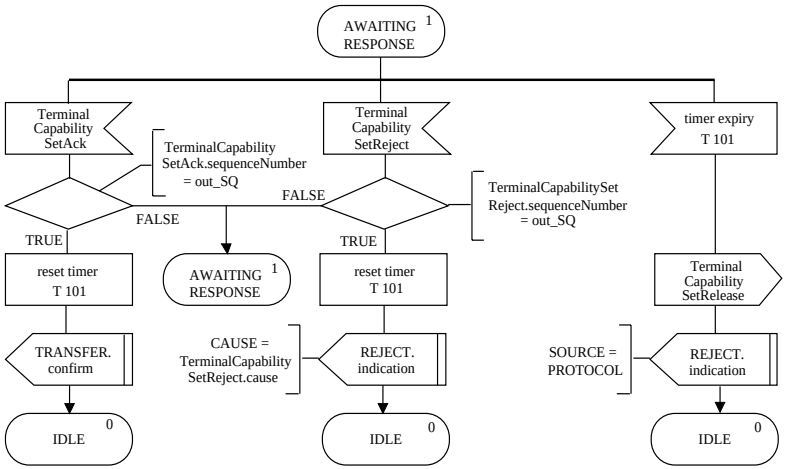
- In Fig. 1.**a**, ":=" is used in expression "out_SQ := out_SQ + 1". From the narrative description we know that the function of this expression should be to increment *out_SQ* by 1. So ":=" is used as an assignment symbol. In contrast, the two "in_SQ = TerminalCapabilitySet.sequenceNumber" expressions of the incoming CESE SDL (Fig. 1.**c** and Fig. 1.**d**), use "=" to assign the value of the sequenceNumber field of the TerminalCapabilitySet message to the state variable, *in_SQ*. In the following sections, we interpret these two "=" as assignment symbols.
- In the narrative description of the CES protocol, it is stated that when a TRANSFER.request primitive is issued by the outgoing user, the state variable *out_SQ* is incremented by 1, modulo 256. However in Fig. 1.**a**, the increment operation is shown as"out_SQ := out_SQ + 1", without indicating that modulo arithmetic must be used.
- In the SDL diagrams, the procedure for writing the values of the state variables, *out_SQ* and *in_SQ*, into the corresponding messages is not stated explicitly. We believe it should be.



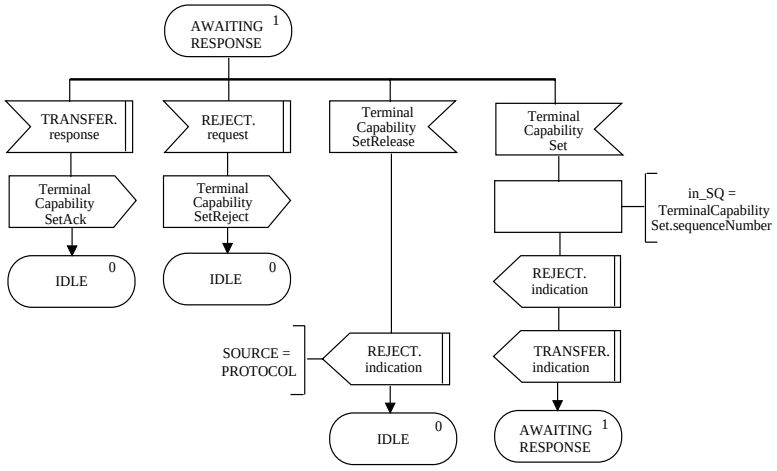
a. Outgoing CESE SDL (1)



c. Incoming CESE SDL (1)



b. Outgoing CESE SDL (2)



d. Incoming CESE SDL (2)

Fig. 1. The outgoing CESE and incoming CESE SDLs

3 Modelling and Analysing the CES Protocol

It is required by recommendation H.323 that the H.245 control channel of an H.323 system is a reliable channel, e.g. a TCP connection is used for an H.323 system on the Internet. Since the application of H.245 to H.323 systems is our interest, in this section we model and investigate this protocol operating over a reliable transport layer.

3.1 Modelling Assumptions

3.1.1 Transport Medium

According to H.323 [6], a reliable transport channel must preserve sequence, be error-free, and provide flow controlled transmission of messages. Our CPN model reflects this requirement.

3.1.2 Interpretation of the CES Protocol Definition

The CPN model is based on the SDL diagrams shown in Fig. 1, modified according to our interpretations explained in Section 2.3.

3.2 Conventions Used in the CPN Model

Figure 2 (ignoring the dashed part attached to transition `TRANSFERreq` and the guard of this transition) is the CPN model created for the CES protocol when the transport channel is reliable. We call it the "original model". In this model, we use the following conventions:

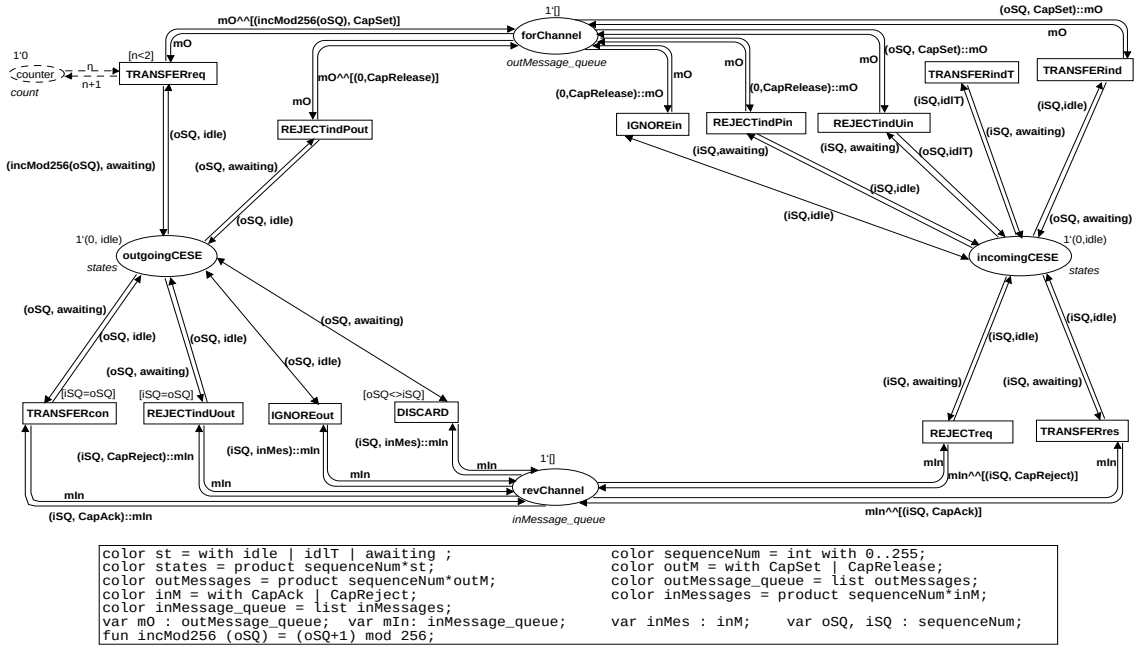
- Service primitives are modelled as transitions. The rules for naming the transitions from the corresponding primitive names are: the general primitive names are preserved, e.g. TRANSFER still appears as TRANSFER in the corresponding transition names; the "." in the name is omitted; the type names of primitives are abbreviated to their first 3 letters (e.g. "request" becomes "req"); when modelling REJECT.indication, two separate transitions are used to distinguish the SOURCE of rejection, so following the type name, a capital letter "U" (USER), or "P" (PROTOCOL), is attached; to distinguish primitives with the same name but at different ends (e.g. REJECT.indication), the suffix "out" is added to a transition name to show that it is at the outgoing end, and suffix "in" for the incoming end.
- All the message names are abbreviated. Referring to Table 2, the prefix "Terminal" is omitted from all the message names, "Capability" is abbreviated to "Cap" and "Set" is omitted from the names of all the messages except the TerminalCapabilitySet message.
- The states' names of the outgoing and incoming CESEs are modified. IDLE is changed to lower case representation "idle" and AWAITING RESPONSE is simplified as "awaiting" in lower case.

3.3 The CPN Model of the CES Protocol

This section introduces the original CPN model of the CES protocol (Fig. 2) in detail. In the following descriptions, all the names of the transitions, places, variables and functions used in this model are highlighted by using the `courier` font. This model is composed of four parts described below.

3.3.1 Declarations

At the bottom of Fig. 2 is the declaration node of this CPN. We introduce these declarations in the order they are declared. Colour `st` defines all the possible states of the outgoing and incoming CESEs, and colour `sequenceNum` specifies the range of the sequenceNumber field of the CES messages, as well as the range of the value of state variables *out_SQ* and *in_SQ*. The Cartesian product of `sequenceNum` and `st` forms the colour `states`. This colour is used to type the state places `outgoingCESE` and `incomingCESE`. Colour `outM` defines the messages that can be sent to the incoming CESE. Colour `outMessages` combines `sequenceNum` with `outM`. Colour `outMessage_queue` forms a list of outgoing




```
color st = with idle | idlT | awaiting ;                color sequenceNum = int with 0..255;
color states = product sequenceNum*st;                  color outM = with CapSet | CapRelease;
color outMessages = product sequenceNum*outM;           color outMessage_queue = list outMessages;
color inM = with CapAck | CapReject;                    color inMessages = product sequenceNum*inM;
color inMessage_queue = list inMessages;
var mO : outMessage_queue;  var mIn: inMessage_queue;   var inMes : inM;    var oSQ, iSQ : sequenceNum;
fun incMod256 (oSQ) = (oSQ+1) mod 256;
```

Fig. 2. CPN model of the CES protocol with reliable transport layer

messages and is used to type the place `forChannel`. Similarly, colour `inMessage_queue` types place `revChannel`.

Variables `mO` and `mIn` range over `outMessage_queue` and `inMessage_queue` respectively. Variable `inMes` has colour `inM`, and is used to discard messages from place `revChannel` (transitions `IGNOREout` and `DISCARD`). Variables `oSQ` and `iSQ` are integers within the range of the sequenceNumber fields of the CES messages and the range of the CESEs' state variables. Lastly function `incMod256(oSQ)` increments the value of variable `oSQ` by 1, modulo 256.

3.3.2 The Outgoing CESE

The left part of this CPN describes the outgoing CES protocol entity. Place `outgoingCESE` models the states of the CESE which control the sequence of operations of the CESE, including the state variable, *out_SQ*. We define the colour set of this place as `states`. Every time a TRANSFER.request primitive is issued by the user (i.e. `TRANSFERreq` occurs), function `incMod256` increments `oSQ` (which represents the previous value of *out_SQ*) by 1, modulo 256, and writes the result into the sequenceNumber field of message `CapSet`. Then this message is concatenated to the tail of the queue in place `forChannel`. The value of *out_SQ* stored in place `outgoingCESE` is incremented by 1, modulo 256, and the state of `outgoingCESE` is updated to `awaiting` when transition `TRANSFERreq` occurs.

There is no sequenceNumber field defined for message TerminalCapabilitySetRelease. In order not to make the type of place `forChannel` complicated, we still use a pair (i.e. `(0, CapRelease)`) to represent this message, but set the first item of the pair to 0 in all cases.

Transitions `TRANSFERreq`, `TRANSFERcon`, `REJECTindUout` and `REJECTindPout` correspond to the service primitives at the outgoing end. On their occurrence, a corresponding CES message will be sent or received by the outgoing CESE. Also, transition `REJECTindPout` models the behaviour of the timer implicitly. The guards attached to transitions, `TRANSFERcon` and `REJECTindUout`, ensure that the CESE accepts a response with the right sequence number. Transitions `IGNOREout` and `DISCARD` are used to discard a message received when the outgoing CESE is not in the right state (`IGNOREout`) or when the message received has a different sequence number from that being expected by the outgoing CESE (`DISCARD`). Each transition in this part corresponds to a SDL state transition (i.e. a SDL diagram branch leading from one state to another state) in Fig. 1.

3.3.3 The Underlying Transport Medium

The two places in the middle, `forChannel` and `revChannel`, model the underlying transport medium, over which the CES messages are transmitted. We have assumed that the channel is reliable, so the two directions of transmission are modelled as two independent FIFO (First In First Out) queues without message loss or duplication. Place `forChannel` models the FIFO queue for messages flowing from the outgoing CESE to the incoming CESE. Similarly, place `revChannel` represents the FIFO queue from the incoming CESE to the outgoing CESE.

3.3.4 The Incoming CESE

The right hand side of this CPN models the behaviour of the incoming CESE. Place `incomingCESE` corresponds to the state of the incoming CESE, including the state variable *in_SQ*.

Transitions `REJECTreq` and `TRANSFERres` correspond to primitives REJECT.request and TRANSFER.response respectively. Transition `REJECTindPin` models a REJECT.indication with the source of rejection being PROTOCOL (corresponding to a timeout at the outgoing end). Transition `IGNOREin` is used to ignore a `CapRelease` message received when the incoming CESE is IDLE.

According to the right most SDL state transition of Fig. 1**.d**, when the incoming CESE is in the AWAITING RESPONSE state and another TerminalCapabilitySet message is received, it will terminate the previous exchange by issuing a REJECT.indication primitive to its user. Immediately following this, a TRANSFER.indication primitive is issued by the incoming CESE, and it returns to the AWAITING RESPONSE state.

Since the occurrences of service primitives are considered as separate atomic events, we model this SDL state transition as two CPN transitions, `REJECTindUin` and `TRANSFERindT`. They correspond to the two primitives, REJECT.indication (with parameter USER) and TRANSFER.indication, in this SDL state transition (Fig. 1**.d**). We introduce a temporary state `idlT` for the incoming CESE. Once transition `REJECTindUin` occurs, the CESE will move to `idlT`. Immediately following this, only transition `TRANSFERindT` can occur at the incoming side, which models the SDL state transition faithfully. Finally, when it is IDLE, the incoming CESE can issue a TRANSFER.indication primitive when it receives a TerminalCapabilitySet message, which is modelled as transition `TRANSFERind`.

3.4 State Space Analysis

It is not hard to see that the original model in Fig. 2 has an infinite state space. This is caused by the unlimited occurrence of the transition sequence `TRANSFERreq` followed by `REJECTindPout` (unlimited timeouts). Therefore an infinite number of messages can be put into place `forChannel` and subsequently into `revChannel`, thus producing an infinite state space.

To apply state space analysis to this model, some measures to limit the size of the state space are needed. We do this in two ways. The first is to limit the queue length of places `forChannel` and `revChannel`, and the second is to limit the number of times transitions `TRANSFERreq` and/or `REJECTindPout` occur. Both methods have inadequacies. Limiting queue length will prevent us from obtaining some possible sequences. For example if the queue length of place `forChannel` is set to 1, then transition `REJECTindPout` can not occur immediately after transition `TRANSFERreq`, because the queue is occupied by message `CapSet`. In fact, transition `REJECTindPout` should be allowed to occur as long as the outgoing CESE is in the AWAITING RESPONSE state. In practice, however, it is seldom that a timeout can occur before the message has been received by the peer end. Even so, to obtain more general analysis results, in the following investigation in Section 3.4.1, we assume that the queue length of places `forChannel` and `revChannel` is 3. This will include the rare situation where a timeout can occur before the `CapSet` message is received by the incoming CESE, which may happen when the network is congested.

Limiting the occurrence times of transitions `TRANSFERreq` and/or `REJECTindPout` will not affect the sequence, but another problem arises. If we limit the times of occurrence to n and observe the properties of the protocol, we would need a proof by induction to generalise the results for arbitrary n. Nevertheless these methods are very useful when we are interested in investigating some particular behaviour of the protocol, and generally, they are complementary. Additionally, to reduce the size of the state space and to simplify its analysis, the sequence number size also needs to be reduced.

The initial marking of the CPN model is shown in Fig. 2. The outgoing and incoming CESEs are IDLE and the initial values of the state variables are both "0" (M_0(`outgoingCESE`) = M_0(`incomingCESE`) = 1`(0,idle)). Both `forChannel` and `revChannel` are marked by the empty list indicating that both channels are empty.

3.4.1 Analysis of Model A

Firstly, we assume that the allowable range of the sequence numbers is {0,1} (modulo 2 arithmetic instead of 256). This is the smallest range we can use without losing the generality of different sequence numbers. Then we limit the queue length of places `forChannel` and `revChannel` to 3. By doing so, we allow the timeout to occur before the incoming CESE receives the `CapSet` message. Also, because the range of the sequence numbers is {0,1}, if it is possible for 3 messages to exist in place `forChannel` or `revChannel` at the same time, we model the case of the wrapping of sequence numbers, which can be observed by examining the markings of these two places (For example, in Fig. 3, from the marking of node 148, we observe that a `CapAck` message and a `CapReject` message with the same sequence number "1" coexist in place `revChannel` (i.e. revCh). Explicit observations like this will facilitate the analysis later).

This simplified model is called "Model A" in the following. We use the OG (Occurrence Graph) tool of Design/CPN [2] to obtain and analyse the state space of this model.

Table 3 shows part of the state space report of the limited model. The first part of this table shows the statistics for the OG and the SCC (Strongly Connected Component) graph. The OG has 4828 nodes and 13480 arcs, and took 54 seconds to calculate using a PC with an Intel Celeron 500MHz CPU and 128MB of RAM. There is only one node for the SCC graph, which implies that in the OG, every node is reachable from every other node. This is consistent with the Home Properties which show that every node in the OG is a home marking.

Table 3. Part of the state space report for Model A

	Occurrence Graph		SCC Graph	
Statistics	Nodes:	4828	Nodes:	1
	Arcs:	13480	Arcs:	0
	Secs:	54	Secs:	5
	Status:	Full		
Home Properties	Home Markings:	All		
Liveness Properties	Dead Markings:	None		
	Dead Transition Instances:	None		
	Live Transition Instances:	All		

The Liveness Properties tell us that there is no dead marking, no dead transition instance and all transitions are live transition instances. These properties generally are expected for a protocol.

However, for this model, we expect transitions `REJECTindUin` and `TRANSFERindT` to be *dead*. From Fig. 2, we see that to enable transition `REJECTindUin`, we need a `CapSet` message in the `forChannel` while the `incomingCESE` is `awaiting`. This means that two successive `CapSet` messages have to be sent by the outgoing CESE, where the first is used to bring the `incomingCESE` to be `awaiting` (via the occurrence of `TRANSFERind`), and after this, the latter is used to enable transition `REJECTindUin`.

However, if the protocol performs as expected, and the underlying transport medium is reliable, it is not possible for two `CapSet` messages to be sent *successively* without receiving a response from the incoming CESE. We know that the outgoing CESE can send a `CapSet` message either when it receives an expected response, or when the timer expires before the response is received (i.e. `REJECTindPout` occurs). When the `outgoingCESE` receives an expected response, the `incomingCESE` must be `idle`. So even if the incoming CESE receives a `CapSet` message at this time, `REJECTindUin` can not be enabled. When there is a timeout at the outgoing side, the `incomingCESE` may be `awaiting`, but a `CapRelease` message has to be sent to the `forChannel` first, before another `CapSet` can be sent. Then if the `incomingCESE` is `awaiting`, this `CapRelease` message will enable `REJECTindPin`, and its occurrence will return the `incomingCESE` to `idle`. If the `incomingCESE` is `idle`, then the occurrence

of IGNOREin will discard this CapRelease message and the incomingCESE will stay in idle. Hence, transition REJECTindUin can not be enabled in either case.

Moreover, transition TRANSFERindT can be enabled only after REJECTindUin occurs so that the incomingCESE can be in the temporary state idlT. Thus we would have expected that transitions REJECTindUin and TRANSFERindT would not have been enabled at all in this model, and should have been reported as *dead transition instances*. From the above analysis it is obvious that this is not related to the limitation of the queue length of places forChannel and revChannel. We checked the model with the protocol specification given in H.245, and believe that it reflects the specification. So we think this unexpected result indicates that some inadequacies or errors exist in this protocol definition.

3.4.2 Further Analysis of Model A

The OG was analysed in more detail, and found the unexpected states that cause this problem, and some other unexpected occurrence sequences. Figure 3 shows a part of the OG that includes some examples of the unexpected sequences showing the wrong operation of the protocol. In this diagram, all the transition and place names are abbreviated. Place outgoingCESE and incomingCESE are named as "outCESE" and "inCESE" respectively, and "forCh" and "revCh" are the abbreviations of place names forChannel and revChannel. The abbreviated transition names will be annotated immediately following the transition names when they first appear in the following text.

Starting from node 1, i.e. the initial state of this model, transition TRANSFERreq (Treq) occurs, state variable *out_SQ* is increased from 0 to 1, and a CapSet message with sequence number 1 is sent to the forChannel. Then TRANSFERind (Tind) occurs followed by TRANSFERres (Tres) at the incoming side, so a CapAck message with sequence number 1 is sent back via the revChannel (node 5). Before the outgoing CESE receives this message, the timer expires, transition REJECTindPout (RindPo) occurs and a CapRelease message is sent to the forChannel. Because the incomingCESE is idle, this release message is ignored (IGNOREin, i.e. IGi in the OG).

Now a new TRANSFERreq is issued and the procedure is just the same as before, but the sequence numbers have been changed. The response message is created by the incoming CESE, and the timer expires again before this message is received by the outgoing side. At this stage there are two response messages with different sequence numbers in the revChannel, as shown in node 68. A new TRANSFERreq occurs and the outgoing CESE is now waiting for the response message with sequence number 1 again, and the sequence number for the newly sent CapSet message also is 1 (node 87).

Then something interesting happens. If we look at the right branch of this diagram first, it describes one of the possible situations where the outgoing CESE takes the first item in the revChannel and transition TRANSFERcon (Tcon) occurs. This primitive confirms the wrong capabilities. The CapAck message with sequence number 1 is the response to the CapSet message (with sequence number 1) sent in the first round of the sequence numbers (i.e. the CapSet message shown in node 2). The outgoing CESE can not identify the wrong response and treats this old response as the current one. Thus the user may receive an incorrect acknowledgment. Not only that, because TRANSFERreq can occur again, two successive CapSet messages can exist in the forChannel (node 142). Now TRANSFERind occurs and awaiting is in place incomingCESE (node 185), so that it is possible for transitions REJECTindUin (RindUi) and TRANSFERindT (TindT) to occur. Then the outgoing CESE again delivers an old CapAck when TRANSFERcon occurs.

Similarly, for the left branch starting from node 87, the outgoing CESE also takes an old CapAck, and in this case, it is even worse. As shown in node 148, the corresponding (correct) response in the revChannel is a negative response (i.e. the last element of the queue, which was concatenated to the queue when REJECTreq (Rreq) occurred), but when TRANSFERcon occurs, the outgoing CESE takes the first element in the queue which is a positive response. This means that although the incoming end did not accept the capabilities of the outgoing end, the outgoing end believes that the incoming end has accepted its capabilities. This could cause a failure of the associated multimedia session, and is thus a serious error.

From the analysis above, we can conclude that this protocol could fail if wrapping of sequence numbers (modulo 2) can happen, while 2 acknowledgments are still outstanding (not yet received). This could occur when the timer is set too short and/or the range of the sequence numbers is not big

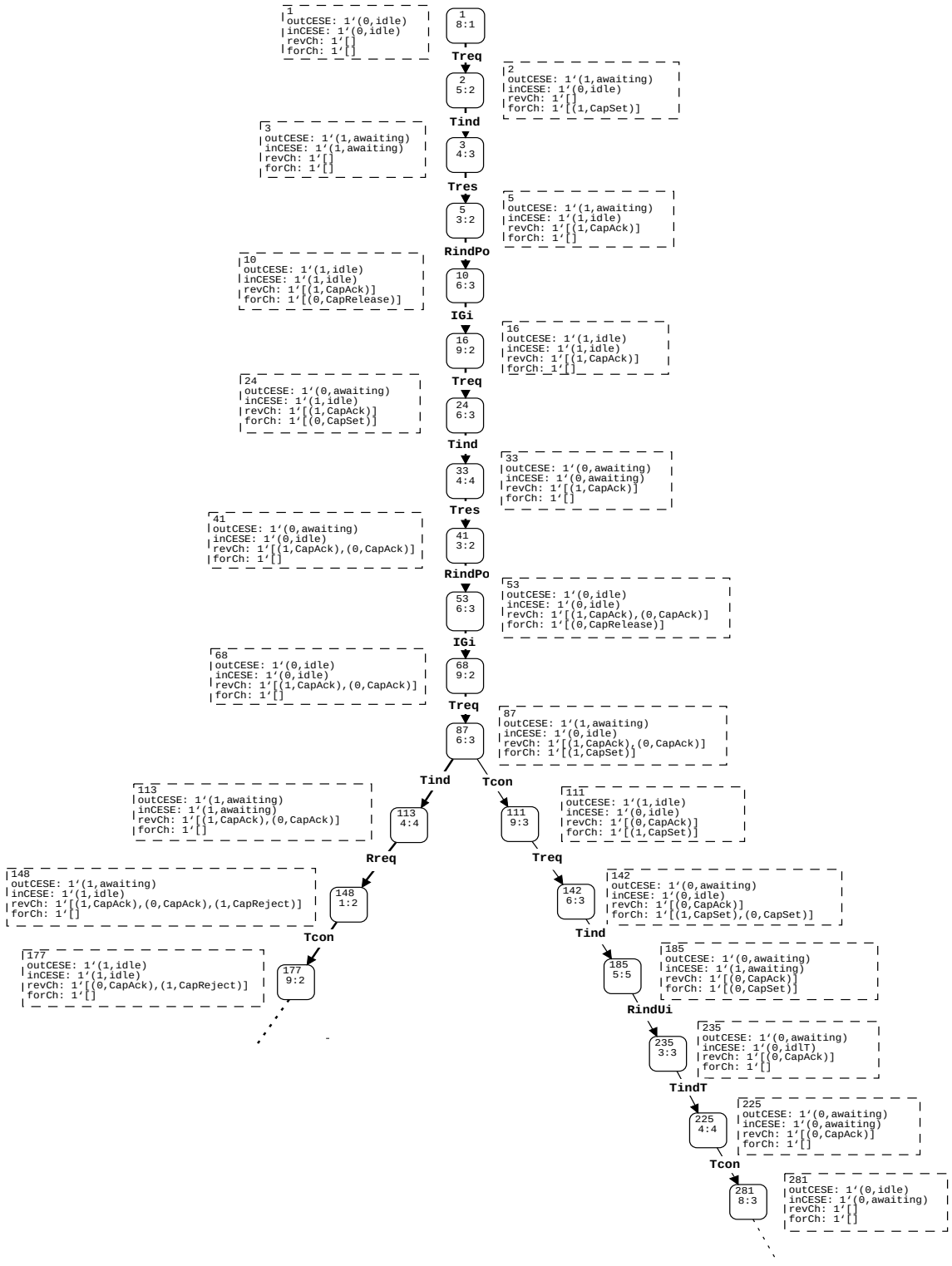


Fig. 3. Partial OG of Model A showing faulty behaviour

enough. In the example shown in Fig. 3, the timer is set too short for the given range of sequence numbers, and it expires consecutively 2 times after the first TerminalCapabilitySet message has been sent. Then, as shown in node 87, the third TerminalCapabilitySet messages is sent, and the wrapping of sequence numbers occurs when there are two outstanding responses in the `revChannel`. Thus it makes possible for the outgoing CESE to mistake the old responses for the expected ones.

3.4.3 Analysis of Model B

To further analyse the behaviour of the protocol, we limit the maximum number of times `TRANSFERreq` can occur to 2. Then only two sequence numbers, 1 and 0, can occur, and only occur once for each, so wrapping of sequence numbers is not possible. We add a counter to the original model, which is shown as the dashed part connected to transition `TRANSFERreq` (Fig. 2). The colour set of the place `counter` is `count`, which is defined as the set {`0,1,2`}. The `n` appearing on the dashed arcs is a variable of type `count`. The declaration of colour `count` and variable `n` is not included in the declaration node of the original model. Also, a guard `[n<2]` is added to transition `TRANSFERreq` to check the value of the `counter`. Furthermore, we do not limit the queue size of places, `forChannel` and `revChannel`, to obtain all the sequences and states in this situation. We name this model, "Model B".

Table 4 is part of the state space report for Model B, Fig. 4 is the OG obtained and Table 5 is the state table showing all the node information of the OG. All the names of the transitions and places in the OG and state table are abbreviated as in Fig. 3, with four more abbreviations: "RindUo", "RindPi" "IGo" and "DIS". "RindUo" represents `REJECTindUout`, "RindPi" is the abbreviation of `REJECTindPin`, "IGo" is for `IGNOREout` and "DIS" stands for `DISCARD`. The OG for Model B is much smaller, comprising 57 nodes and 120 arcs. The SCC graph has the same size as that of the OC, which indicates that there is no infinite occurrence sequence of transitions, i.e. no loops in the OG.

Table 4. Part of the state space report for Model B

Statistics	Occurrence Graph		SCC Graph	
	Nodes:	57	Nodes:	57
	Arcs:	120	Arcs:	120
	Secs:	0	Secs:	0
	Status:	Full		
Home Properties	Home Markings:	[36]		
Liveness Properties	Dead Markings:	[36]		
	Dead Transition Instances:	REJECTindUin & TRANSFERindT		
	Live Transition Instances:	None		

From the Home Properties we see that there is only one home marking, i.e. node 36 in the OG, which is also the only dead marking of this model. This implies that all sequences terminate at this state and we know from Table 5 that this state is the expected terminal state. In this state both the outgoing and incoming CESEs are IDLE, no message is left in either channel and the `counter` has reached its limit. Moreover, Table 4 shows that transitions `REJECTindUin` and `TRANSFERindT` no longer occur in this case. This is consistent with our expectation before and also implies that if the transport layer is reliable and there is no possibility of sequence number wrapping, these two transitions cannot occur.

3.4.4 Further Analysis of Model B

By investigating the full OG (Fig. 4) of Model B, we can see that this protocol works as desired when there is only one round of sequence numbers, since there is no confusion of sequence numbers during the running of the protocol. Firstly, in Fig. 4, the sequence highlighted (bold) shows the two successful capability exchanges. This sequence shows that, under the limitation of Model B, it is possible for this protocol to carry out consecutively successful capability exchanges.

Secondly, when a timeout (i.e. transition `REJECTindPout`) occurs, that is to say, the outgoing side wants to terminate the current exchange before it recieves the response, this protocol can operate correctly as well.

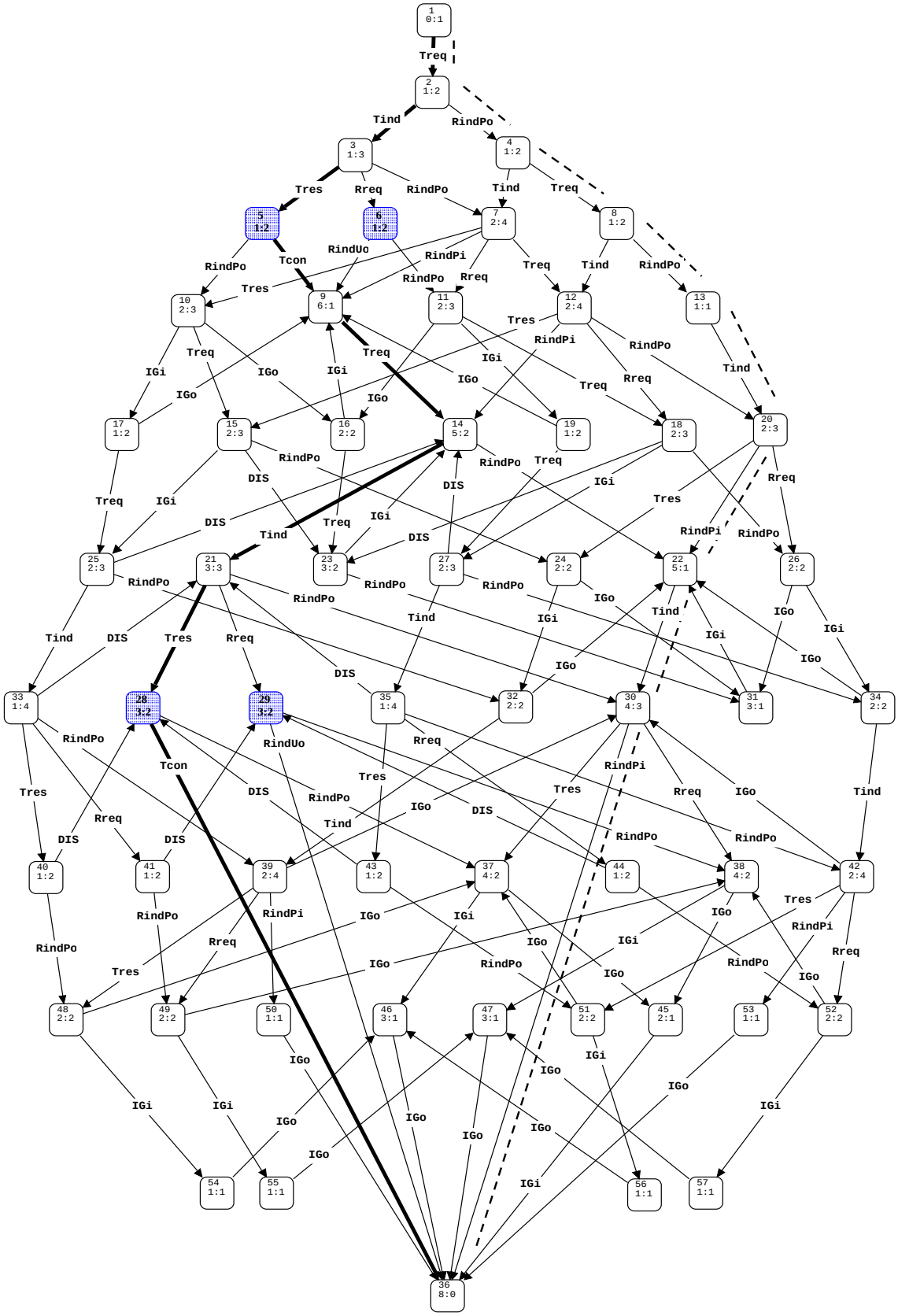


Fig. 4. The OG of Model B

Table 5. State table of Model B

NO.	counter	outCESE	inCESE	forCh	revCh
1	1`0	1`(0,idle)	1`(0,idle)	1`[]	1`[]
2	1`1	1`(1,awaiting)	1`(0,idle)	1`[(1,CapSet)]	1`[]
3	1`1	1`(1,awaiting)	1`(1,awaiting)	1`[]	1`[]
4	1`1	1`(1,idle)	1`(0,idle)	1`[(1,CapSet),(0,CapRelease)]	1`[]
5	1`1	1`(1,awaiting)	1`(1,idle)	1`[]	1`[(1,CapAck)]
6	1`1	1`(1,awaiting)	1`(1,idle)	1`[]	1`[(1,CapReject)]
7	1`1	1`(1,idle)	1`(1,awaiting)	1`[(0,CapRelease)]	1`[]
8	1`2	1`(0,awaiting)	1`(0,idle)	1`[(1,CapSet),(0,CapRelease),(0,CapSet)]	1`[]
9	1`1	1`(1,idle)	1`(1,idle)	1`[]	1`[]
10	1`1	1`(1,idle)	1`(1,idle)	1`[(0,CapRelease)]	1`[(1,CapAck)]
11	1`1	1`(1,idle)	1`(1,idle)	1`[(0,CapRelease)]	1`[(1,CapReject)]
12	1`2	1`(0,awaiting)	1`(1,awaiting)	1`[(0,CapRelease),(0,CapSet)]	1`[]
13	1`2	1`(0,idle)	1`(0,idle)	1`[(1,CapSet),(0,CapRelease),(0,CapSet), (0, CapRelease)]	1`[]
14	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapSet)]	1`[]
15	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet)]	1`[(1,CapAck)]
16	1`1	1`(1,idle)	1`(1,idle)	1`[(0,CapRelease)]	1`[]
17	1`1	1`(1,idle)	1`(1,idle)	1`[]	1`[(1,CapAck)]
18	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet)]	1`[(1,CapReject)]
19	1`1	1`(1,idle)	1`(1,idle)	1`[]	1`[(1,CapReject)]
20	1`2	1`(0,idle)	1`(1,awaiting)	1`[(0,CapRelease),(0,CapSet),(0,CapRelease)]	1`[]
21	1`2	1`(0,awaiting)	1`(0,awaiting)	1`[]	1`[]
22	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapSet),(0,CapRelease)]	1`[]
23	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet)]	1`[]
24	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet),(0,CapRelease)]	1`[(1,CapAck)]
25	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapSet)]	1`[(1,CapAck)]
26	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet),(0,CapRelease)]	1`[(1,CapReject)]
27	1`2	1`(0,awaiting)	1`(1,idle)	1`[(0,CapSet)]	1`[(1,CapReject)]
28	1`2	1`(0,awaiting)	1`(0,idle)	1`[]	1`[(0,CapAck)]
29	1`2	1`(0,awaiting)	1`(1,idle)	1`[]	1`[(0,CapReject)]
30	1`2	1`(0,idle)	1`(0,awaiting)	1`[(0,CapRelease)]	1`[]
31	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapRelease),(0,CapSet),(0,CapRelease), (0,CapRelease)]	1`[]
32	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapSet),(0,CapRelease)]	1`[(1,CapAck)]
33	1`2	1`(0,awaiting)	1`(0,awaiting)	1`[]	1`[(1,CapAck)]
34	1`2	1`(0,idle)	1`(1,idle)	1`[(0,CapSet),(0,CapRelease)]	1`[(1,CapReject)]
35	1`2	1`(0,awaiting)	1`(0,awaiting)	1`[]	1`[(1,CapReject)]
36	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[]
37	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(0,CapAck)]
38	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(0,CapReject)]
39	1`2	1`(0,idle)	1`(0,awaiting)	1`[(0,CapRelease)]	1`[(0,CapAck)]
40	1`2	1`(0,awaiting)	1`(0,idle)	1`[]	1`[(1,CapAck),(0,CapAck)]
41	1`2	1`(0,awaiting)	1`(0,idle)	1`[]	1`[(1,CapAck),(0,CapReject)]
42	1`2	1`(0,idle)	1`(0,awaiting)	1`[(0,CapRelease)]	1`[(1,CapReject)]
43	1`2	1`(0,awaiting)	1`(0,idle)	1`[]	1`[(1,CapReject),(0,CapAck)]
44	1`2	1`(0,awaiting)	1`(0,idle)	1`[]	1`[(1,CapReject),(0,CapReject)]
45	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[]
46	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(0,CapAck)]
47	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(0,CapReject)]
48	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(1,CapAck),(0,CapAck)]
49	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(1,CapAck),(0,CapReject)]
50	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapAck)]
51	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(1,CapReject),(0,CapAck)]
52	1`2	1`(0,idle)	1`(0,idle)	1`[(0,CapRelease)]	1`[(1,CapReject),(0,CapReject)]
53	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapReject)]
54	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapAck),(0,CapAck)]
55	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapAck),(0,CapReject)]
56	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapReject),(0,CapAck)]
57	1`2	1`(0,idle)	1`(0,idle)	1`[]	1`[(1,CapReject),(0,CapReject)]

If a timeout occurs after the response has been sent out by the incoming CESE, then the outgoing CESE can recognise the expired response correctly and discard it. Also, the `CapRelease` message sent after the occurrence of `REJECTindPout` can be ignored by the incoming CESE. For example, when the sequence number is 1 (i.e. the first capability exchange), nodes 5 and 6 (Fig. 4) are the states where the response has been sent out by the incoming CESE and the outgoing CESE is waiting for the response. Following these two states, if `REJECTindPout` occurs, the system transits to state 10 or 11 respectively. The deferred responses eventually are discarded (when the CESE is AWAITING RESPONSE) or ignored (when the CESE is IDLE) properly by the outgoing CESE (via the occurrences of `DISCARD` or `IGNOREout`). Also, the `CapRelease` messages are ignored by the occurrence of `IGNOREin`. Similarly, nodes 28 and 29 are the two states where the sequence number is 0, the response has been sent out and the outgoing CESE is waiting for them. If we follow the OG carefully and refer to the state table, we can also see that, if `REJECTindPout` occurs, these responses can also be discarded or ignored properly.

If a timeout occurs and the incoming CESE has received a TerminalCapabilitySetRelease message in time, i.e. before it receives the response from its user, then transition `REJECTindPin` can occur to inform its user to terminate the current exchange. Therefore no response is generated by the incoming side. The sequence accompanied with a dashed line illustrates two successive exchanges which are terminated by both ends.

In general, we can conclude from the OG that, under the limitation of Model B, this protocol can perform well.

4 The CES Protocol Operating over an Unreliable Transport Layer

Although in an H.323 system the CES protocol operates over a reliable H.245 control channel, as modelled and analysed above, it is interesting as well to investigate this protocol's behaviour when the transport medium is not reliable. For example, `REJECTindUin` and `TRANSFERindT` of the Model B discussed in Sections 3.4.3 and 3.4.4 are dead transitions, are they also dead transitions when the underlying medium is unreliable ? In other words, why the protocol function modelled by these two transitions is designed ?

As specified in H.323, an unreliable transport medium may have message loss, duplication, and the ordering of the messages sent may not be preserved. In the following, we will discuss the CPN model and analyse the properties of the CES protocol using such an unreliable transport medium. Also, the assumption regarding the CES protocol definition made in Section 3.1.2 is also applicable to the CPN model in this section.

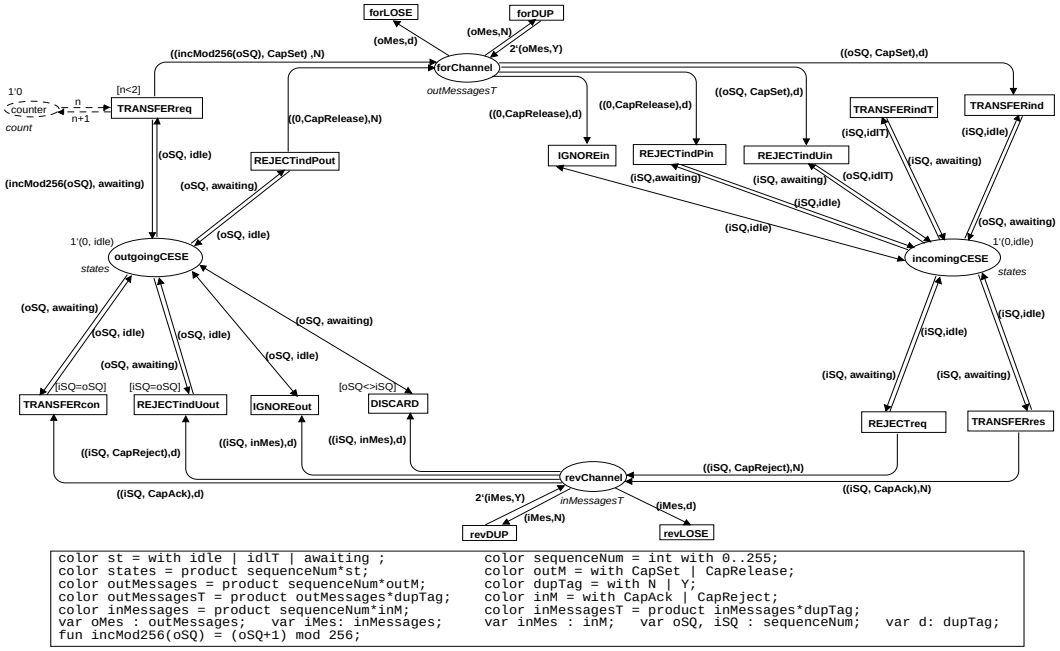
4.1 Modifications of the CPN Model

In the case of an unreliable transport medium, the CPN model of both the outgoing CESE and the incoming CESE is still the same as that of the reliable medium case. To model the unreliable transport medium, we modify the middle part of the CPN model of Fig. 2 (including the arcs connecting places `forChannel` and `revChannel` and the outgoing CESE and incoming CESE parts) to obtain a new CPN model as shown in Fig. 5. Again, the dashed part attached to transition `TRANSFERreq` and the guard of this transition are ignored for the moment.

For each transfer direction, one place and two transitions are used to model the characteristics of the channel. The messages sent may be overtaken, lost or duplicated in the channels. In the following, we use the channel in the forward direction (i.e. from the outgoing side to the incoming side) as an example to illustrate how these two transitions and one place model the properties of this unreliable channel. For the reverse direction, a similar modelling mechanism is used.

4.1.1 Modelling of Message Overtaking

In Fig. 5, the messages sent by the outgoing CESE are first put into place `forChannel`. This place no longer stores a list of messages, and thus does not preserve message order. The colour set of this place is `outMessagesT`, a *product* of colour `outMessages` and colour `dupTag`. The colour `outMessages` is the same as the `outMessages` in Fig 2. It specifies the sequenceNumber field and the content of each message sent by the outgoing CESE. The colour `dupTag` is used as a duplication tag of a message.




```
color st = with idle | idlT | awaiting ;
color states = product sequenceNum*st;
color outMessages = product sequenceNum*outM;
color outMessagesT = product outMessages*dupTag;
color inMessages = product sequenceNum*inM;
var oMes : outMessages;   var iMes: inMessages;
fun incMod256(oSQ) = (oSQ+1) mod 256;
color sequenceNum = int with 0..255;
color outM = with CapSet | CapRelease;
color dupTag = with N | Y;
color inM = with CapAck | CapReject;
color inMessagesT = product inMessages*dupTag;
var inMes : inM;   var oSQ, iSQ : sequenceNum;   var d: dupTag;
```

Fig. 5. CPN model of the CES protocol with unreliable transport layer

4.1.2 Modelling of Message Duplication

The transition `forDUP` is used to duplicate the messages sent to `forChannel`. In practice, a message can rarely be duplicated multiple times, and allowing arbitrary times of duplication can result in an infinite state space of the model, so we assume that all the messages can only be duplicated once at most. To implement this, a duplication tag is attached to each message, i.e. the `dupTag` item of colour `outMessagesT`. When a message is sent by the outgoing CESE, its duplication tag is set to `N` (which means that the message has not been duplicated by the channel yet). When messages are put into place `forChannel`, they may be duplicated via the occurrence of transition `forDUP`. Once a message is duplicated, the duplication tags of the message and its copy are set to `Y` (which means that the message has already been duplicated). Only messages with duplication tag `N` can be duplicated so that each message can be duplicated only once.

4.1.3 Modelling of Message Loss

This property of the channel is modelled by transition `forLOSE`. Messages and their duplicates are lost from the `forChannel` when transition `forLOSE` occurs.

4.2 State Space Analysis

The model in Fig. 5 also has an infinite state space because of the unlimited occurrence sequence of transition `TRANSFERreq` followed by transition `REJECTinPout`. It is also obvious that when the underlying medium is unreliable this protocol also could fail if the wrapping of sequence numbers can happen. In order to make the state space tractable to investigate the protocol properties of interest, as with Model B discussed in Sections 3.4.3 and 3.4.4, we also limit the maximum number of times transition `TRANSFERreq` can occur to 2. As before, this is realised by a counter modelled by the dashed part and the guard of transition `TRANSFERreq` in Fig. 5. The definitions of colour set `count` and variable `n` is the same as those for Model B, and these definitions again are not included in the declaration node of the model in Fig. 5.

Table 6 shows part of the state space report. We see that even though we only allow two sequence numbers and only one round of them, the state space is large, with 40011 nodes and 385314 arcs. The result was obtained on another machine with a Pentium 733MHz CPU and 256MB of memory. Figure

6 shows part of the OG and it is used in the following to illustrate the analysis results. In this figure, again all the names of the transitions and places are abbreviated as in Fig. 3. Those abbreviations which do not appear in Fig. 3 will be explained when they are used.

Table 6. Part of the state space report for the limited model in Fig. 5

	Occurrence Graph		SCC Graph	
Statistics	Nodes:	40011	Nodes:	40011
	Arcs:	385314	Arcs:	385314
	Secs:	497	Secs:	69
	Status:	Full		
Home Properties	Home Markings:	None		
Liveness Properties	Dead Markings:	[575, 583]		
	Dead Transition Instances:	None		
	Live Transition Instances:	None		

It is immediately found from the liveness properties that there is no dead transition instance, which means that `REJECTindUin` and `TRANSFERindT` can occur when the transport medium is unreliable. We can imagine, for example, if a `CapRelease` message is lost in the `forChannel`, it is possible that when `awaiting` is in place `incomingCESE` and another `CapSet` message is received, transition `REJECTindUin` is enabled, and thus can occur. The branch marked with (3) in Fig. 6 shows the occurrence sequence of this example. It explains why this mechanism is designed for the CES protocol.

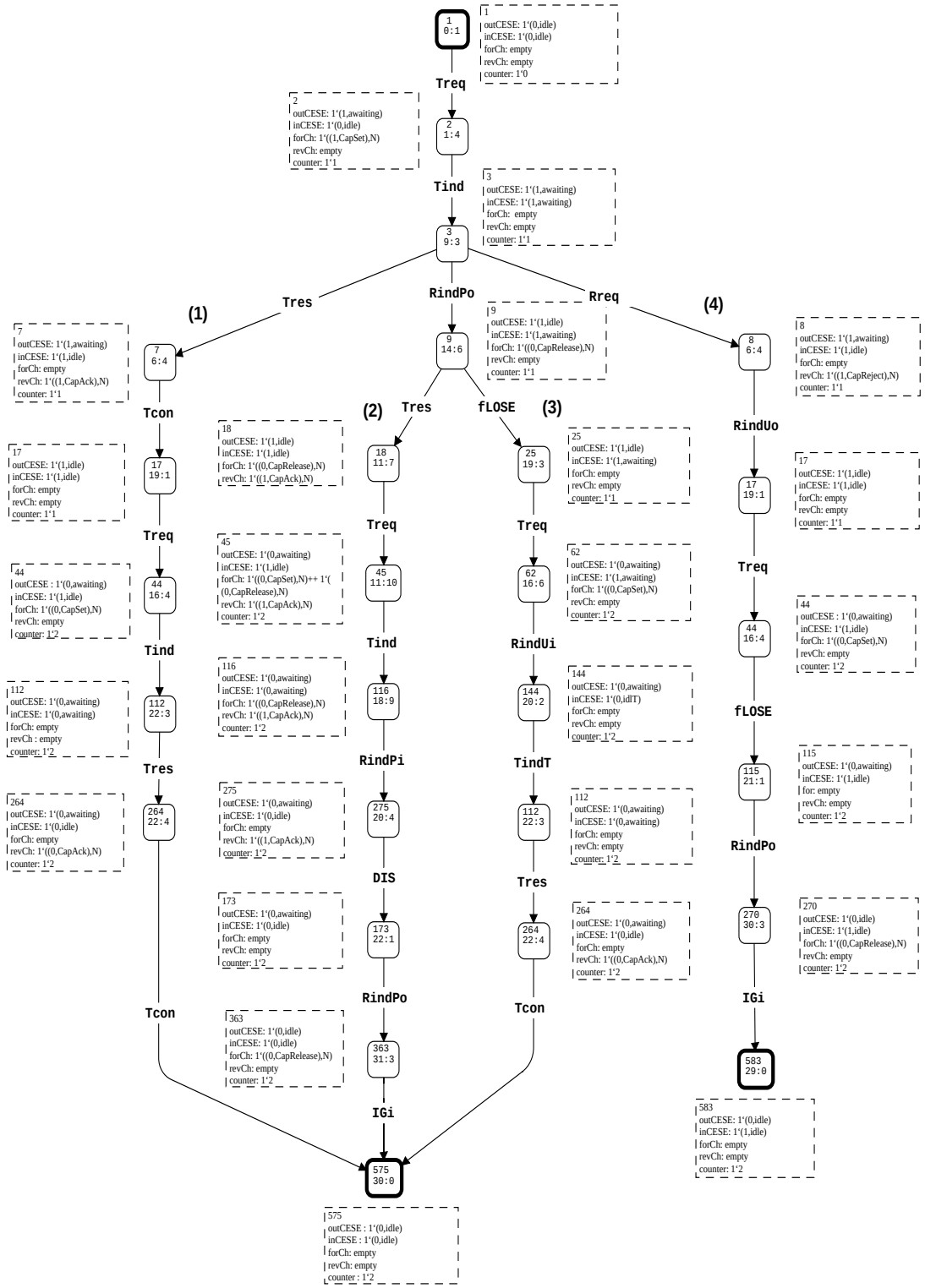
There are two dead markings, nodes 575 and 583, as reported in Table 6. Referring to Fig. 6, we can see that node 575 is an expected terminal state where there are no messages left in the channel, both the outgoing and incoming CESEs are IDLE with the state variable zero and the counter at its limit. For node 583, there is also no message left in either channel and both the outgoing and incoming CESEs are IDLE. The only difference from node 575 is that the state variable *in_SQ* has value 1. As we know, however, as long as *out_SQ* has been set back to its initial value, 0, it does not matter which value *in_SQ* can have. So node 583 is also a reasonable terminal state.

Terminal state, node 583, is caused by the lossy channel. For example, as shown in Fig. 6, starting from the initial state (node 1), a transfer is initiated (Treq), the incoming CESE receives a `CapSet` message (with sequence number 1) successfully and informs its user (Tind). State variable *in_SQ* is set to 1 (node 3). If we look at the branch marked with (4), the incoming user refuses this capability set (Rreq), and a `CapReject` message is sent to the outgoing CESE. The outgoing user is notified of this rejection (RindUo, i.e. transition `REJECTindUout`) (node 17). Then a second capability exchange is initiated (Treq), but the `CapSet` message is lost in the `forChannel` (fLOSE for `forLOSE`). A timeout occurs at the outgoing side (transition `REJECTindPout` (RindPo)) and a `CapRelease` message is sent to the incoming CESE. Because there is an `idle` in place `incomingCESE`, this release message is ignored (IGi) and the `idle` stays in `incomingCESE`, with state variable kept as 1, as shown in marking 583.

The OG for this model is too large to be drawn and visualised fully, however, it is possible to illustrate sequences of transitions and the related states of interest. In the following, we describe more results observed from the partial OG shown in Fig. 6.

The branch marked with (1) gives the sequence of two successful capability exchanges, which demonstrates desired behaviour.

However, there are some other sequences which are not ideal, e.g. the sequence marked with (2) in Fig. 6. After the state of the model moves to node 3 as in other branches, transition `REJECTindPout` occurs, a `CapRelease` message is sent to the `forChannel`, and the state transits to node 9. If `TRANSFERres` occurs, a `CapAck` message is put into place `revChannel` (node 18). Since now there is an `idle` in place `outgoingCESE`, transition `TRANSFERreq` occurs again, a `CapSet` message with sequence number 0 is sent to the `forChannel` (node 45). This `CapSet` message overtakes the `CapRelease` message and is passed to the incoming CESE (Tind and node 116). So the `CapRelease` message is still left in the `forChannel`. Following this, the `CapRelease` message is received by the incoming side and terminates the current exchange (node 275), but this `CapRelease` message was sent by the outgoing side to terminate the first exchange. Then the response to the first `CapSet` message is transferred to the outgoing side. Fortunately, the state variable *out_SQ* has value 0, i.e. the outgoing CESE is waiting



Note: in order to show the individual sequences clearly, some nodes are duplicated in this figure (e.g. node 264).

Fig. 6. Partial OG of the limited model in Fig. 5

for the response to the second `CapSet` message with sequence number 0, so the old `CapAck` message with sequence number 1 is discarded (DIS). It is not possible for the sequence numbers to be wrapped under the limitation that we have made on this model. Therefore, transition `REJECTindPout` has to occur, a `CapRelease` message is sent to place `forChannel` (node 363), transferred to the incoming side, which ignores it and the system moves to the terminal state, node 575.

From this occurrence sequence we can see that if we limit the occurrence times of transition `TRANSFERreq`, it is not possible for the outgoing side to accept the wrong response, so no fatal error occurs. We also have observed that the exchange was terminated improperly due to message overtaking. Although no response has been accepted incorrectly, no successful capability exchange has occurred. This shows that this protocol does not allow the wrong acceptance of a response (if the problem of wrapping of sequence numbers is ignored), nevertheless it may not work efficiently.

5 Conclusion and Future Work

As the pilot study of the research project proposed in [11], this paper has modelled and analysed the CES protocol of H.245 with CPNs. The following results have been obtained.

- The complete CPN models of this protocol when it operates over both a reliable and unreliable transport layer have been created. These CPN models remove the ambiguity in the CES protocol definition given in H.245, thus they provide a more rigorous specification of the CES protocol.
- By analysing these models, we found that:
  - When the wrapping of sequence numbers (modulo 2) is not possible while there are 2 outstanding acknowledgments (i.e. not yet received by the outgoing CESE), this protocol can carry out the capability exchange function properly. An induction proof is needed to generalise this conclusion for arbitrary modulo arithmetic.
  - This protocol could fail if wrapping of the sequence numbers can happen, either when the underlying medium is reliable or not. However, this protocol could work properly if the protocol had a mechanism for not sending out a TerminalCapabilitySet message if there are n outstanding acknowledgments, for modulo n arithmetic. Then it could time-out and report a problem to the user, rather than operating incorrectly.
  - If wrapping of sequence numbers is not possible, then when the transport medium is unreliable, this protocol can work, but may be inefficient.

As the next step, we plan to verify the CES protocol against its service specification. However the service definition of the CES protocol is not defined completely in H.245. We have created a complete and general CES service specification in the form of a CPN model [12]. Currently, we are working on generating the CES service language from this service model. After the CES service language is generated, the CES protocol can be verified against the CES service specification.

Based on the experience and methodology gained from this pilot study, we shall investigating other parts of H.323 with CPNs. Hopefully, the work on H.323 will contribute to the development of Internet multimedia standards.

References

1. M. Diaz and P. Sénac. Time Stream Petri Nets, A Model for Timed Multimedia Information. In *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, June 1994.
2. Design/CPN homepage. http://www.daimi.au.dk/designCPN/.
3. ITU-T. *ITU-T Recommendation H.310, Broadband audiovisual communication systems and terminals*, September 1998.
4. ITU-T. *ITU-T Recommendation H.324, Terminal for low bit-rate multimedia communication*, February 1998.
5. ITU-T. *ITU-T Recommendation H.225.0, Call Signalling Protocols and Media Stream Packetization for Packet-Based Multimedia Communication Systems*, September 1999.
6. ITU-T. *ITU-T Recommendation H.323, Packet-Based Multimedia Communications Systems*, September 1999.

7. ITU-T. *ITU-T Recommendation Z.100, Specification and Description Language (SDL)*, November 1999.
8. ITU-T. *ITU-T Recommendation H.245, Control protocol for multimedia communication*, November 2000.
9. ITU-T. *MEDIACOM 2004, A Framework for Multimedia Standardization, Project Description - Version 1.1*, March 2001.
10. Kurt Jensen. *Coloured Petri nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1.* Springer, second edition, 1997.
11. Lin Liu. Modelling and Analysis of Internet Multimedia Communication Systems. PhD Research Proposal, University of South Australia, July 2000.
12. Lin Liu and Jonathan Billington. Investigating the CES Protocol of H.245 Using CPNs - A Pilot Study of the H.323 Standard. Technical report, Computer Systems Engineering Centre, University of South Australia, May 2001.
13. Lin Liu and Jonathan Billington. Modelling and Analysis of Internet Multimedia Protocols - Methodology and Initial Results. In *Proceedings of INCOSE2001, the Eleventh Annual International Symposium of the International Council on Systems Engineering*, Melbourne, Australia, 1-5, July 2001.
14. P. Sénac, M. Diaz, A. Léger, and P.de Saqui-Sannes. Modeling Logical and Temporal Synchronization in Hypermedia Systems. *Journal on Selected Areas in Communications*, 14(1):84–103, January 1996.
15. H.323 Information Site. http://www.packetizer.com/iptel/h323/.
16. M.E. Villapol and J. Billington. Modelling and Initial Analysis of the Resource Reservation Protocol using Coloured Petri Nets. In *Proceedings of the Workshop on Practical Use of High-Level Petri Nets, Aarhus, Denmark*, pages 91–110, June 2000.

An Improved Architectural Specification of the Internet Open Trading Protocol*

Chun Ouyang, Lars Michael Kristensen**, and Jonathan Billington

Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
chuoy001@students.unisa.edu.au {lars.kristensen,jonathan.billington}@unisa.edu.au

Abstract. The *Internet Open Trading Protocol* (IOTP) is an international standard (RFC 2801) currently being developed by the Internet Engineering Task Force. IOTP aims at providing an interoperable framework for electronic commerce (e-commerce) over the Internet. IOTP is expected to evolve into one of the central building blocks for the developing of next generation e-commerce on the Internet. We apply Coloured Petri Nets and Design/CPN for modelling and analysing IOTP, focussing on its protocol architecture. The contribution of this paper is the construction of a CPN specification of RFC 2801. Based on the constructed CPN model, we propose a complete and simplified architecture of IOTP compared with the partial architectural specification given in the RFC. We apply simulation, message sequence charts, and state space analysis to validate the CPN models of IOTP exchanges and transactions, and to validate that the suggested protocol architecture conforms to the specification of IOTP given in the RFC.

1 Introduction

Today's Internet is revolutionising commerce. It provides the first affordable and secure way to link people and computers spontaneously across organisational boundaries. Electronic commerce (e-commerce), traditionally conducted with the use of information technologies based on *Electronic Data Interchange* (EDI) over proprietary *Value Added Networks* (VAN), is rapidly moving onto the Internet [17]. With the advent of Internet-based e-commerce, such as online shopping and online retailing, a multitude of technology standards and specifications have been (or are being) developed to build the required communication infrastructures.

The *Internet Open Trading Protocol* (IOTP) [3, 4] is a standard being developed by the Internet Engineering Task Force (IETF) [7], and is expected to evolve into one of the central building blocks for the developing of next generation e-commerce on the Internet. IOTP aims at providing an interoperable framework for e-commerce over the Internet. It is designed to specify the communication protocols to complete an electronic business transaction between two parties which have no prior association. The initial focus of IOTP is on the payment and delivery aspects of business-to-consumer (b-c) e-commerce.

Concurrently with the specification work being conducted within IETF, a number of research groups and companies are planning and working on the first trial implementations of selected parts of IOTP. This includes the project of the Open Trading Protocol toolkit for Java

* Supported by (1) Australian Technology Network (ATN) Small Research Grant, and (2) University of South Australia Divisional Small Grant.
** Supported by the Danish Natural Science Research Council.

(JOTP) in Xenosys Corporation [9], and the project of the Standard sMart card Integrated settLEment system (SMILE) [14] conducted by Hitachi [8]. There is however still no complete implementation of IOTP, and an interoperability test between independent implementations has not yet been conducted. As an emerging communication protocol, the development of IOTP is still in an early stage and can therefore benefit from the use of formal methods for modelling and analysis before becoming an Internet standard. Moreover, there are several aspects of IOTP that need development. First of all, IOTP lacks a detailed specification of its protocol architecture and as a consequence the internal organisation of IOTP needs clarification. Apart from that, RFC 2801 contains several ambiguities in the specification of the IOTP protocol itself.

In this paper we present our results from applying Coloured Petri Nets (CP-nets or CPNs) [10, 11] and the Design/CPN tool [13] for the modelling and analysis of IOTP. The primary focus of the work presented has been on the specification of the IOTP protocol architecture based on the modelling of the IOTP *baseline transactions*. The set of baseline transactions is what constitutes the main service provided by IOTP. The construction of the CPN models has identified the important protocol layers and components, and clarified their relationship.

This paper is organised as follows. Section 2 gives a brief overview and introduction to IOTP using the IOTP purchase transaction as an example. Section 3 gives an overview of the CPN model of IOTP. Sections 4 and 5 explain how the set of IOTP transactions and exchanges have been modelled. Section 6 presents the simulation and the state space analysis results. Section 7 presents the IOTP protocol architecture derived from the constructed CPN model. Finally, in Section 8 we summarize our contribution and discuss future work. The reader is assumed to be familiar with Coloured Petri Nets and Design/CPN.

2 The Internet Open Trading Protocol

In this section we introduce the basic concepts of IOTP. Baseline IOTP [3] defines eight transactions: *Authentication*, *Deposit*, *Withdrawal*, *Purchase*, *Refund*, *Value Exchange*, *Inquiry*, and *Ping*. The first six transactions are designed for business interactions and will be referred to as *trading transactions*[1]. The last two transactions are independent of business processes and are so-called *infrastructure transactions*. Baseline IOTP trading transactions are consumer oriented. The Authentication transaction supports the authentication of a party involved in a transaction to validate that the party is who it claims to be. The Purchase transaction is carried out for the purchase of goods or services using some payment method. The Refund transaction occurs when the refund of a payment is required, usually as a result of an earlier purchase. The Deposit and the Withdrawal transactions are used for the deposit or the withdrawal of electronic cash at a financial institution. Finally, Value Exchange transaction supports the transfer or the conversion of an amount of electronic cash in one currency using one payment method to an amount of electronic cash in the same or another currency using the same or another payment method. The Inquiry transaction can be used to obtain information on the status of an ongoing or completed trading transaction, while the Ping transaction tests basic connectivity between parties involved in a trading transaction. Below we use the Purchase transaction to illustrate the basic operation of IOTP.

[1] In RFC 2801 [3], they are referred to as Authentication- and Payment-related IOTP Transactions (A-P Transactions).

2.1 IOTP Purchase Transaction

The Purchase transaction can be used when one party wants to purchase goods or services from another party over the Internet. Figure 1 shows a possible sequence of messages exchanged between the parties involved in a Purchase transaction. Each column of the Message Sequence Chart (MSC) corresponds to a so-called *trading role*. Trading roles are used to identify the different roles that organisations can play in a trade. IOTP defines five trading roles: *Consumer*, *Merchant*, *Payment Handler*, *Delivery Handler*, and *Merchant Customer Care Provider*. The *Consumer* is the organisation which receives and pays for the goods/services. The *Merchant* supplies the goods and receives payment for them. The *Payment Handler* is the organisation receiving the money from the *Consumer* on behalf of the *Merchant*. The *Delivery Handler* is the organisation responsible for delivering the goods to the *Consumer* on behalf of the *Merchant*. Finally, the *Merchant Customer Care Provider* (which we will not consider further in this paper) is the organisation responsible for resolving *Consumer* disputes and problems on behalf of the *Merchant*.

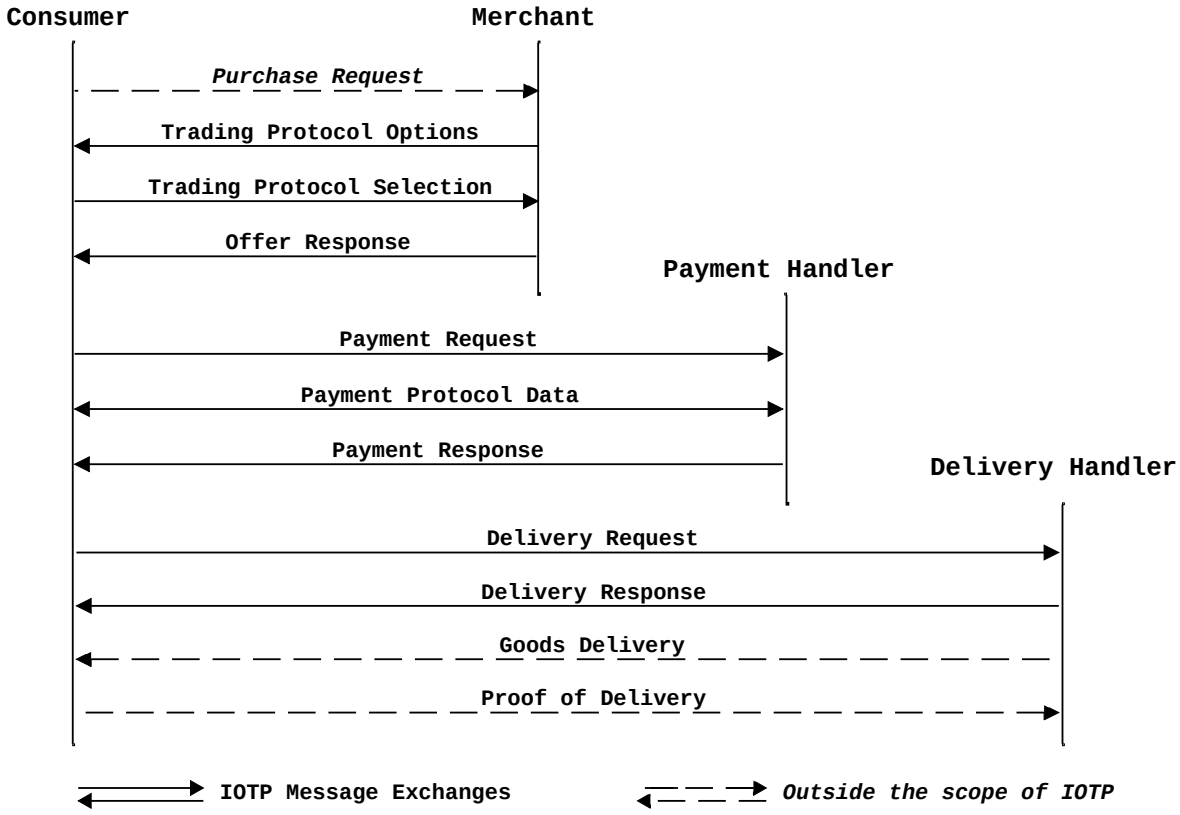


Fig. 1: A possible message flow in a Purchase transaction.

In the following we consider in detail the messages exchanged between trading roles in a Purchase transaction as shown in Fig. 1. The first phase in the Purchase transaction is the negotiation between the Consumer and Merchant regarding the payment brand and the payment protocol to be used. At some point the Consumer decides to buy goods from the Merchant, and sends a Purchase Request to the Merchant by, e.g., clicking on a button in a

web browser. Whereas this is outside the scope of IOTP, it enables the start of the Purchase transaction at the other side, i.e., the Merchant side. The Merchant then starts the Purchase transaction upon receiving the Purchase Request from the Consumer. The Merchant offers a list of Trading Protocol Options to the Consumer. The Trading Protocol Options include a list of payment brands (e.g., Visa Credit, MasterCard, Mondex card) that are accepted by the Merchant and the correspondant payment protocols available (e.g., SET[2], Mondex VTP[3]). The Consumer selects the payment brand (e.g. MasterCard) and the payment protocol (e.g. SET) among the options offered by the Merchant, and sends them back to the Merchant in a Trading Protocol Selection. The Merchant uses the selection made by the Consumer and the related information to create the Offer Response, and sends it to the Consumer. The Offer Response contains details of the goods, pay amount, and delivery instructions. More generally, the Offer Response can be considered as an invoice before the actual payment is carried out.

The next phase in the transaction is the payment. After checking the payment information contained in the Offer Response, the Consumer sends a Payment Request to the Payment Handler. The Payment Handler checks the information provided (such as a signature) in the Payment Request. If the information is valid, the payment is carried out via the exchange of Payment Protocol Data between the Consumer and the Payment Handler. This data could be SET protocol data if the SET protocol was selected as the payment protocol. When the payment exchanges finish, the Payment Handler sends a Payment Response which includes the payment receipt and an optional signature. This provides the Consumer with proof of the payment.

The final phase in the transaction is the delivery of the goods. The Consumer checks the delivery instructions in the Offer Response, and uses the payment receipt as authorisation in the Delivery Request that is sent to the Delivery Handler. The Delivery Handler starts or schedules the delivery, and sends a delivery note in the Delivery Response to the Consumer. The delivery of the goods might be a physical delivery, or an on-line delivery if the goods are electronic such as an electronic journal. Finally, the Consumer sends Proof of Delivery to the Delivery Handler upon receiving the goods. It should be mentioned that the last two steps in the phase of delivery, i.e., the actual delivery of goods/service and the proof of delivery, are outside the scope of IOTP.

From the above it can be seen that a Purchase transaction is conducted in three phases. At first, the choice of available payment brand and payment protocol is negotiated between Consumer and Merchant. Secondly, the payment is made between Consumer and Payment Handler. Thirdly, the delivery occurs between Consumer and Delivery Handler. A Purchase transaction can therefore be seen as being divided into three sub-transactions. IOTP defines two sets of such sub-transactions called *trading exchanges* and *document exchanges*. Each document and trading exchange involves a set of messages exchanged between trading roles. All eight IOTP baseline transactions can be expressed as combinations of such document and trading exchanges. Document exchanges are constructed from trading exchanges by grouping together parts of trading exchanges. One of the contributions of this paper is to merge the set of trading and document exchanges into one common set which we call *IOTP exchanges* leading to a simpler specification and architecture of IOTP.

[2] Secure Electronic Transaction (SET) is an open technical standard developed by Visa and MasterCard to facilitate secure payment card transactions on the Internet [16].

[3] Mondex Value Transfer Protocol (VTP) enables a secure and legitimate transfer between two Mondex cards [15].

3 Overview of CPN IOTP Model

A CPN model has been constructed for each of the six trading transactions of IOTP. All six CPN models have a similar structure and share a common set of pages. All the CPN models could in principle be integrated into one single CPN model capturing all trading transactions. This has however not yet been done. We therefore focus on the CPN model of the Purchase transaction. The CPN models of the other trading transactions are similar. Since the Purchase transaction involves all types of IOTP exchanges, it can be considered a representative example.

Figure 2 shows the hierarchy page of the CPN model. The prime page Purchase provides the most abstract view of the Purchase transaction and has five subpages. Four of these subpages: Consumer, Merchant, PHandler, and DHandler correspond to the four trading roles involved in a Purchase transaction, and they specify how a Purchase transaction is implemented for each of the trading roles. We will refer to these pages as trading role pages. The prime page and the trading role pages constitute the transaction layer in the protocol architecture of IOTP which we will specify in Sect. 7. The specification of the transactions are presented in detail in Sect. 4. The last subpage of Purchase is Transport. It models the transport medium over which the trading roles communicate. In this paper we will not go into detail with the modelling of the transport layer.

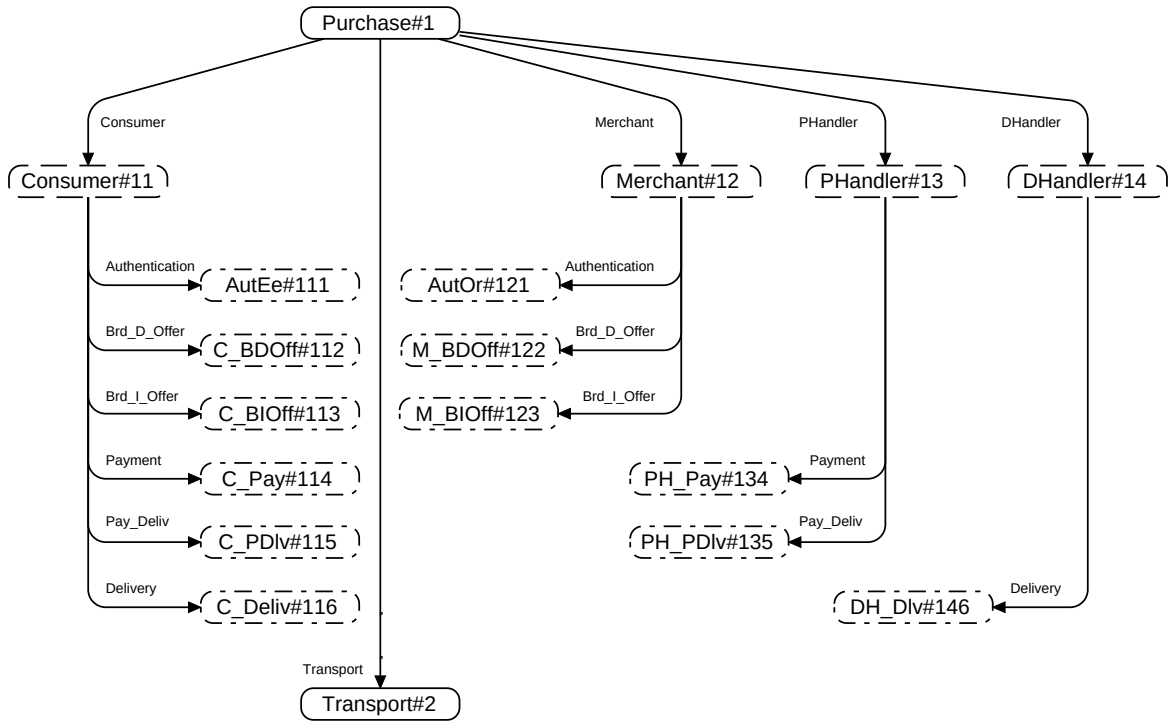


Fig. 2: The hierarchy page of the CPN Purchase transaction model.

The 12 subpages of the four trading role pages specifies the IOTP exchanges in a Purchase transaction. Baseline IOTP defines six document exchanges: *Authentication*, *Brand Dependent Offer*, *Brand Independent Offer*, *Payment*, *Delivery*, and *Payment with Delivery*. The Authentication exchange (subpages AutEe and AutOr) allows one trading role (the

authenticator) to authenticate another trading role (the authenticatee), i.e. to validate that the organisation is the one it claims to be. The Brand Dependent Offer exchange (subpages C_BDOff and M_BDOff) allows the Merchant to provide a list of payment brands to the Consumer for selection. The selected payment brand is to be used to carry out the payment. The Brand Independent Offer exchange (subpages C_BIO and M_BIO) allows the Merchant to specify (without Consumer selection) which payment brand is to be used for the payment. The Payment exchange (subpages C_Pay and PH_Pay) supports payment using some payment brand to be made by the consumer to the payment handler. The Payment with Delivery exchange (pages C_PDlv and PH_PDlv) supports a combined payment and delivery. The Delivery exchange (subpages C_Deliv and DH_Deliv) supports the delivery of goods. The IOTP exchanges are combined to implement the different transactions. For example, the Purchase transaction shown in Fig. 1 consists of a Brand Dependent Offer, a Payment, and a Delivery exchange. Considering that each IOTP exchange is defined as a bilateral business interaction between two trading roles, each exchange has been modelled as a pair of subpages – one for each trading role side. For example, the subpages C_Pay and PH_Pay represent the payment exchange carried out between the Consumer and the Payment Handler trading role. We will go into more detail with the specification of the IOTP exchanges in Sect. 5.

3.1 Trading Roles and Messages

An IOTP transaction consists of a set of *IOTP Messages* exchanged between trading roles. Figure 3 depicts the prime page Purchase. This page has a substitution transition for each of the four trading roles in a Purchase transaction. The substitution transition Transport represents the transport medium by means of which the trading roles communicate. Each trading role has two associated places modelling an input and an output message buffer.

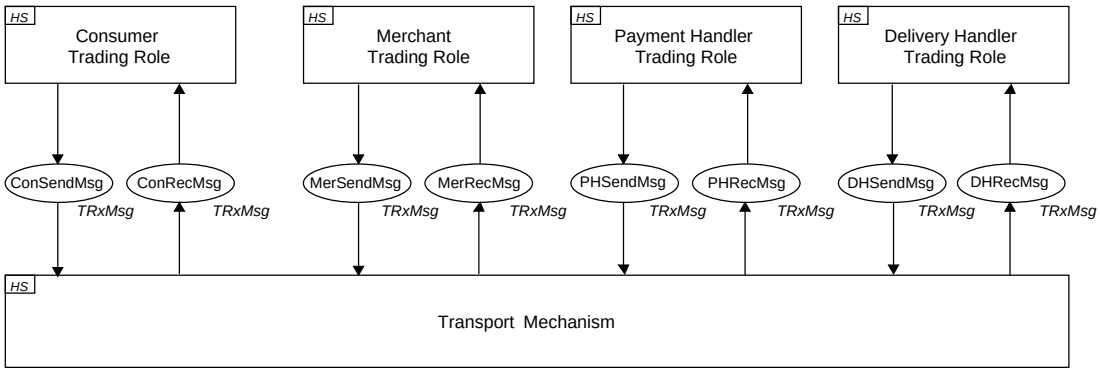


Fig. 3: Top level structure of IOTP Purchase transaction.

Figure 4 lists the definition of the colour sets used to model IOTP messages. The colour set TRxMsg (line 5) is used to model the message buffers containing IOTP messages to be transmitted or received. For an example, a token residing in a place that models a buffer for sending messages (e.g., ConSendMsg) specifies the message receiver trading role and the message itself. In the other case, a token residing in a place that models a buffer for receiving messages (e.g., ConSendMsg) specifies the message sender trading role and the message itself.

IOTP messages are modelled by the colour set IotpMsg (line 4) which is derived from the XML (eXtensible Markup Language) definition of IOTP messages given in [3]. An IOTP

message consists of a list of so-called *Trading Blocks* modelled by the colour set TradingBlk (line 3). The colour set ProcessState (line 2) represents the *ProcessState* attribute of an *Authentication Status Block*. It contains two values, indicating whether the result of the Authentication exchange is successful (CompletedOk) or has failed (Failed). The representation of IOTP messages in the CPN model is a simplified and abstract representation of IOTP messages as specified in [3]. The reason is that not all attributes of an IOTP message are required at the level of abstraction where the trading transactions are being modelled here.

```
1   color TradingRole = with Consumer | Merchant | PHandler | DHandler;

2   color ProcessState = with CompletedOk | Failed;

3   color TradingBlk = union AuthReq + AuthResp + AuthStatus: ProcessState
                           + TPO + TPOSelection + OfferResp
                           + PayReq + PayExch + PayResp
                           + DeliveryReq + DeliveryResp
                           + Cancel;

4   color IotpMsg = list TradingBlk;

5   color TRxMsg = product TradingRole * IotpMsg;
```

Fig. 4: Colour sets for modelling IOTP trading roles and messages.

4 IOTP Transaction Layer

The IOTP transactions are implemented via combinations of IOTP exchanges at each trading role side. The Purchase transaction is the most complex trading transaction involving four trading roles and covers six IOTP exchanges. The transaction level pages in the CPN model are composed of four trading role pages. Each trading role page describes how the trading role operates in combining the IOTP exchanges to the Purchase transaction. In the following subsections we consider each of the trading roles at the transaction layer in detail.

4.1 Consumer Trading Role

The IOTP transactions are consumer oriented. Therefore, the Consumer is the most important trading role in the Purchase transaction. As defined in IOTP, the bilateral communications during the Purchase are always carried out between the Consumer and one of the other three trading roles. As a result, all the procedures for the Purchase transaction are performed at the Consumer side. Figure 5 depicts the page Consumer modelling the Consumer side of a Purchase transaction. IOTP defines the valid combinations of IOTP exchanges to implement an IOTP transaction. In Fig. 5, the IOTP exchanges that are part of a Purchase transaction are all represented by means of substitution transitions. A purchase transaction consists of an optional Authentication exchange, followed by an Offer exchange, and then either a Payment exchange followed by a Delivery exchange, a Payment exchange only, or a Payment with Delivery exchange.

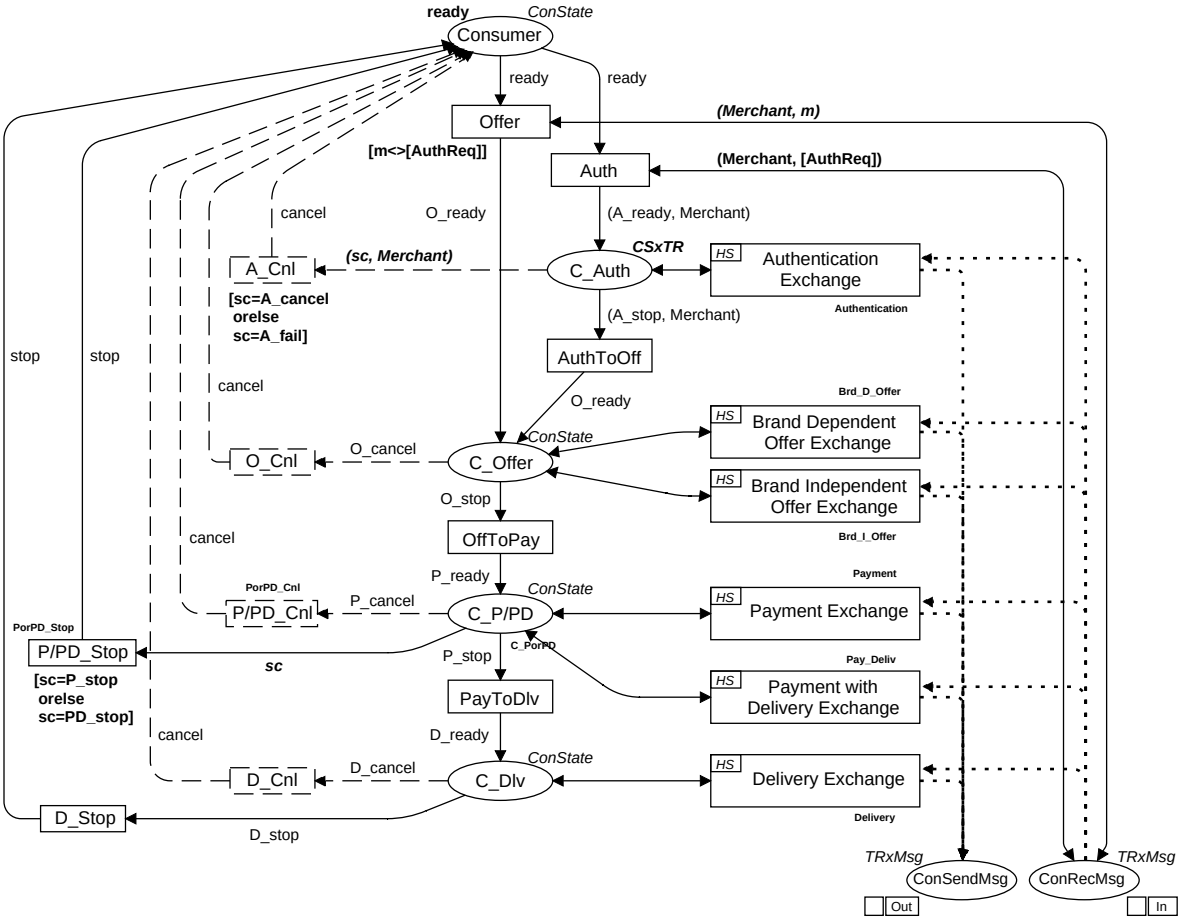


Fig. 5: Purchase transaction – Consumer trading role.

The five places Consumer, C_Auth, C_Offer, C_PayDlv, and C_Deliv are used to model the state of the Consumer during the execution of the Purchase transaction. The place Consumer (top) models the initial state ready and the two terminal states cancel and stop for a transaction. The other four places model the states of the Consumer in the six different IOTP exchanges involved in the transaction. Except for the arcs between the places and the substitution transitions, each input or output arc of these places has a constant in the arc expression. The constant represents the initial or the terminal state(s) of an IOTP exchange. The colour set ConState attached to these five places contains all the possible states of the Consumer, while the colour set CSxTR of place C_Auth also specifies the trading role corresponding to the other party. We will return to the definitions of these two colour sets when we explain how the IOTP exchanges have been modelled.

The Consumer is initially in the state ready. If the first IOTP message received from the Merchant is an Authentication Request message represented by a single-element list [AuthReq], an Authentication exchange will start by putting a token with colour (A_ready, Merchant) in place C_Auth, indicating that the Consumer is now ready to be authenticated by the Merchant. Otherwise, the Consumer will directly start an Offer exchange by putting an O_ready token in place C_Offer. The occurrence of the transition Auth or Offer indicates that the Purchase transaction is carried out with an optional Authentication exchange. Moving

onto the Offer exchange, the Consumer will then start a Brand Dependent Offer exchange or a Brand Independent Offer exchange. The socket place C_Offer is common to the two substitution transitions Brand Dependent Offer Exchange and Brand Independent Offer Exchange. This indicates that only one of the two Offer exchanges can occur in a Purchase transaction. A similarly choice is made in the next phase between a Payment exchange and a Payment with Delivery exchange. Finally, the Consumer can start a Delivery exchange (D_ready) only if the last IOTP exchange carried out was a Payment exchange. This is modelled by the transition PaytoDlv with an input arc containing only one token value P_stop, which represents the successful terminal state of a Payment exchange.

The four transitions A_Cnl, O_Cnl, P/PD_Cnl, and D_Cnl are used to collect the terminal state cancel from the four places modelling the Consumer states in the six IOTP exchanges. Occurrences of these transitions indicate that the cancellation of some IOTP exchange results in the cancellation of the whole Purchase transaction at that time. The two transitions P/PD_Stop and D_Stop are used to collect the terminal state P_stop, PD_stop, or D_stop from the two places modelling the Consumer states in the Payment, the Payment with Delivery, or the Delivery exchanges. Occurrences of these transitions indicate the successful completion of the Purchase transaction after any of these three IOTP exchanges.

4.2 Merchant Trading Role

Figure 6 depicts page Merchant which specifies how the Merchant operates during the execution of a Purchase transaction. This page is structured in a similar way to the Consumer page with substitution transitions representing the IOTP exchanges on the Merchant side of the transaction. The Merchant is initially ready and makes a choice as to whether the optional Authentication Exchange should be conducted first. After the optional Authentication exchange there is a choice between doing a Brand Dependent Offer Exchange or a Brand Independent Offer Exchange.

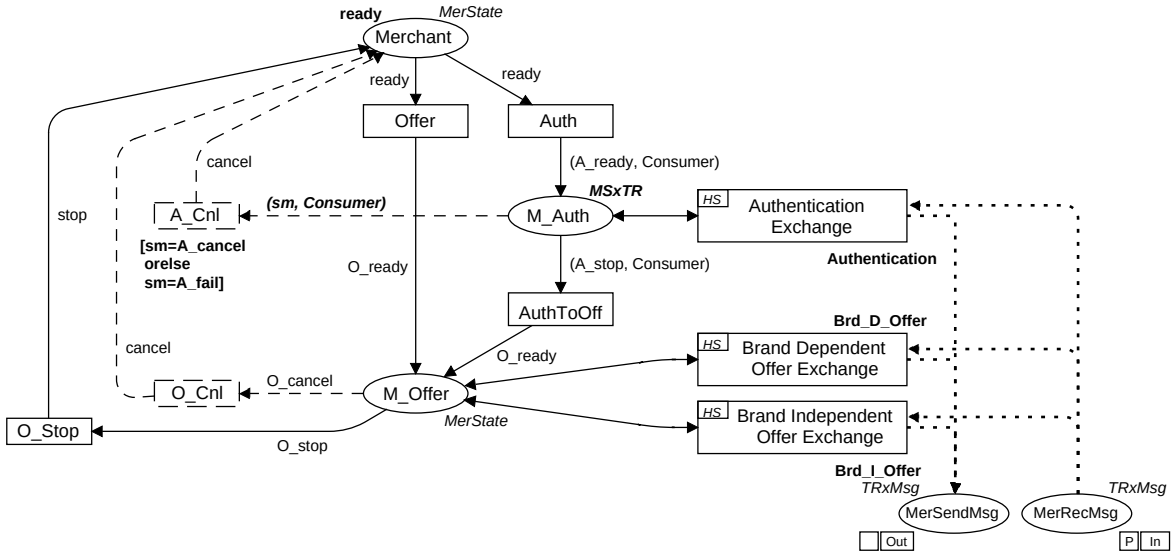


Fig. 6: Purchase transaction – Merchant trading role.

4.3 Payment Handler Trading Role

Figure 7 depicts page PHandler which specifies how the Payment Handler operates in a Purchase transaction. The Payment Handler is initially ready and will start either a Payment Exchange or a Payment with Delivery Exchange once receiving the message [PayReq] from the Consumer.

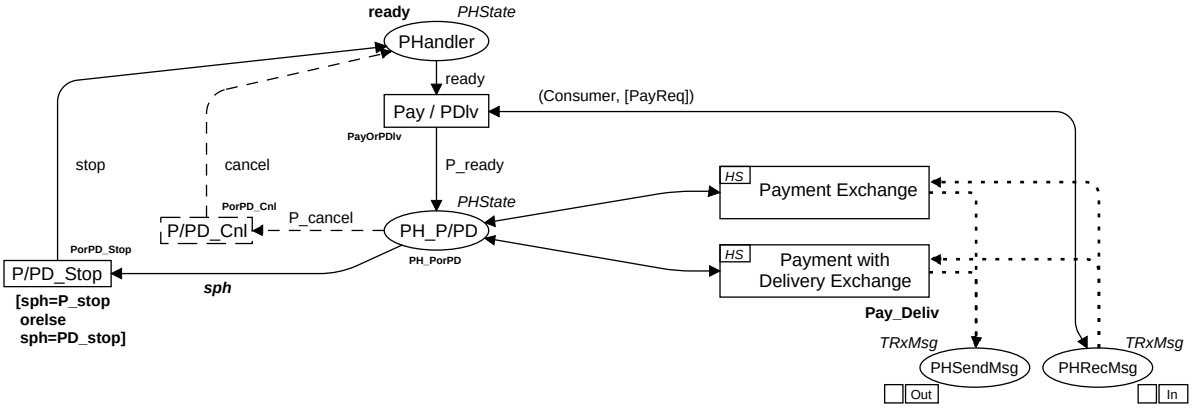


Fig. 7: Purchase transaction – Payment Handler trading role.

4.4 Delivery Handler Trading Role

Figure 8 depicts page DHandler which specifies how the Delivery Handler operates in a Purchase transaction. The Delivery Handler is initially ready and will start the Delivery Exchange as soon as the message [DeliveryReq] is sent from the Consumer indicating a delivery request.

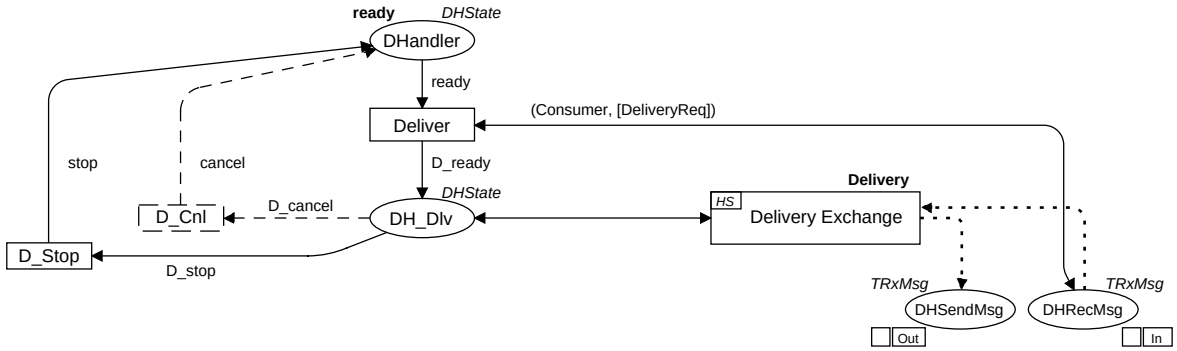


Fig. 8: Purchase transaction – Delivery Handler trading role.

5 IOTP Exchange Layer

We now consider the specification of IOTP exchanges. The same modelling approach has been applied to the six exchanges constituting a Purchase transaction. We therefore only give a detailed account and description of how the Payment exchange has been modelled.

5.1 State Definitions for Trading Roles

First we consider the declarations related to the states of the trading roles. It should be mentioned that there are no direct definitions of states for trading roles in the RFC [3], and they are thus indirectly derived from the RFC based on the messages sent and received by the trading roles. Figure 9 lists definitions of the colour sets relating to the states of trading roles. Colour set State (line 1) contains the states of all trading roles in a Purchase transaction. It has four subsets: ConState (line 2), MerState (line 3), PHState (line 4), and DHState (line 5). Each of them indicates the possible states of the corresponding trading role, i.e., Consumer, Merchant, Payment Handler, and Delivery Handler. The colour sets CSxTR and MSxTR contain a trading role component, and are used only for the Authentication exchange. This is needed because only one party (the Consumer) is fixed in an Authentication exchange whereas the other party could be any of the other trading roles. As part of its state the Consumer therefore needs to know who the other party is. In the following we explain each value in the colour set State in detail.

- ready, cancel or stop represents the initial state, the aborted terminal state, and the successful terminal state, respectively, of a trading role in a transaction. By adding a prefix to the above three states, corresponding states are obtained at the level of IOTP exchanges. For example, if the prefix is set to A for Authentication exchange, A_ready, A_cancel and A_stop then represent the initial state, and the two terminal states of a trading role in an Authentication exchange. Similarly, the prefix O stands for the two Offer exchanges, P stands for both the Payment and the Payment with Delivery exchanges, and D stands for the Delivery exchange. The only exception is the definition of the successful terminal state for the Payment with Delivery exchange. It is defined as PD_stop, which is different from P_stop for the Payment exchange.
- A_fail represents the unsuccessful terminal state for an Authentication exchange.
- wait represents the generic state indicating that a trading role is waiting for an IOTP message from another trading role.
- proAuthReq indicates that the trading role is in the state of processing an *Authentication Request Message*. Here, pro is a prefix standing for message processing, while AuthReq stands for the corresponding IOTP message. In this way, the various message processing states have been defined.

5.2 IOTP Payment Exchange

The Payment exchange allows a payment using some payment brand to be made by the Consumer to the Payment Handler. The RFC [3] defines a set of message processing guidelines for each exchange. As shown in Fig. 1, the messages exchanged in a successful Payment exchange consist of:

- A *Payment Request Message* sent from the Consumer to the Payment Handler to start the payment.
- A set of *Payment Exchange Messages* exchanged between the Consumer and the Payment Handler to process the payment.
- A *Payment Response Message* sent to the Consumer from the Payment Handler to complete the payment.

```
1   color State = with ready | wait | cancel | stop |
          A_ready | proAuthReq | proAuthRsp | A_cancel | A_stop | A_fail |
          O_ready | proTPO | proTPOSel | O_cancel |O_stop |
          P_ready | proPayReq | proPayExch | P_cancel |P_stop | PD_stop |
          D_ready | proDelivReq | D_cancel |D_stop;

2   color ConState = subset State with [ready, cancel, wait, stop,
                        A_ready, proAuthReq, A_cancel, A_stop, A_fail,
                        O_ready, proTPO, O_cancel, O_stop,
                        P_ready, proPayExch, P_cancel, P_stop, PD_stop,
                        D_ready, D_cancel, D_stop];

3   color MerState = subset State with [ready, cancel, wait, stop,
                        A_ready, proAuthRsp, A_cancel, A_stop, A_fail,
                        O_ready, proTPOSel, O_cancel, O_stop];

4   color PHState = subset State with [ready, cancel, wait, stop,
                        P_ready, proPayReq, proPayExch, P_cancel, P_stop,
                        PD_stop];

5   color DHState = subset State with [ready, cancel, wait, stop,
                        D_ready, proDelivReq, D_cancel, D_stop];

6   color CSxTR = product ConState * TradingRole;
7   color MSxTR = product MerState * TradingRole;
8   var sc: ConState;        var sm: MerState;        var sph: PHState;
```

Fig. 9: Colour sets and variables for modelling states of trading roles.

During a Payment exchange, both the Consumer and the Payment Handler may cancel the payment by sending a *Cancel Message* to the other side. As a result, the payment should be cancelled at both sides resulting in a cancelled Payment exchange. The page specifying the Payment exchange has been developed according to the above message processing guidelines. It captures not only the successful Payment exchange, but all the possible executions of it.

Figure 10 depicts the page C_Pay (of Fig. 2) specifying the Consumer side of a Payment exchange. Figure 11 depicts the page PH_Pay specifying the Payment Handler side of a Payment exchange. In these two pages, each transition represents the event of sending or receiving a message. Then, the message to be transmitted contains not only the message content, but also the receiver trading role, while the message received contains the content together with the sender trading role.

We now illustrate how the pages shown in Fig. 10 and Fig. 11 reflect the Payment exchange. Both trading roles are initially in state P_ready. The payment starts when the Consumer sends [PayReq] to PHandler. On receiving [PayReq] from the Consumer, the Payment Handler checks the message and thus is in the state of proPayReq. If the result is valid, [PayExch] is sent to the Consumer. Otherwise, the Payment Handler will send a [Cancel] to cancel the payment. In Fig. 11 this is indicated by transition SendPayExch and SendPayCnl. In Fig. 10, identically named transitions are also used to indicate the same events are carried out by the Consumer

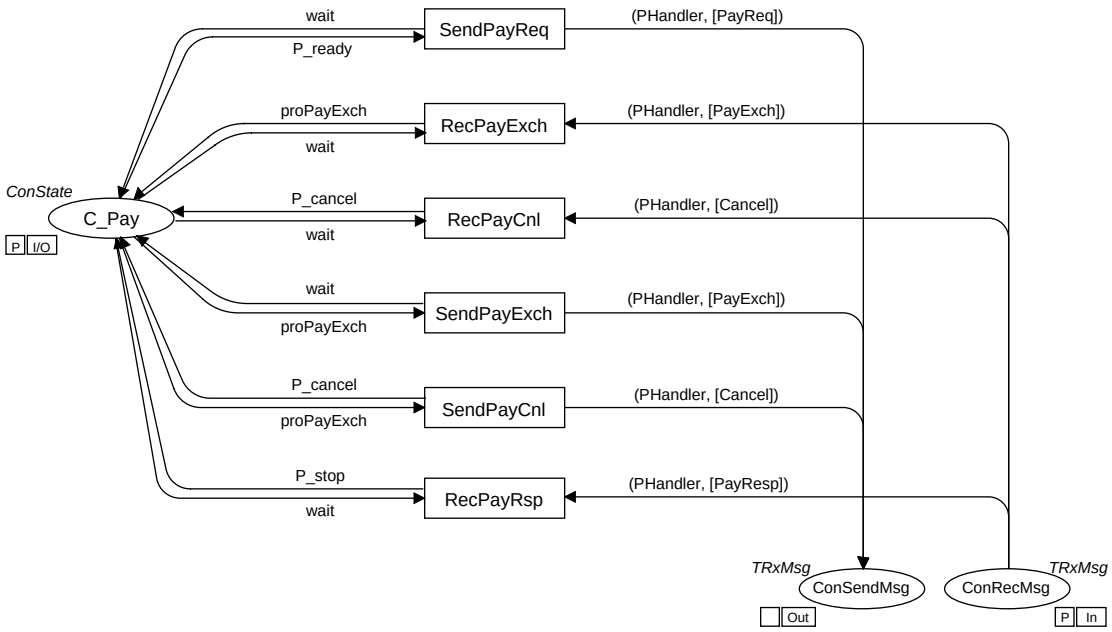


Fig. 10: The Payment Exchange – Consumer.

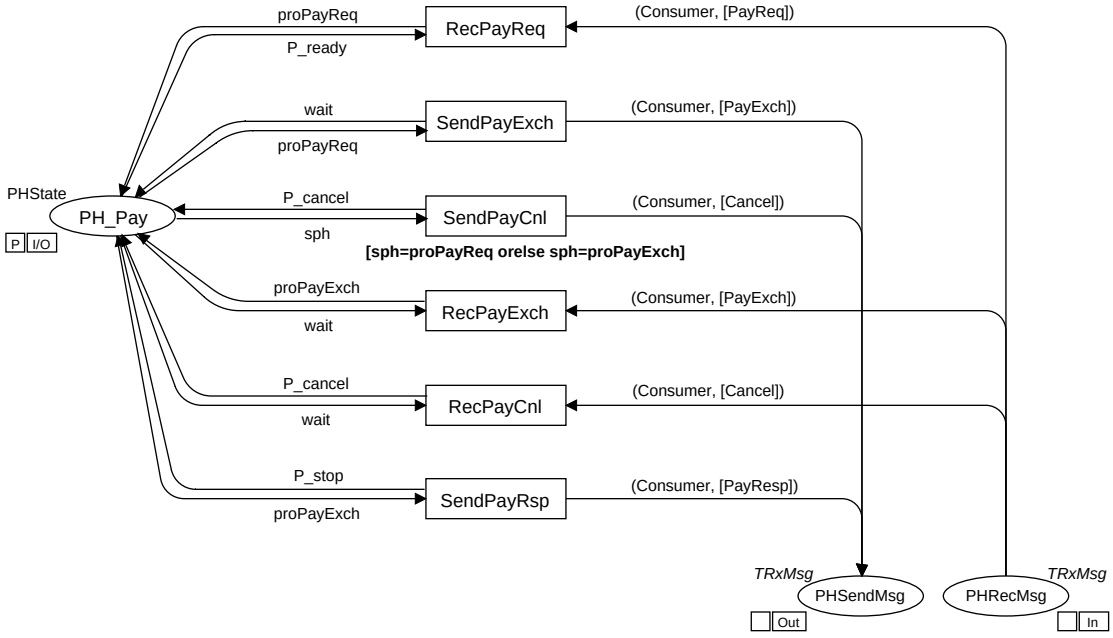


Fig. 11: The Payment Exchange – Payment Handler.

after receiving [PayExch], while the occurrence of RecPayCnl results in the cancellation of payment on the Consumer side after [cancel] is received. Then, on receiving the [PayExch], the Payment Handler may either send [PayResp] to complete the payment, or send [PayCnl] to cancel the payment. A successful Payment exchange will be completed on both sides when the Consumer receives [PayResp].

6 Validation of CPN IOTP Model

In this section, we apply simulation and state space analysis to validate the CPN model of the Purchase transaction. The simulation results are presented using *Message Sequence Charts* (MSCs) [1] as implemented in the MSC library of Design/CPN [12]. State space analysis has been conducted using the state space report produced by Design/CPN, and standard query functions. Simulation and state space analysis similar to the one presented for the Purchase transaction in this section has been conducted for the CPN models of the other IOTP transactions.

6.1 Simulation Analysis

The aim of the simulation analysis is to validate that the constructed CPN model of the Purchase transaction conforms with the specification of the Purchase transaction in the RFC. We do so by demonstrating that the CPN model of the Purchase transaction can generate the same set of message exchanges between trading roles as specified for the Purchase transaction in the RFC. The RFC specifies the message exchanges between trading roles in a Purchase transaction using MSCs. To make it easy to compare the message exchanges between trading roles as specified by the CPN model and as specified by the RFC, the CPN model has been instrumented so that MSCs displaying the message exchanges between trading roles can be automatically produced during a simulation. Figures 12-14 show three representative MSCs resulting from simulation of the CPN model. Each of them shows the message exchanges between the trading role entities (i.e., Consumer, Merchant, Payment Handler, and Delivery Handler) during a Purchase transaction. Time increases from the top of the chart to the bottom.

Figure 12 shows a successful completed Purchase transaction. The first three events are concerned with an Authentication exchange (labelled as event 1-3 in the figure). After that, a Brand Dependent Offer exchange is represented by the following 3 events (event 4-6). Next, a Payment exchange is carried out (event 7-10). Finally, the last two events shows the occurrence of a Delivery exchange (event 11-12). Therefore, Fig. 12 also provides the MSCs for these four IOTP exchanges, and each of them matches the corresponding MSCs from the RFC as shown in Fig. 1. However, Fig. 12 differs to Fig. 1 in two points. First, Fig. 12 contains the MSC for an initial Authentication exchange between Consumer and Merchant as part of a Purchase transaction, while there is no Authentication considered in Fig. 1. Second, messages outside the scope of IOTP are shown in Fig. 1 but become invisible in Fig. 12.

The specification of IOTP transactions in the RFC contains only the MSCs for those IOTP exchanges that are completed successfully without any cancellation during their execution. In contrast, the CPN model of the Purchase transaction specifies all the possible executions of a Purchase transaction. Figure 13 shows examples of a cancelled and a failed authentication exchange, and Figure 14 shows that the Payment exchange can be cancelled at two different stages during the transactions. These unsuccessful exchanges then result in the cancellation of the transaction they are part of.

6.2 State Space Analysis

The aim of the state space analysis is to investigate the behaviour of the CPN model as well as to validate and verify the functional correctness of the Purchase transaction. A full

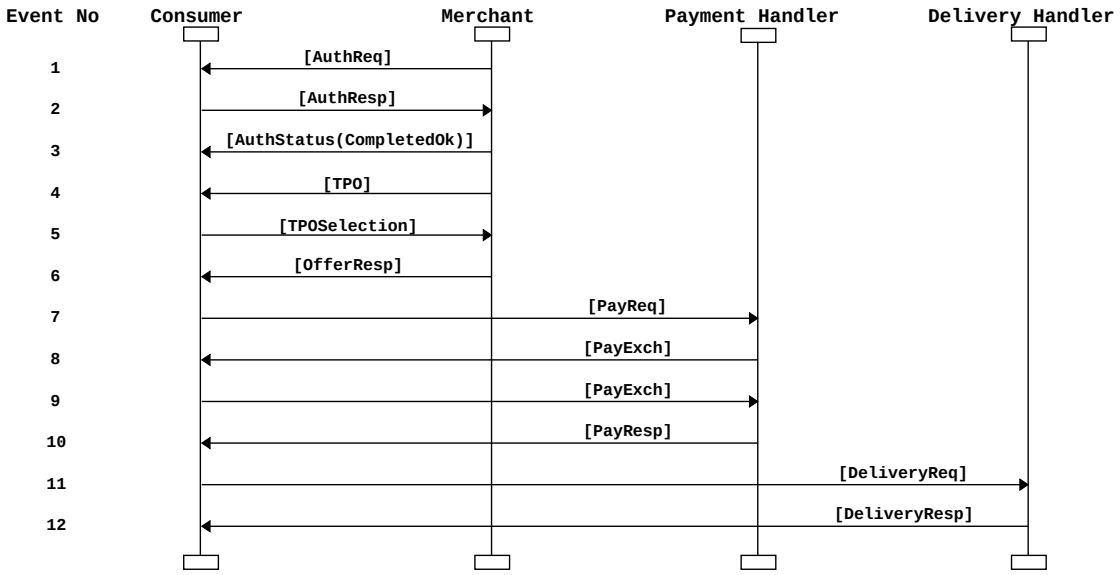


Fig. 12: The MSC for a successful Purchase transaction.

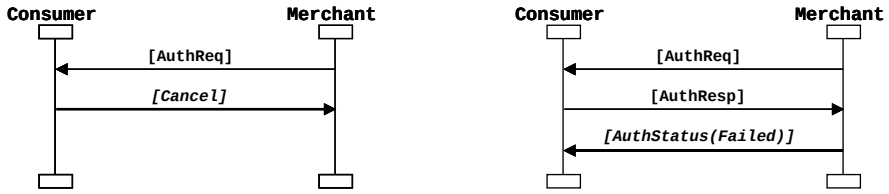


Fig. 13: The MSC for cancelled Authentication exchanges.

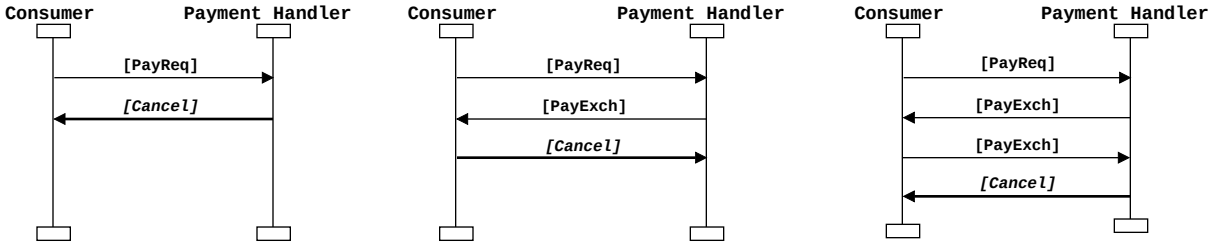


Fig. 14: The MSC for cancelled Payment exchanges.

state space report for the Purchase transaction model has been generated. The statistical information shows that the state space (OCC) has 244 nodes and 490 arcs.

Table 1 shows the home and the liveness properties and has been extracted from the state space report. There are five dead markings representing the terminal states of a Purchase transaction. Table 2 lists the corresponding states of each trading role in the five dead markings. Marking 164 and 243 represent the two possible successful executions of a Purchase transaction. In marking 164, the Purchase transaction is completed at the end of a Payment or a Payment with Delivery exchange between the Consumer and the Payment Handler. The

Delivery Handler is therefore never active and is in state ready at the end of the transaction. In marking 243, the purchase is completed with a Delivery exchange after the payment. The cancelled Purchase transactions are represented by the other three dead markings. Marking 57 corresponds to the state in which the transaction is cancelled during the Authentication exchange or one of the two Offer exchanges between the Consumer and the Merchant. The transaction can also be cancelled during the Payment exchange or the Payment with Delivery exchange, as indicated by marking 126. Finally, marking 244 represents the state where the transaction is cancelled during the Delivery exchange. Carefull inspection of Table 2 also shows that the states which the trading roles are in upon termination of the transaction are consistent. Therefore, investigating all the dead markings shows that the Purchase transaction terminates properly. The set of dead markings also constitutes a *home space*, i.e. it is always possible during the execution of the Purchase transaction to reach one of the dead markings. This has been checked using the query functions HomeSpace and ListDeadMarkings of the state space tool.

Table 1: Home and liveness properties.

Home Markings	None
Dead Markings	[57,126,164,243,244]
Dead Transitions Instances	None
Live Transitions Instances	None

Table 2: Trading role states in the dead markings.

Trading role	Dead marking				
	[57]	[126]	[164]	[243]	[244]
Consumer	1‘cancel	1‘cancel	1‘stop	1‘stop	1‘cancel
Merchant	1‘cancel	1‘stop	1‘stop	1‘stop	1‘stop
Payment Handler	1‘ready	1‘cancel	1‘stop	1‘stop	1‘stop
Delivery Handler	1‘ready	1‘ready	1‘ready	1‘stop	1‘cancel

Table 3 lists upper and lower integer bounds of the places modelling the message buffers (see Fig. 3). The boundedness results show that the minimal number of messages in each of the buffers is zero, which corresponds to the terminal states of the Purchase transaction. The maximum number of messages is 1 except that both the Consumer receiving buffer (ConRecMsg) and the Merchant sending buffer (MerSendMsg) may contain 2 at a time. The corresponding markings can be identified using the PredAllNodes query function of the state space tool, and an execution of the transaction leading to such marking can be found using the NodesInPath function. To understand why these two buffers may contain two messages at the same time consider the MSC in Fig. 12. The events 3 and 4 show that the Merchant sends the TPO message ([TPO]) directly after the successful Authentication status message ([AuthStatus(ComletedOk)]) with no confirmation message from the Consumer in between. The Consumer may receive the second message before processing the first one. Inspection of the multi-set bounds in the state space report shows that the trading roles do enter their expected states during the transaction, and that the messages exchanged between them are as expected.

Table 3: Upper and lower integer bounds.

Place	Upper	Lower	Place	Upper	Lower
ConRecMsg	2	0	MerRecMsg	1	0
ConSendMsg	1	0	MerSendMsg	2	0
DHRecMsg	1	0	PHRecMsg	1	0
DHSendMsg	1	0	PHSendMsg	1	0

7 IOTP Protocol Architecture

The RFC for IOTP does not contain a precise and detailed specification of the protocol architecture and the internal organisation of the individual layers. In this section we present our initial proposal for a protocol architecture of IOTP. The proposed protocol architecture is derived from the constructed CPN models of the trading transactions.

IOTP is clearly an application protocol in the context of the OSI reference model [2] possibly operating across different transport protocol services for transmission of IOTP messages. This is also evident from the prime page of the CPN model shown in Fig. 3. From the hierarchy page of the CPN model shown in Fig. 2, IOTP itself can be seen as consisting of two layers: a *transaction layer* and a *exchange layer*, where the transaction layer uses the services provided by the exchange layer to implement the trading transactions.

IOTP protocol entities are not designed to work solely as stand alone applications. An IOTP protocol entity will typically be embedded in other applications such as HTTP clients and HTTP servers when, e.g., the Consumer trading role entity is implemented in a HTTP web browser and the Merchant trading role entity is implemented in a HTTP web server for trading across the Internet. Moreover, IOTP will in most cases operate across the same transport service as the application it is part of. This transport service will typically vary depending on the application and the underlying communication medium. This suggests the existence of a *transport adaption layer*. This layer makes it possible to adapt the transport service of the specific application to the service required by IOTP.

Figure 15 shows the proposed protocol architecture of IOTP with the four main protocol layers as identified above. In the following we discuss the service provided by individual layers as well as their internal organisation in more detail.

Transactions Layer. This layer implements the IOTP trading and infrastructure transactions. For baseline IOTP this layer implements *Deposit*, *Withdrawal*, *Purchase*, *Refund*, *Value Exchange*, *Inquiry*, and *Ping* transactions.

Exchange Layer. This layer implements the IOTP exchanges. For baseline IOTP this layer implements *Authentication*, *Brand Dependent Offer*, *Brand Independent Offer*, *Payment*, *Delivery*, and *Payment with Delivery* exchanges. The CPN model presented in this paper specifies in detail how the individual trading transactions can be implemented as combinations of these exchanges.

Payment Sublayer. IOTP can support different payment instruments (such as VISA, MasterCard, and Mondex Card) by encapsulating the underlying payment protocols (such as

SET, Mondex VTP) during the payment-related exchanges. The payment sublayer has been defined as a sublayer embedded in the exchange layer since the *payment scheme component* defined in IOTP will be used to encapsulate the payment protocol data, e.g., a SET message, during a Payment exchange. One component of the payment sublayer is the so-called IOTP *payment bridges* specified in the Internet Draft for IOTP Payment API [6]. These payment bridges specify the interface and interaction between IOTP exchanges and the payment system.

Application Transport Adaption Layer. This layer interfaces IOTP to the underlying transport layer and transport service. This layer will ensure that the IOTP messages, which are well-formed XML documents, are carried successfully between the trading role entities. Typically, the application transport adaptation layer includes the mechanisms which support the mapping of the IOTP data format to the underlying transport layer, e.g., an XML to HTTP mapping.

Application Transport Layer. This is the basic transport service on top of which the IOTP entities are operating. This layer could be TCP/IP as well as other higher level transport protocols such as HTTP and SMTP. It might also be the Wireless Application Protocol (WAP) [5] transport service if IOTP is used in wireless e-commerce such as mobile commerce (m-commerce) applications. This layer can also provide session layer services such as those provided by Secure Socket Layer (SSL) for encryption.

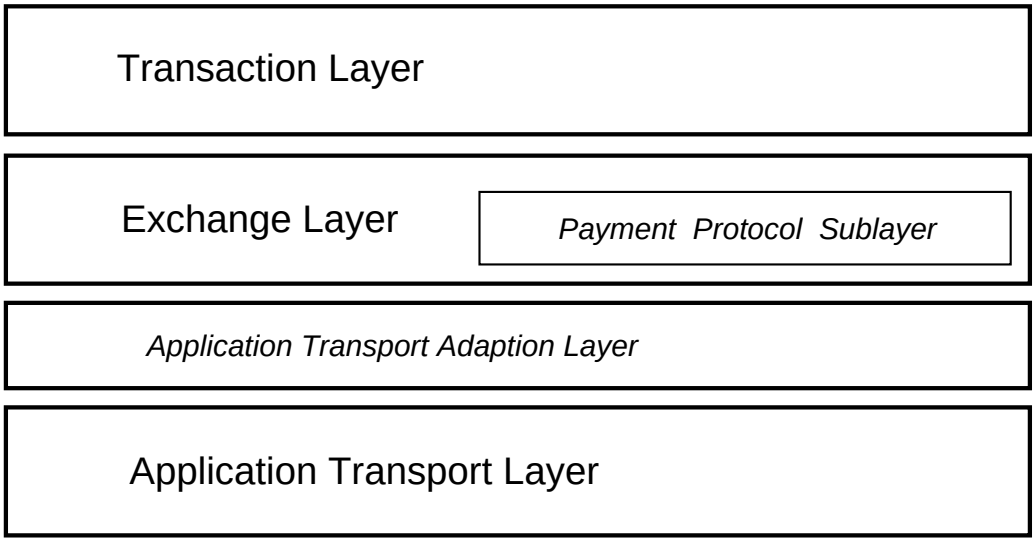


Fig. 15: The layered IOTP architecture.

The protocol architecture above has only one concept of *IOTP exchanges* – constituting the exchange layer. This contrast with the RFC which operators with two notions of exchanges: *document Exchanges* and *trading exchanges*. The CPN models of the IOTP transactions and the validation presented in Sect. 6 show that the concept of *document exchanges* suffices, since it can implement all the IOTP trading transactions. As a simplification of the IOTP specification we therefore propose to eliminate the concept of *trading exchanges* since it is not required from a protocol perspective.

8 Conclusions

We have presented a hierarchical CPN model of IOTP based on RFC 2801. The validation results demonstrate that our CPN IOTP model conforms to the specification given in the RFC. We have also proposed a complete and simplified architecture for IOTP based on the constructed CPN models. In addition, we propose the merging of the two unclarified notions of *trading exchanges* and *document exchanges* into one concept which we call *IOTP exchanges* leading to a simpler specification and architecture for IOTP.

For the current modelling of the IOTP transactions, we have a number of simplifying assumptions and abstractions. These simplifications are primarily related to IOTP error handling. For example, we have only investigated IOTP over a perfect transport medium without any message loss. Besides the error handling, the arbitrary cancellation during a transaction has not yet been taken into consideration. As part of future work we plan to include more internal operations in each layer such as the error handling and the arbitrary cancellation in our CPN model. However, the assumptions and simplifications will not affect the proposed protocol architecture.

Investigating the IOTP service specification will be another challenging part of future work, since there are no concepts of IOTP service defined in the RFC. The IOTP architecture proposed in this paper not only leads to a protocol specification, but also presents a first step towards developing a service specification for IOTP.

References

1. *ITU Recommendation Z.120, Message Sequence Chart*, 1992.
2. *ITU-T Recommendation X.200, Information Technology - Open Systems Interaction - Basic Reference Model*, July 1994.
3. D. Burdett. *Internet Open Trading Protocol - IOTP. RFC 2801.* IETF Trade Working Group, April 2000. Version 1.0.
4. D. Burdett, D. Eastlake, and M. Goncalves. *Internet Open Trading Protocol.* McGraw-Hill, 2000.
5. WAP Forum. Wireless Application Protocol Architecture Specification. Available via `http://www.wapforum.org/`, 30 April 1998.
6. W. Hans, Y. Kawatsura, and M. Hiroya. *Payment API for v1.0 Internet Open Trading Protocol (IOTP).* IETF Trade Working Group, April 2001.
7. The Internet Engineering Task Force - IETF. `http://www.ietf.org/`.
8. Hitachi Research Topics IOTP. `http://www.hitachi.co.jp/english/topics/t_iotp/iotp.html`.
9. JOTP Open Trading Protocol Toolkit For Java. `http://www.livebiz.com/`.
10. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume-1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
11. 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. Springer-Verlag.
12. Design/CPN Message Sequence Chart Library. Department of Computer Science, University of Aarhus, Denmark., 1998. Version 1.1. Available via `http://www.daimi.au.dk/designCPN/`.
13. Design/CPN online. Available via `http://www.daimi.au.dk/designCPN/`.
14. Hitachi Smiles. `http://www.hitachi.co.jp/Div/nfs/whats_new/smiles.html`.
15. Mondex USA. `http://www.mondexusa.com/html/content/secur/security.htm`.
16. Visa and MasterCard. *SET Secure Electronic Transaction Specification. Volume-1,2,3*, May 1997. Version 1.0.
17. V. Zwass. Structure and Macro-Level Impacts of Electronic Commerce: From Technological Infrastructure to Electronic Marketplaces. In *Foundations of Information Systems.* McGraw-Hill, 1998.