ISSN 0105-8517

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

Aarhus, Denmark, October 24-26, 2006

Kurt Jensen (Ed.)

DAIMI PB - 579
October 2006

Preface

This booklet contains the proceedings of the Seventh Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, October 24-26, 2006. 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/workshop06/

Coloured Petri Nets and the CPN Tools are now used by more than 4000 users in 124 countries. The aim of the workshop is to bring together some of the users and in this way provide a forum for those who are interested in the practical use of Coloured Petri Nets and their tools.

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

Wil van der Aalst, Netherlands
João Paulo Barros, Protugal
Jonathan Billington, Australia
Jörg Desel, Germany
Joao M. Fernandes, Portugal
Jorge de Figueiredo, Brazil
Monika Heiner, Germany
Kurt Jensen, Denmark (chair)
Ekkart Kindler, Germany
Lars M. Kristensen, Denmark
Johan Lilius, Finland
Tadao Murata, USA
Daniel Moldt, Germany
Laure Petrucci, France
Robert Valette, France
Rüdiger Valk, Germany
Lee Wagenhals, USA
Jianli Xu, Finland
Karsten Wolf, Germany

The programme committee has accepted 15 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 six CPN Workshops can be found via the web pages: http://www.daimi.au.dk/CPnets/. After an additional round of reviewing and revision, some of the papers have also been published as special sections 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

Lars M. Kristensen, Peter Mechlenborg, Lin Zhang, Brice Mitchell, and Guy E. Gallasch
Model-based Development of a Course of Action Scheduling Tool 1

Jens Bæk Jørgensen, Kristian Bisgaard Lassen, and Wil M. P. van der Aalst
From Task Descriptions via Coloured Petri Nets Towards an Implementation of a New Electronic Patient Record 17

Cong Yuan and Jonathan Billington
A Coloured Petri Net Model of the Dynamic MANET On-demand Routing Protocol 37

A. Rozinat, R.S. Mans, and W.M.P. van der Aalst
Mining CPN Models
Discovering Process Models with Data from Event Logs 57

M.H. Jansen-Vullers and M. Netjes
Business Process Simulation - A Tool Survey 77

Michael Westergaard
The BRITNeY Suite: A Platform for Experiments 97

Guy Edward Gallasch, Nimrod Lilith, Jonathan Billington, Lin Zhang, Axel Bender, and Benjamin Francis
Modelling Defence Logistics Network*s* 117

F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and H.M.W. Verbeek
Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes 137

Somsak Vanit-Anunchai, Jonathan Billington and Guy Edward Gallasch
Sweep-line Analysis of DCCP Connection Management 157

Hendrik Oberheid
A Colored Petri Net Model of Cooperative Arrival Planning in Air Traffic Control 177

Vijay Gehlot and Anush Hayrapetyan
A CPN Model of a SIP-Based Dynamic Discovery Protocol for Webservices in a Mobile Environment 197

P.M. Kwantes
Design of Clearing and settlement operations:
A case study in business process modelling and evaluation with Petri nets 217

Óscar R. Ribeiro and João M. Fernandes
Some Rules to Transform Sequence Diagrams into Coloured Petri Nets 237

Invited Talk:
Karsten Wolf
Inside LoLA - Experiences from Building a State Space Tool for Place Transition Nets 257

Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto
Using Coloured Petri Nets to Simulate DoS-resistant protocols 261

M. Westergaard
Game Coloured Petri Nets 281

Model-based Development of a Course of Action Scheduling Tool

Lars M. Kristensen[1], Peter Mechlenborg[1],
Lin Zhang[2], Brice Mitchell[2], and Guy E. Gallasch[3]

[1] Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, DENMARK,
{lmkristensen,metch}@daimi.au.dk
[2] Defence Science and Technology Organisation
PO Box 1500, Edinburgh, SA 5111, AUSTRALIA
{Lin.Zhang, Brice.Mitchell}@dsto.defence.gov.au
[3] Computer Systems Engineering Centre, University of South Australia,
Mawson Lakes Campus, SA 5095, AUSTRALIA
guy.gallasch@postgrads.unisa.edu.au

Abstract. This paper shows how a formal method in the form of Coloured Petri Nets (CPNs) and the supporting CPN Tools have been used in the development of the Course of Action Scheduling Tool (COAST). The aim of COAST is to support human planners in the specification and scheduling of tasks in a Course of Action. CPNs have been used to develop a formal model of the task execution framework underlying COAST. The CPN model has been extracted in executable form from CPN Tools and embedded directly into COAST, thereby automatically bridging the gap between the formal specification and its implementation. The scheduling capabilities of COAST are based on state space exploration of the embedded CPN model. Planners interact with COAST using a domain-specific graphical user interface (GUI) that hides the embedded CPN model and analysis algorithms. This means that COAST is based on a rigorous semantical model, but the use of formal methods is transparent to the users. Trials of operational planning using COAST have been conducted within the Australian Defence Force.

Keywords: Application of Coloured Petri nets, State space analysis, Scheduling, Command and Control, Methodologies, Tools.

1 Introduction

Planning and scheduling [6] are activities performed in many domains such as building construction, natural disaster relief operations, search and rescue missions, and military operations. Planning is a major challenge due to several factors, including time pressure, ambiguity in guidance, uncertainty, and complexity of the problem. The development of computer tools that can aid planners in developing and analysing Courses of Actions such that they meet their objectives is therefore of key interest in many application domains. Recently, there has been an increased interest in the application of formal methods and associated analysis techniques in the planning and scheduling domain (see, e.g., [1, 3, 11]).

This paper presents the Course of Action Scheduling Tool (COAST) being developed for the Australian Defence Force (ADF) by the Australian Defence Science and Technology Organisation (DSTO). The aim of COAST is to support planners in *Course of Action* (COA) development and analysis which are two of the main activities in planning processes within the ADF. The framework underlying COAST has been deliberately made generic in order to make COAST applicable to a broad spectrum of domains, and so not restricting its applicability to the military planning domain. The basic entities in a COA (representing a planning problem) are *tasks* which have associated pre- and postconditions describing the *conditions* required for a task to execute and the *effect* of executing the task. The dependencies between tasks expressed via conditions capture the logical structure of a COA. The execution of a task also requires *resources*; a subset of which are released in accordance with the specifications when the task terminates. Task *synchronisations* make it possible to directly specify temporal and precedence information for tasks, e.g., a set of tasks must start simultaneously. The main analysis capability of COAST is the computation of task schedules called *lines of operation* (LOPs) which are execution sequences of tasks that lead from an initial state to a desired *goal state* which is a state in which a certain set of conditions are satisfied. COAST also supports the planners in debugging and identifying errors in COAs.

The development of COAST has been driven by the use of formal methods in the form of *Coloured Petri Nets* (CP-nets or CPNs) [7, 8] and the supporting computer tool CPN Tools [2]. CPN modelling was chosen because CP-nets support construction of compact parametriseable models, support structured data types, make it possible to model time, and allow models to be hierarchically structured into a set of modules.

The basic idea behind the development of COAST has been to use CP-nets to develop, formalise, and implement the task execution framework which forms the core of COAST. The CPN model formalises the execution of tasks according to the pre- and postconditions of tasks, imposed synchronisations, and assigned resources. The concrete tasks, conditions, synchronisations, and resources that make up the COA to be analysed are represented as *tokens* populating the CPN model. The analysis capabilities of COAST are based on *state space exploration* [14]. State space exploration relies on computing all reachable states and state changes of the system and representing these as a directed graph where nodes represent reachable states and arcs represent occurring events. Two algorithms are implemented for computing lines of operation: a two-phase algorithm consisting of a depth-first state space generation followed by a breadth-first traversal, and an algorithm that is based on the so-called sweep-line method [10]. The CPN model and the analysis algorithms that form the core of COAST have been hidden behind a domain-specific graphical user interface. This makes the use of the underlying formal method transparent to the planners who cannot be expected to be familiar with CP-nets and state space methods.

A preliminary version of the task execution framework of COAST has been informally presented in [15, 16] together with the graphical user interface. Some early algorithms for computing lines of operation were presented in [9]. The contribution of this paper is to present the formal engineering aspects of COAST in the form of the underlying CPN model and the new state space exploration algorithms implemented for obtaining lines of operation. We also demonstrate how the sweep-line method [10] can be applied to the planning domain. Furthermore, we explain how COAST has been engineered via embedding of an executable CPN model. The latter demonstrates how formal methods in the form of CP-nets can be used in software development.

The rest of the paper is organised as follows. Section 2 presents the methodology used for the formal engineering of COAST. Section 3 presents selected parts of the CPN model that formalises the task execution framework. Section 4 presents the state space exploration algorithms for generating lines of operation. Finally, we sum up the conclusions in Sect. 5. The reader is assumed to be familiar with the CPN modelling language and the basic ideas behind state space methods.

2 Formal Engineering Methodology

COAST is based on a client-server architecture. The client constitutes the domain-specific graphical user interface and is used for the specification of COAs. It supports the human planners in specifying tasks, resources, conditions, and synchronisations. When the COA is to be developed and analysed this information is sent to the COAST server. The client can now invoke the analysis algorithms in the server to compute lines of operation. The server also supports the client in exploring and debugging the COA in case the analysis shows that no lines of operation exist. Communication between the client and the server is based on a remote procedure call (RPC) mechanism implemented using the Comms/CPN library [5].

This paper concentrates on the development of the COAST server which constitutes the computational back-end of COAST. The development of the server has followed a model-based engineering process [12]. Figure 1 depicts the engineering process of developing the application that constitutes the server. The first step was to develop and formalise the planning domain that provides the semantical foundation of COAST. This was done by constructing a CPN model using CPN Tools that formally captures the semantics of tasks, conditions, resources, and synchronisations. This activity involved discussions with the prospective users of COAST (i.e., the planners) to identify requirements and determine the concepts and working processes that were to be supported. The second step was then to extract the constructed CPN model from CPN Tools. This was done by saving a *simulation image* from CPN Tools. This simulation image contains the Standard ML (SML) [13] code that CPN Tools generates for simulation of the CPN model. An important property of the CPN model is that it has been parameterised with respect to the set of tasks, conditions, resources, and synchronisations. This ensures that a given COA can be analysed by setting the initial state of the CPN model accordingly, i.e., no changes to the structure of the CPN model is required to analyse a different COA. This means that the simulation image extracted from CPN Tools is able to simulate any COA, and CPN Tools is no longer needed once the simulation image has been extracted. The third step was the implementation of a suitable interface to the extracted CPN model and the implementation of the state space exploration algorithms.

The Model Interface module contains two sets of primitives:

Initialisation primitives that make it possible to set the initial state of the CPN model according to a concrete set of tasks, conditions, resources, and synchronisation that constitute the COA to be analysed.

Transition relation primitives that provide access to the transition relation determined by the CPN model. This set of primitives make it possible to obtain the set of events enabled in a given state, and the state reached when an enabled event occurs in a given state.

The transition relation primitives are used to implement the state space exploration algorithms in the Analysis module for computing lines of operation. The state space exploration algorithms will be presented in Sect. 4. The Comms/CPN module was added implementing a remote procedure call mechanism allowing the client to invoke the primitives in the analysis and the initialisation module. The resulting application constitutes the COAST server.

3 The COAST CPN Model

The conceptual framework underlying COAST is based on the notion of tasks, conditions, synchronisations, and resources representing the entities of the planning domain. The complete COAST CPN model is hierarchically structured into 24 modules. As the CPN model is too large to be presented in full in this paper, we provide an overview of the CPN model, and illustrate how the key concepts in the planning domain have been modelled and represented using CP-nets.

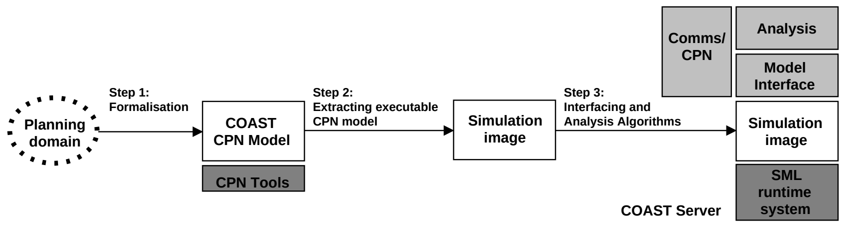


Fig. 1. Engineering process for the COAST server.

3.1 Modelling of COA Entities

A concrete COA to be analysed consists of a set of tasks (T), conditions (C), synchronisations (S), and a multi-set[1] of resources (R). These entities are represented as *tokens* in the CPN model based on the *colour set* definitions listed in Figure 2. Figure 2 lists the definitions of the colour sets that represent the key entities of a COA. Not all colour set definitions are given as the CPN model contains 53 colour sets in total.

```
colset Task = record
    name           : STRING       * duration     : Duration    *
    normalprecond  : SConditions  * vanprecond   : SConditions *
    sustainprecond : SConditions  * termprecond  : SConditions *
    instanteffect  : SConditions  * posteffect   : SConditions *
    sustaineffect  : SConditions  *
    startresources : ResourceList * resourceloss : ResourceList;

colset Resource = product INT * STRING;
colset ResourceList = list Resource;
colset ResourcexAvailability = product Resource * Availability;
colset ResourceSpecs = list ResourcexAvailability;
colset Resources = union IDLE : ResourceSpecs + LOST : ResourceSpecs;

colset STRINGxBOOL = product STRING * BOOL;
colset SCondition = STRINGxBOOL;
colset SConditions = list SCondition;
colset Condition = union STRINGxBOOL;
colset Conditions = list Condition;

colset BeginSynchronisation = list Task;
colset EndSynchronisation = list Task;
```

Fig. 2. Colour set definitions for representing COA entities.

Tasks are the executable entities in a COA and are modelled by colours (data values) of the colour set (type) `Task` which is defined as a record consisting of 11 fields. The `name` field is used to specify the name of the task and the `duration` field to specify the minimal duration of the task. The duration of a task may be extended due to synchronisations, and not all tasks are required to have a specified minimal duration since their durations may be implicitly given by synchronisations and conditions. The remaining fields can be divided into:

[1] A multi-set (bag) is required since there may be several resources of the same type.

Preconditions that specify the conditions that must be valid for starting the task. The colour set `SConditions` is used for modelling the condition attributes of tasks. A task has a set of *normal preconditions* (represented by field `normalprecond`) that specify the conditions that must be satisfied for the task to start. A subset of the normal preconditions may be further specified as *vanishing preconditions* to represent the effect that the start of the task will invalidate such preconditions. The *sustaining preconditions* (field `sustainprecond`) specify the set of conditions that must be satisfied for the entire duration of execution of the task. If a sustaining precondition becomes invalid, then it will cause the task to *abort* which may in turn cause other tasks to be *interrupted*. The *termination preconditions* (field `termprecond`) specify the conditions that must be satisfied for the task to terminate.

Effects that specify the effects of starting and executing a task. The *instant effects* (field `instanteffect`) are conditions that become immediately valid when the task starts executing. The *post effects* (field `posteffect`) are conditions that become valid at the moment the task terminates. *Sustained effects* (field `sustaineffect`) are conditions that are valid as long as the task is executing.

Resources that specify the resources required by the tasks during their execution. Resources typically represent planes, ships, and personnel required to execute a task. Resources may be lost or consumed in the course of executing a task. *Start resources* specify the resources required to start the task, and they are allocated as long as the task is executing. The *resource loss* field specifies resources that may be lost when executing the task. Each type of resources is modelled by the colour set `Resource` which is a product of an integer (`INT`) specifying the quantity and a string (`STRING`) specifying the resource name. The colour set `ResourceList` is used for specifying the resource attributes of a task.

Conditions are used to describe the explicit logical dependencies between tasks via preconditions and effects. As an example, a task `T1` may have an effect used as a precondition of a task `T2`. Hence, `T2` logically depends on `T1` in the sense that it cannot be started until `T1` has been executed.

The colour set `Conditions` is used for representing the value of the conditions in the course of executing tasks in the COA. A condition is a pair consisting of a `STRING` specifying the name of the condition and a boolean (`BOOL`) specifying the truth value. The colour set `ResourceSpecs` is used to represent the state of resources assigned to the COA. The colour set `Resources` is defined as a `union` for modelling the idle and lost resources. The assigned resources also have a specification of the *availability* of the resources (via the `Availability` colour set) specifying the time intervals at which the resource is available. Resources may be available only at certain times due to e.g., service intervals.

Synchronisations are used to directly specify precedence and temporal constraints between tasks. For example, a set of tasks must begin or end simultaneously, a specific amount of time must elapse between the start and end of certain tasks, or a task can only start after a certain point in time. Tasks that are required to begin at the same time are said to be *begin-synchronised*, and tasks required to end at the same time are said to be *end-synchronised*. End-synchronisations can cause the duration of a task to be extended. The colour sets `BeginSynchronisation` and `EndSynchronisation` represent that a set (list) of tasks are begin and end-synchronised, respectively.

3.2 Modelling Task Behaviour

Figure 3 shows the top level module of the CPN model which is composed of three main parts represented by the three substitution transitions Initialise, Execute, and Environment. The substitution transition Initialise and its submodules are used for the initialisation of the model according to the concrete set of tasks, conditions, synchronisations, and resources in the COA to be analysed. The substitution transition Execute and its submodules model the execution of tasks, i.e., start, termination, abortion, and interruption of tasks. The substitution transition Environment and its submodules model the environment in which tasks execute, and are responsible for managing the availability of resources over time and the change of conditions over time. The text in the small rectangular box attached to each substitution transition gives the name of the associated submodule.

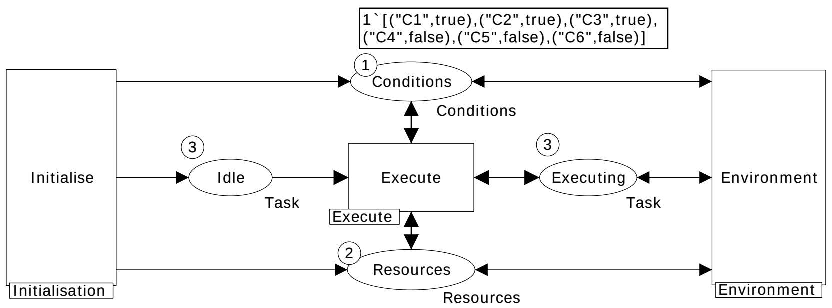


Fig. 3. Top level module of the CPN model.

There are four places in Fig. 3. The Resources place models the state of the resources, the Idle place models the tasks that are yet to be executed, the Executing place models the tasks currently being executed, and the Conditions place models the values of the conditions. The state of a CPN model is a distribution of tokens on the places of the CPN model. Figure 3 depicts a simple example state for a COA with six tasks. The number of tokens on a place is written in the small circle positioned above the place. The detailed data values of the tokens are given in the box positioned next to the circle. Place Conditions contains one token that is a list containing the conditions in the COA and their truth values. Place Resources contains two tokens. There is one token consisting of a list describing the current set of idle (available) resources, and the other token consisting of a list describing the resources that have been lost until now. Since the colours of the tokens on the places Resources, Executing and Idle are of a complex colour set, we have only shown the numbers of tokens and not the data values.

Figure 4 shows one of the submodules of the `Execute` substitution transition (see Fig. 3). This submodule represents one of the steps in starting tasks. The transition Start models the event of starting a set of begin-synchronised tasks. The two places Resources and Conditions are interface places linked to the accordingly named places of the top-level module shown in Fig. 3. The begin-synchronised tasks are represented as tokens on place Tasks. An occurrence of the transition removes a token representing the begin-synchronised tasks (bound to the variable tasks) from place Tasks, the token representing the idle resources (bound to the variable idleres) from place Resources, and the list representing the values of the conditions (bound to the variable conditions) from place Conditions. The transition adds tokens representing the set of tasks to be started to the place Starting, puts the idle resources back on place Resources, and puts a token back on place Conditions updated according to the start effects of the tasks. The updating of conditions is handled by the function `InstantEffects` on the arc expression on the arc from the Start transition to the place Conditions. All idle resources are put back on place Resources since the actual allocation of resources to the tasks are done in a subsequent step and handled by another submodule. The *guard* of the transition specified in the square brackets expresses the predicate that the transition is enabled only if the conditions specified in the preconditions of the tasks are satisfied and the resources are available. This requirement is expressed by the two predicate functions `SatPreconditions` and `ResourcesAvailable`.

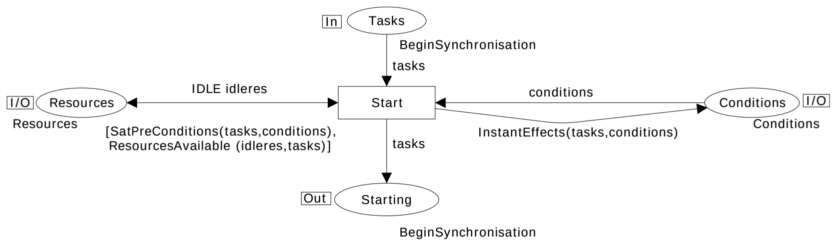


Fig. 4. Submodule for starting of tasks.

The CPN model contains a number of submodules of a similar complexity to the Start submodule shown in Fig. 4 that model the details of task execution and their effect on conditions and resources.

4 State Space Exploration

The main analysis capability of COAST is the computation of *lines of operation* (LOPs). From the previous sections it follows that a COA can be syntactically described as a tuple $COA = (T, C, R, S)$ consisting of a finite set of tasks T, a finite set of conditions C, a finite multi-set of resources R, and a finite set of synchronisations S. The semantics of task execution is defined by the CPN model discussed in the previous section. A LOP for a COA describes an execution of a subset of the tasks in the COA. A LOP of length n is a sequence of tuples (t_i, s_i, e_i, r_i) for $1 \leq i \leq n$ where $t_i \in T$ is a task, s_i is the start time of t_i, e_i is the end time of t_i and r_i is the multi-set of resources assigned to t_i. Two classes of LOPs are considered. *Complete LOPs* are LOPs leading to a *desired end-state* defined by a *goal state predicate* ϕ_{COA} on states that captures the purpose of the COA. *Incomplete* LOPs are LOPs leading to an *undesired end-state* not satisfying ϕ_{COA}. Incomplete LOPs typically arise in early phases of COA development and may be caused by insufficient resources implying that certain tasks cannot be executed, or logical errors caused, e.g., by missing tasks. COAST also supports the planning staff in identifying such errors and inconsistencies in the COA.

The computation of the set of complete and incomplete LOPs is based on state space exploration of the CPN model. Figure 5 shows the state space for an example COA with six tasks, T1, T2,…,T6. The nodes correspond to the set of reachable states and the arcs correspond to the occurrences of enabled binding elements (events). Node 1 to the upper left corresponds to the initial state and node 21 to the lower right corresponds to a desired end-state. The state space represents all the possible ways in which the tasks in the COA can be executed, given that tasks will execute as soon as they can.

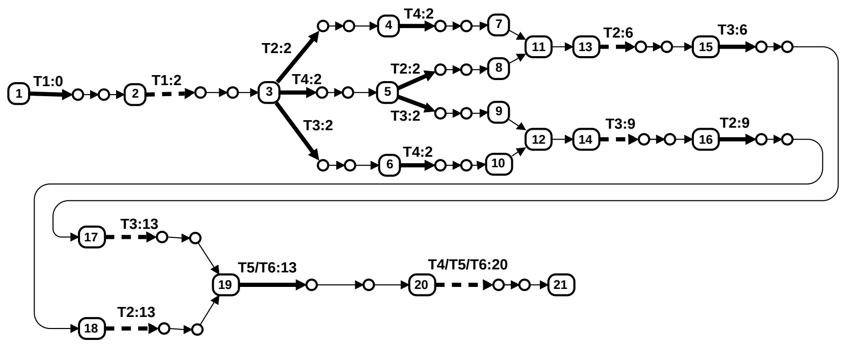


Fig. 5. Example of a state space for the CPN model.

A path in the state space corresponds to a particular execution of the CPN model, i.e., determines a LOP. It is the binding elements in the state space that represent start and termination of tasks that define the LOPs. A distinction is therefore made between visible and invisible binding elements. The *visible binding elements* represent start and termination of tasks, and allocation of resources. All other binding elements are internal events in the CPN model and are *invisible*. The thick arcs in Fig. 5 have labels of the form $Ti : t$ where i specifies the task number and t specifies the time at which the event takes place. Thick solid arcs represent start of tasks and thick dashed arcs represent termination of tasks. The thin arcs represent invisible events. As an example, task T1 starts at time 0 as specified by the label on the outgoing arc from node 1 and terminates at time 2 as specified by the label on the outgoing arc from node 2. The state space in Fig. 5 has four paths leading from the initial state to the desired end-state depending on the branch chosen at node 3 and node 5. When considering the start and termination of tasks it be seen that the four paths determine two complete LOPs L_1 and L_2:

$$L_1 = (\mathsf{T1}, 0, 2), (\mathsf{T2}, 2, 6), (\mathsf{T4}, 2, 20), (\mathsf{T3}, 6, 13), (\mathsf{T5}, 13, 20), (\mathsf{T6}, 13, 20)$$
$$L_2 = (\mathsf{T1}, 0, 2), (\mathsf{T4}, 2, 20), (\mathsf{T3}, 2, 9), (\mathsf{T2}, 9, 13), (\mathsf{T5}, 13, 20), (\mathsf{T6}, 13, 20)$$

Two algorithms to be presented in Sections 4.1 and 4.2 have been implemented in COAST for the computation of LOPs. Both algorithms are based on state space exploration and are complete in that they report all complete and incomplete LOPs. The two algorithms rely on the following theorem that can be proved from the net structure of the CPN model and by inspecting each transition observing that the occurrence of each binding element has a unique effect on the state of the CPN model. We have omitted the proof in this paper since we do not have sufficient space to present the CPN model in full.

Theorem 1. *Let $COA = (T, C, R, S)$ be a COA and let $CPN(COA)$ be the COAST CPN model initialised according to COA. Then the following holds:*

1. *The state space of $CPN(COA)$ is finite, i.e., the CPN model has a finite set of reachable states and a finite set of enabled binding elements in each state.*
2. *The state space is an acyclic directed graph.*
3. *Let σ_1 of length l_1 and σ_2 of length l_2 be two paths in the state space starting from the initial state and both leading to the state s. Then $l_1 = l_2$.*

Item 1 ensures termination of state space exploration, independently of the COA with which the CPN model is initialised. Item 2 implies that there are finitely many paths leading to a given reachable state and hence there can only be finitely many LOPs to be reported. Item 3 ensures that when visiting a state s during a breadth-first state space traversal all predecessors of s will already have been visited. This is exploited by both algorithms.

4.1 Two-Phase Algorithm

The first algorithm is a two-phase algorithm. The first phase is a depth-first construction of the full state space where complete and incomplete LOPs are reported on-the-fly as they are encountered. The second phase is a breadth-first traversal of the state space constructed in the first phase where the LOPs not found in the first phase are computed. Depth-first construction in the first phase allows LOPs to be reported as soon as they are found. The second phase is required since not all LOPs may be reported by the depth-first phase. As an example, a depth-first generation of the state space in Fig. 5 would only find one of the LOPs L_1 and L_2, since node 19 will already have been visited when it is encountered the second time (either via node 17 or 18).

The procedure DepthFirstPhase in Fig. 6 specifies the first phase of the algorithm. It uses three data structures: Nodes which stores the set of nodes (states) generated, Arcs which stores the set of explored arcs, and Stack which is used to store the set of unprocessed states and ensures depth-first generation. The procedure is invoked with the binding element corresponding to the initialisation step of the CPN model.

```
 1: procedure DepthFirstPhase $(s, a, s')$
 2: Arcs.insert $(s, a, s')$
 3: if $\neg$ Nodes.member$(s')$ then
 4:    Nodes.insert$(s')$
 5: end if
 6: if $\phi_{COA}(s')$ then
 7:    LOP.create(Stack.prefix (),complete)
 8: else
 9:    successors $=$ ModelInterface.getnext$(s')$
10:    if successors $= \emptyset$ then
11:       LOP.create (Stack.prefix (),incomplete)
12:    else
13:       Stack.push (successors)
14:    end if
15: end if
16: if $\neg$ Stack.empty() then
17:    DepthFirstPhase (Stack.pop ())
18: end if
```

Fig. 6. Depth-first Phase of LOP computation.

The procedure DepthFirstPhase first inserts the arc (s, a, s') into the set of arcs (line 2) and then checks whether s' has already been visited before (lines 3-4). If s' is a new state, then it is inserted into the set of nodes. If s' corresponds to a desired end-state then the sequence of binding elements executed to reach s' is extracted from the depth-first stack and reported as a complete LOP (lines 6-7). Otherwise the set of successors of s' in the state space is computed (line 9) using the ModelInterface module (see Sect. 2). An incomplete LOP is reported if s' does not have any successor states and is not a desired end-state (line 11). If state space exploration is to continue from s' then the set of successors is pushed onto the stack (line 13). The exploration of the state space continues in lines 16-17 as long as the stack is not empty. When the procedure terminates, Nodes contains the set of reachable states cut-off according to the goal state predicate ϕ_{COA}, and Arcs contains to the set of arcs between the reachable states.

The second phase of the LOP generation is specified in Fig. 7 and conducts a breadth-first traversal of the state space computed in the first phase.

```
 1: procedure BreadthFirstPhase ()
 2: if ¬ (Queue.empty ()) then
 3:    $s$ = Queue.delete ()
 4: end if
 5: lops = LOP.get ($s$)
 6: LOP.delete ($s$)
 7: successors = Arcs.out ($s$)
 8: if successors = $\emptyset$ then
 9:    if $\phi_{COA}(s)$ then
10:       LOP.create (lops,complete)
11:    else
12:       LOP.create (lops,incomplete)
13:    end if
14: else
15:    for all $(s, a, s') \in$ successors do
16:       $lops'$ = LOP.augment ($lops, a$)
17:       LOP.add ($s', lops'$)
18:       if ¬ Visited.member($s'$) then
19:          Visited.insert ($s'$)
20:          Queue.insert ($s'$)
21:       end if
22:    end for
23: end if
```

Fig. 7. Breadth-first Phase of LOP computation.

The procedure BreadthFirstPhase uses three data-structures: Visited which keeps track of states that have been visited, LOP which is used to associate *partial LOPs* with states, i.e., the LOPs corresponding to the possible ways in which the given state can be reached, and Queue that implements the breadth-first traversal queue. The procedure is invoked with the initial state inserted into the queue. The procedure starts by selecting a state s to be processed from the queue and obtaining the LOPs stored with the state s (lines 3-6). The partial LOPs stored with s are then deleted since all predecessors of s will have been processed according Thm. 1(3). If the state has no successors (line 8) then the associated

LOPs are reported as complete if s is a goal state; otherwise they are reported as incomplete LOPs. If the state s has successors then these successors are handled in turn (lines 15-22) by augmenting the partial LOPs from s according to the event executed to reach the successor state s'. These augmented LOPs will then be added to the LOPs associated with s' (line 17). If the successor state has not been visited before, then it is inserted into the queue and into the set of visited states (lines 18-21).

4.2 Sweep-Line Algorithm

The second algorithm implemented in COAST is based on a variant of the sweep-line method [10]. This sweep-line method reduces peak memory usage by not keeping the complete state space in memory. The basic idea of the sweep-line method is to exploit a notion of *progress* found in many systems to delete states from memory during state space construction and thereby reduce the peak memory usage. The states that are deleted are known to be unreachable from the set of unexplored states and can therefore be safely deleted without compromising the termination and complete coverage of the state space construction. The COAST CPN model exhibits progress which is reflected in the state space of the CPN model which according to Thm. 1 is acyclic and has the property that all paths to a given state have the same length. This property implies that when conducting a breadth-first construction of the state space, it is possible to delete a state s when it is removed from the breadth-first queue for processing. The reason is that when the state s is removed from the queue all the predecessors s have been processed, and hence s is no longer needed for comparison with newly generated states. The basic idea in the application of the sweep-line method in the context of COAST is therefore to construct the state space in a breadth-first order and compute the LOPs on-the-fly during the state space construction in a similar way as in the breadth-first traversal in the two-phase algorithm. At any given moment the algorithm only stores the nodes and associated partial LOPs on the frontier of the state space exploration and the peak memory usage will be reduced.

4.3 Experimental Results

Table 1 provides a set of experimental results with the algorithms presented in the previous subsections on some representative examples. All experimental results have been obtained on a Pentium III 1 GHz PC with 1Gb of main memory. The first part of the table lists the number of Nodes and Arcs in the state space. The second part gives the experimental results for the two-phase algorithm. It specifies the total CPU Time in seconds used by the algorithm and the percentage of the time spent in the depth-first phase (DFTime) and the breadth-first phase (BFTime). For the depth-first phase it also specifies the percentage of time spent on the calculation of enabled binding elements (DFEna) and inserting newly generated states and arcs into the state space (DFSto). The third part of the table specifies the results for the sweep-line algorithm by giving the total CPU

Time in seconds used for exploring the state space in percentage of the CPU time for the two-phase algorithm and the Peak number of states stored during the exploration in percentage of the total number of states. For the two-phase algorithm it can be seen that most time is spent in the depth-first phase. This is expected since this is where the actual state space construction is conducted. The time spent in the depth-first phase is divided almost evenly between the storage of nodes and arcs, and the computation of enabling. Almost no time is spent in the breadth-first phase which shows that the time used on computing the LOPs is insignificant compared to the time used to construct the state space. The results for the sweep-line algorithm show that the peak number of stored states is reduced to between 27.5 % and 70 % depending on the example. The sweep-line is faster than the two-phase algorithm which may at first seem surprising. The reason is that states are only present in the breadth-first queue, and hence the relative expensive check for whether a state has already been visited has been eliminated.

Table 1. Selected experimental results.

COA	State space		Two-Phase					Sweep-Line	
	Nodes	Arcs	Time	DFTime	DFSto	DFEna	BFTime	Peak	Time
COA_1	283	317	2.81	99.3 %	44.1 %	53.7 %	0.36 %	45.2 %	71.2 %
COA_2	1,253	1,443	16.13	99.7 %	44.7 %	53.9 %	0.25 %	27.5 %	61.9 %
COA_3	2,587	2,974	34.06	99.7 %	44.8 %	53.6 %	0.29 %	38.5 %	58.7 %
COA_4	77	85	0.36	97.2 %	38.9 %	58.3 %	2.78 %	46.8 %	83.3 %
COA_5	142	169	1.68	99.4 %	39.9 %	58.9 %	0.59 %	59.9 %	59.6 %
COA_6	8,263	10,394	101.94	99.6 %	43.2 %	54.9 %	0.35 %	70.0 %	59.8 %
COA_7	1,977	2,249	25.72	99.7 %	43.9 %	54.5 %	0.27 %	39.5 %	62.2 %

The *reachability of a goal state problem* solved by COAST server when computing the LOPs is equivalent to the *reachability of a submarking problem* which is known to be PSPACE-hard for one-safe Petri nets [4]. Since the CPN model when initialised with a COA $COA = (T, C, R, S)$ can be unfolded to a one-safe Petri net of size $\Theta(|T|+|C|+|R|+|S|)$, i.e., a Petri net which is linear in the size of the COA, and since any one-safe Petri net can be obtained by selecting the proper COA, this implies that the problem solved by COAST is PSPACE-hard.

The example COAs that COAST has been tested against consist of 15 to 30 tasks resulting in state spaces with 10,000 to 20,000 nodes and 25,000 to 35,000 arcs. Such state spaces can be generated in less than 2 minutes on a standard PC. The state spaces are relatively small because the conditions, available resources, and imposed synchronisations in practice strongly limit the possible orders in which the tasks can be executed. This observation is one of the main reasons why state space construction appears to be a feasible approach for COAST.

Both the two-phase algorithm and the sweep-line algorithm are implemented because their relative performance depends on the structure of the state space. The two-phase algorithm has the advantage that it can report LOPs early due to the depth-first construction where LOPs can be extracted from the depth-first search stack. This means that the two-phase algorithm is able to work well on very wide state spaces that may be too large to explore in full using the sweep-line algorithm. The two-phase algorithm is implemented in such a way that it is possible to terminate the LOP generation when a certain number of LOPs (specified by the user) has been found. Wide state spaces are caused by interleaving when there are very few constraints on the execution of tasks in the COA. The sweep-line algorithm, on the other hand, performs better on long and narrow state spaces, e.g., when there are many but highly constrained tasks in the COA. The two algorithms therefore complement each other.

5 Conclusions

This paper has demonstrated how formal modelling has been applied in a model-based development of the COAST server. CP-nets were applied since a COA consisting of tasks, resources, conditions, and synchronisations is naturally viewed as a concurrent system. The main benefit of using a formal modelling language for concurrent systems has, in our view, been that it allowed us to concentrate on formalisation of the task execution framework for COAST and abstract from implementation issues. Furthermore, the graphical representation of CP-nets was extremely useful for discussions in the development process. The main advantage of our approach is that the resulting formal model is then directly embedded in the final implementation. This effectively eliminates the challenging step of going from a formal specification to its implementation. Furthermore, our practical experiments on representative examples have demonstrated that state space exploration is a feasible analysis approach within the COAST domain.

Our approach requires that the modelling language is expressive enough to support a level of parameterisation that makes it possible to initialise the constructed model with problem instances. Furthermore, the computer tool supporting the formal modelling language must support the extraction of models in executable form that allows the transition relation of the model to be accessed. The methodology applied for the development of COAST is therefore also applicable to other formal modelling languages where the above requirements are satisfied, and generally applicable to cases where a formalisation of a particular domain is required as a basis for the development of a domain specific tool.

References

1. T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times - A Tool for Modelling and Implementation of Embedded Systems. In *Proc. of TACAS'02*, volume 2280 of *LNCS*, pages 460–464, 2002.
2. CPN Tools. `www.daimi.au.dk/CPNtools`.
3. S. Edelkamp. Promela Planning. In *Proc. of SPIN'03*, volume 2648 of *LNCS*, pages 197–212, 2003.

4. J. Esparza. Decidability and Complexity of Petri net Problems - An Introduction. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 374–428. Springer-Verlag, 1998.
5. G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In *Proc. of the 3rd Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 79–93. Department of Computer Science, University of Aarhus, 2001. DAIMI PB-554.
6. M. Ghallab, D.Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.
7. K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use. Vol. 1-3*. Springer-Verlag, 1992-1997.
8. 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.
9. L.M. Kristensen, J.B. Jørgensen, and K. Jensen. Application of Coloured Petri Nets in System Development. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 626–686. Springer-Verlag, 2004.
10. L.M. Kristensen and T. Mailund. Efficient Path Finding with the Sweep-Line Method using External Storage. In *Proc. of ICFEM'03*, volume 2885 of *LNCS*, pages 319–337, 2003.
11. J.I. Rasmussen, K.G. Larsen, and K. Subramani. Resource-Optimal Scheduling Using Priced Timed Automata. In *Proc. of TACAS'04*, volume 2988 of *LNCS*, pages 220–235. Springer-Verlag, 2004.
12. B. Schätz. Model-Based Development: Combining Engineering Approaches and Formal Techniques. In *Proc. of ICFEM'04*, volume 3308 of *LNCS*, pages 1–2, 2004.
13. J.D. Ullman. *Elements of ML Programming*. Prentice-Hall, 1998.
14. A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 429–528. Springer-Verlag, 1998.
15. L. Zhang, L.M. Kristensen, C. Janczura, G. Gallasch, and J. Billington. A Coloured Petri Net based Tool for Course of Action Development and Analysis. In *Proc. of Workshop on Formal Methods Applied to Defence Systems*, volume 12 of *CRPIT*, pages 125–134. Australian Computer Society, 2001.
16. L. Zhang, L.M. Kristensen, B. Mitchell, C. Janczura, G. Gallasch, and P. Mechlenborg. COAST – An Operational Planning Tool for Course of Action Development and Analysis. In *Proc. of 9th International Command and Control Research and Technology Symposium*, 2004.

From Task Descriptions via Coloured Petri Nets Towards an Implementation of a New Electronic Patient Record

Jens Bæk Jørgensen[1], Kristian Bisgaard Lassen[1], and Wil M. P. van der Aalst[2]

[1] Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
{jbj,k.b.lassen}@daimi.au.dk

[2] Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl

Abstract. We consider a given specification of functional requirements for a new electronic patient record system for Fyn County, Denmark. The requirements are expressed as task descriptions, which are informal descriptions of work processes to be supported. We describe how these task descriptions are used as a basis to construct two executable models in the formal modeling language Colored Petri Nets (CPNs). The first CPN model is used as an execution engine for a graphical animation, which constitutes an Executable Use Case (EUC). The EUC is a prototype-like representation of the task descriptions that can help to validate and elicit requirements. The second CPN model is a Colored Workflow Net (CWN). The CWN is derived from the EUC. Together, the EUC and the CWN are used to close the gap between the given requirements specification and the realization of these requirements with the help of an IT system. We demonstrate how the CWN can be translated into the YAWL workflow language, thus resulting in an operational IT system.

Keywords: Workflow Management, Executable Use Cases, Colored Petri Nets, YAWL.

1 Introduction

In this paper, we consider how to come from a specification of user requirements to a realization of these requirements with the help of an IT system.

Our starting point is a requirements specification for a new Electronic Patient Record (EPR) system for Fyn County [10]. Fyn County is one of the 13 counties in Denmark and is responsible for all hospitals and other health-care organizations in its county. We focus on functional requirements for the new EPR system for Fyn County; specifically, we look at seven work processes that must be supported. The work processes cover what can happen from the moment a patient is considered for treatment at a hospital until the patient is eventually dismissed or dead.

In the requirements specification, these work processes are presented in terms of *task descriptions* [18, 19], in the sense of Søren Lauesen. A task description is an informal, prose description. An essential characteristic of a task description is that it specifies what users and IT system do together. In contrast to a *use case* [9], the split of work between users and IT system is not determined at this stage. Task descriptions are meant to be used at an early stage in requirements engineering and software development projects.

This means that there is a natural and large gap between a task description and its actual support by an IT system. To help bridging this gap, we propose to use *Colored Petri Nets (CPNs)* [14, 17] models. CPNs provide a well-established and well-proven language suitable for describing the behavior of systems with characteristics like concurrency, resource sharing, and synchronization. CPN are well-suited for modeling of workflows or work processes [4]. The CPN language is supported by *CPN Tools* [27], which has been used to create, simulate, and analyze the CPN models that we will present in this paper.

Figure 1 outlines the overall approach to be presented in this paper.

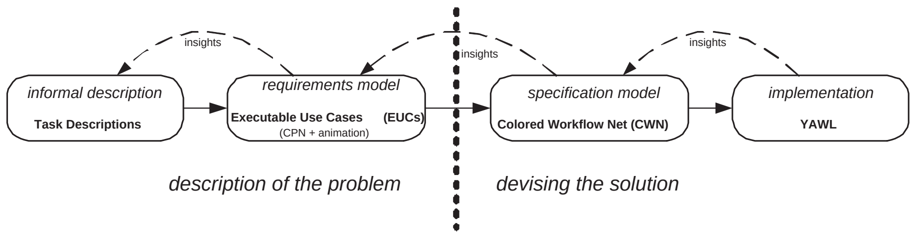


Fig. 1. Overall approach.

The boxes in the figure present the artifacts that we will consider in this paper. A solid arrow between two nodes means that the artifact represented by the source node is used as basis to construct the artifact represented by the destination node.

The leftmost node represents the given task descriptions. Going from left to right, the next node represents an *Executable Use Case (EUC)* [16], which is a CPN model augmented with a graphical animation. EUCs are formal and executable representations of work processes to be supported by a new IT system, and can be used in a prototyping fashion to specify, validate, and elicit requirements. The node *Colored Workflow Net* (CWN) represents a CPN model, derived from the EUC CPN, that is closer to an implementation of the given requirements. The rightmost node represents the realization of the IT system itself. In this case study, a prototype has been developed using the YAWL workflow management system [1].

The vertical line in the middle of the figure marks a significant division between "analysis artifacts" to the left and "design and implementation artifacts" to the right. The analysis artifacts represent descriptions of the problems to be solved, in the form of specifying the core work processes that must be supported by the new IT system. To the left of the line, the focus is on describing the problems, not on devising solutions to these problems. In particular, to the left of the line, it is not specified exactly what we want the new IT system itself to do. The arrow between the nodes Executable Use Cases and Colored Workflow Nets represents the transition from analysis, in the form of describing the problem, to design, in the form of devising the solution.

It should be noted that we are not advocating any particular kind of development process in this paper. Figure 1 should not be read to imply that we are proposing waterfall development. There will often be iterations back and forth between the artifacts in consideration, as is indicated by the dashed arrows.

The case-study presented in this paper is used to illustrate Figure 1. It has been taken from the medical domain. As pointed out in [22, 23] "careflow systems" pose particular requirements on workflow technology, e.g., in terms of flexibility. Classical workflow-based approaches typically result in systems that restrict users. As will be shown in this paper, task descriptions aim at avoiding such restrictions. Moreover, the state-based nature of CPNs and YAWL allows for more flexibility than conventional event-based systems, e.g., using the *deferred choice pattern* [2], choices can be resolved implicitly by the health-care workers (rather than an explicit decision by the system).

This paper is related to one of our previous publications [3] where we also apply CPN Tools to model EUCs and CWNs. However, in the earlier work, we considered a different domain, namely banking, we did not consider task descriptions, and we used BPEL as target language instead of YAWL.

This paper is structured as follows: Section 2 is about task descriptions, both in general and about the specific task description we will use as case study. Section 3, in a similar fashion, is about Executable Use Cases (EUCs). In Section 4, we describe the Colored Workflow Net (CWN). Section 5 considers the realisation of the system. Related work is discussed in Section 6 and the conclusions are drawn in Section 7.

2 Task Descriptions

In this section, we first present task descriptions in general and then we introduce the specific task description related to Fyn County's Electronic Patient Record (EPR) that we will focus on in this paper. Finally, we motivate why we move from task descriptions only to EUCs rather than directly implementing the system.

2.1 Task Descriptions in General

In this context, a *task* is a unit of work that must be accomplished by users and an IT system together. A task forms a unit in the sense that after having completed a task, it will feel natural for the user to take a break. Tasks may be split into *subtasks*. An example of a subtask is "register patient".

The *descriptions of subtasks* in a task description are on the left side of the dividing line in Figure 1. However, a task description may also contain proposals about how to support the given subtasks. *Solution proposals* constitute descriptions, which are to the right of the split line in Figure 1. The explicit division into subtasks and solution proposals enforces a strict split between describing a problem and proposing a solution. With solution proposals, the description then properly changes name to a *Task and Support description.* A solution proposal for the subtask “register patient” could be “transfer data electronically from own doctor”.

Variants in task description are used to specify special cases in a subtask. Instead of writing a complex subtasks, [19] suggests to extract the special cases in variants, making the subtasks and variants easier to read.

2.2 Task Descriptions for Fyn County’s EPR

The task descriptions for Fyn County’s EPR that we consider are the following:

1. Request before patient arrives
2. Patient arrives without prior appointment
3. Reception according to appointment
4. Mobile clinical session
5. Stationary clinical session
6. Terminate course of events
7. Patient dies

Task descriptions for each of these seven work processes are given in [10] (in Danish). In this paper, we will use the task description for “Request before patient arrives” to illustrate our approach. This task description is translated into English and presented in Table 1. As can be seen, it is a task and support description. Except from the translation from Danish to English, the task description is presented here unchanged (which explains the presence of question marks and other peculiarities).

Table 1: Task description: Request before patient arrives

Task 1: Request before patient arrives		
Establish episode of care or continue the establishment process if it had been parked or transferred. The request can involve a clinical session where the episode of care is refined before the patient arrives. **Start** Request from the patient's practitioner, specialist doctor, other hospital, or authority. Request can also be supplementary information that were missing previously, or when the task was transferred to another person (e.g. from the secretary to the doctor). **End** When the episode of care is established/adjusted and the patient called in or added to the waiting list. **Frequency** Per user: ??. For the whole hospital: ??. **Critical situations** **Users** The secretary is the immediate user, but the task can be transferred to others.		
Subtask and variant number	Subtask	Solution proposal
1.	Register patient. (See data description)	Transfer data electronically from the patients doctor, etc. (Medcom)
1a.	Patient exist in system. Update data	
1b.	Healthy partner must be enrolled	??
1c.	Personal security ??	??
2.	Establish episode of care and register data, i.e., the preliminary diagnosis. (See data description, including support in use of SKS classification.)	Transfer data electronically from own doctor, etc. (Medcom)
2p.	Problem: Diverging code systems and structures in the electronic messages.	Support the manual transfer of data from the electronic data form to the system form.
2a.	Episode of care is already established. Data may need to be adjusted, e.g., date of patient appointment.	
2q.	Problem: The patient can concurrently be involved in other episodes of care and be enrolled more places and in more departments. It can be hard to get an overview of who has the nursing responsibility and who is providing a bed. Also, there may be a need to see previous episode of care, given that the patient agrees.	

3.	Possible clinical session to plan the episode of care (e.g. if the establishment process is transferred to a doctor).	
4.	Print patient call-up (or other form of call-up).	
4a.	Patient is transferred to the waiting list	
4b.	Information is missing and the task is parked with time monitoring	
4c.	The case is transferred to another, perhaps with time monitoring.	
4d.	The request is possibly denied.	
5.	Request interpreter for the time of admission.	

2.3 From Task Descriptions to Executable Use Cases

One of the main motivations behind task descriptions is to alleviate some problems related to use cases. A use case describes an interaction between a computer system and one or more external actors. In the sense of Sommerville [25], use cases are effective to capture "interaction viewpoints", but not adequate for "domain requirements". A task description typically has a broader perspective than a use case, and, as such, is a means to address domain requirements as well.

In a use case description, the split of work between users and system is determined. In contrast, in a task description, this split of work is not fixed. A task description describes what the user and the system must do together. Deciding who does what is done at a later stage. Thus, a task description can help to avoid making premature (and sometimes arbitrary) design decisions. In other words, a task description is a means to help users to keep focus on their domain and the problems to be solved, instead of drifting into designing solutions of sometimes ill-defined and badly understood problems.

On the other hand, use cases and task descriptions share the salient characteristics that they are static descriptions: They are mainly prose text (may be structured or semi-structured) possibly supplemented with some drawings, e.g., containing ellipses, boxes, stick men, and arrows as in UML use case diagrams. Both task descriptions and use cases may be read, inspected, and discussed, and in this way, they may be improved. *However, both use cases and task descriptions lack the ability to "talk back to the user". Even though they describe behavior, the descriptions themselves are not dynamic and cannot be made subject for experiments and investigations in a trial-and-error fashion.* In comparison, prototypes have these properties.

A traditional prototype, though, tends to focus on an IT system itself, in particular on that system's GUI, more than explicitly on the work processes to be supported by the new IT systems. This has been a main motivation to introduce EUCs as a means to be used in requirements engineering; to provide executable descriptions of new work processes and possibly of their intended computer support, and in this way, be able to talk back to the user — facilitating discussions about both work processes and IT systems support.

3 Executable Use Cases (EUCs)

In this section, we first present EUCs in general and then we introduce the specific EUC related to Fyn County's EPR that we will focus on in this paper. We also consider how to come from EUCs to CWNs.

3.1 Executable Use Cases in General

An EUC consists of three tiers, as indicated in Figure 2.

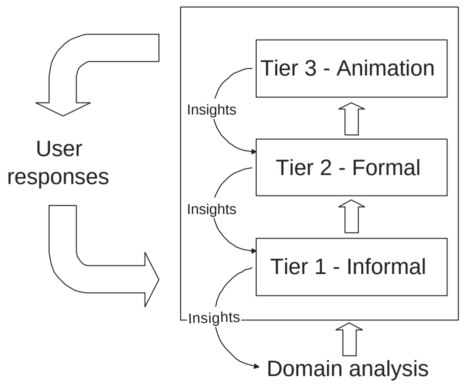


Fig. 2. Executable Use Cases.

Each tier represents the considered work processes that must be supported by a new system. The tiers use different representations: Tier 1 (the *informal tier*) is an informal description; Tier 2 (the *formal tier*) is a formal, executable model; Tier 3 (the *animation tier*) is a graphical animation of Tier 2, which uses only concepts and terminology that are familiar to and understandable for the future users of the new system.

As indicated by Figure 2, the three tiers of an EUC should be created and executed in an iterative fashion. The first version of Tier 1 is based on domain analysis, and the first version of tiers 2 and 3, respectively, is based on the tier immediately below.

The formal tier of an EUC may in general be created in a number of formal modeling languages. We have chosen CPN because we have good experience with this language and its tool support, but other researchers and practitioners may have other preferences, e.g., other options could be statecharts [12] or UML activity diagrams [20].

As was mentioned in Section 2.3, EUCs have notable similarities with traditional high-fidelity prototypes of IT systems; this comparison is made in more detail in [8]. In [15], it is described how an EUC can be used to link and ensure consistency between, in the sense of Jackson [13], user-level requirements and technical software specifications. Jackson's division into requirements and specifications resembles the division into subtasks and solution proposals in task descriptions. User-level requirements and subtasks lie to the left of the dividing line in Figure 1; technical software specifications and solution proposals lie to the right.

Like a task description, an EUC can have a broader scope than a traditional use case. The latter is a description of a sequence of interactions between external actors and a system that happens at the interface of the system. An EUC can go further into the environment of the system and also describe potentially relevant behavior in the environment that does not happen at the interface. Moreover, an EUC does not necessarily fully specify which parts of the considered work processes will remain manual, which will be supported by the new system, and which will be entirely automated by the new system. An EUC can be similar to, indeed, a task description. Therefore, Executable Use Cases do not necessarily have the most suitable name. The name "executable use cases" was originally chosen to make it easy to explain the main idea of our approach to people, who were already familiar with traditional prose use cases.

3.2 Executable Use Case for Fyn County's EPR

We have made an EUC that covers all seven task descriptions listed in the beginning of Section 2.2. The EUC lies strictly on the left-hand side of the dividing line in Figure 1, i.e., the EUC does not include solution proposals.

In this section, we will present the part of the EUC that corresponds to the task description of Table 1. The informal tier of the EUC is the task description itself.

An extract of the formal tier is shown in Figure 3; this figure presents the CPN model that corresponds to the task description from Table 1. Note that this is only one of the seven task descriptions for Fyn County's EPR.

Thick lines denote the path that the user and system has to complete to solve the task; i.e. to go from the place `Ready to make appointment` to `Patient ready for arrival`. *Solid lines* denote subtasks and variants of subtasks. *Dashed lines* denote added structure to the model to assert that desired interleavings of subtasks/variants are possible.

In Figure 4, we outline how the formal and animation tiers are related. At the bottom, we see the formal tier executing in CPN Tools. Please note that the shown module of the CPN model contains seven transitions (the rectangles), and that each of these transitions corresponds to one of the considered tasks (cf. the list in the beginning of Section 2.2). At the top is the animation tier in BRITNeY, the new animation facility of CPN Tools. The two tiers are connected by adding animation drawing primitives to transitions in the CPN model. These primitives update the animation.

The animation tier is a view on the state of, and actions in the formal tier. When a transition occurs in the formal model it is reflected by updates to the animation tier. Therefore, the behaviors of the two tiers remain synchronized.

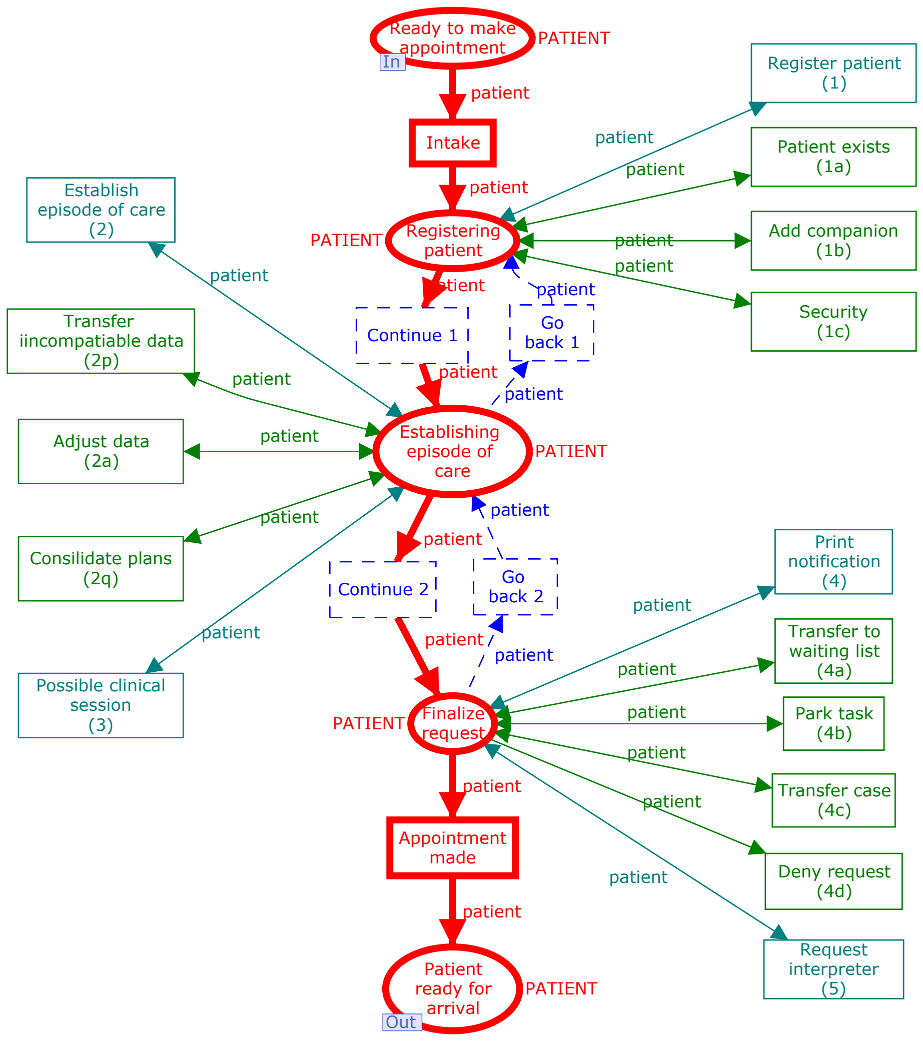


Fig. 3. Task 1 modeled in CPN

Using the animation tier the user can interact with each of the seven tasks. Within the animation of each task, subtasks can be selected and executed. When a subtasks is chosen for execution, the animation user can see visually what is happening and see which entities that are involved in completing the subtask. In the snapshot shown in Figure 4, the animation visualizes Task 1. It shows that the animation user has chosen to execute subtasks 1, 3, 4, and is about to execute Subtask 4a. We also see that Subtask 4a involves a computer and a secretary.

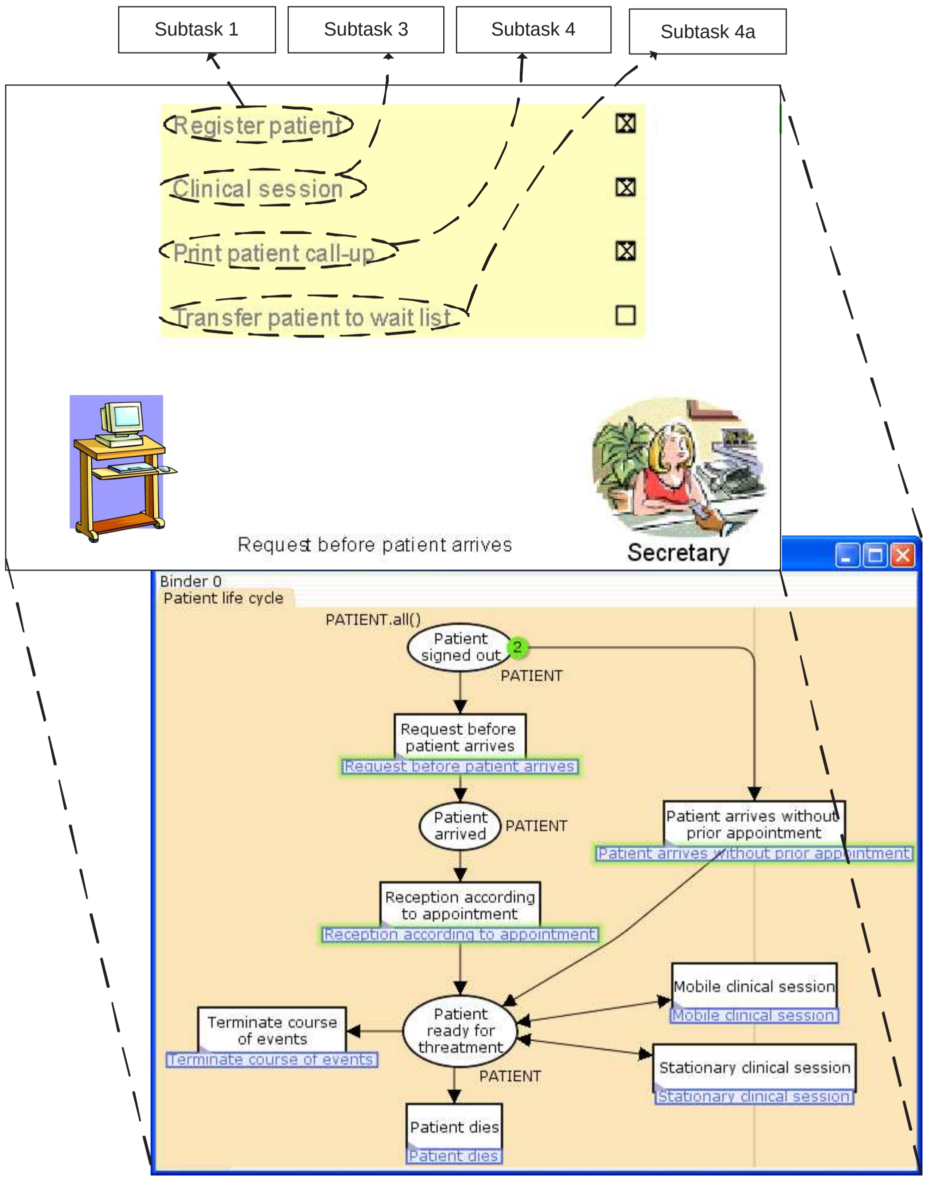

Subtask 1
Subtask 3
Subtask 4
Subtask 4a
Register patient
Clinical session
Print patient call-up
Transfer patient to wait list
Request before patient arrives
Secretary
Binder 0
Patient life cycle
PATIENT.all()
Patient signed out
2
PATIENT
Request before patient arrives
Patient arrived
PATIENT
Patient arrives without prior appointment
Reception according to appointment
Mobile clinical session
Terminate course of events
Patient ready for threatment
Stationary clinical session
PATIENT
Patient dies

Fig. 4. Connection between animation and formal layer

In the task description in Table 1, it was not mentioned, who does what. It is us, the creators of the EUC (software people), who have interpreted the subtasks in this way, i.e., described who does what and what a normal execution of a task is. When showing this animation to the staff at a hospital in Fyns County, we are likely to get more feedback on our interpretations of their daily work than we could get with the static task descriptions only.

3.3 From Executable Use Cases to Colored Workflow Nets

The EUC we have presented above describes real-world work processes at a hospital. When these work processes are to be supported by a new IT system, of course, what goes on inside that system is highly related to what goes on in the real world.

In the approach of this paper (cf. Figure 1), we make separate models of real-world work processes at a hospital (the EUC) and the IT system that must support these work processes (the CWN). This is done to clearly distinguish between the real world, on one hand, and the software, on the other hand. This distinction is advocated by a number of software experts, see, e.g., [13]. Not making this distinction may cause serious confusion.

In this way, the CWN we will now present describes the IT system, and, as we will see, can be used to automatically generate parts of that system.

4 Colored Workflow Nets

A *Colored Workflow Net* (CWN) [3] is a CPN as defined in [17]. Although both the CWN and the formal tier of the EUC use the same language, there are some notable differences. First of all, the scope of the CWN is limited to the IT system, i.e., only those activities that are supported by the system appear in the model. Second, the CWN covers the *control-flow* perspective, the *resource* perspective, and the *data/case* perspective [4]. In the case study of this paper, the EUC covered the control-flow perspective only, but as we move to the right in Figure 1, it is necessary to include the other perspectives as well (if they have not already been included). Finally, CWNs are restricted to a subset of the CPN language, i.e., CWNs need to satisfy some syntactical and semantical requirements to allow for the automatic configuration of a workflow management system [3].

Although a CWN covers the control-flow, resource, and data/case perspectives, it abstracts from implementation details and language/application specific issues. A CWN should be a CPN with only places of type `Case` or `Resource`. These types are as defined in Table 2.

A token in a place of type `Case` refers to a case and some or all of its attributes. Each case has an ID and a list of attributes. Each attribute has a name and a value. Tokens in a place of type `Resource` represent resources. Each resource has an ID and a list of roles and organizational units. The distribution of resources over roles and organizational units can be used in the allocation of resources. For more details on CWNs, we refer to [3].

Figure 5 shows the CWN for the task `Request before patient arrives`. When comparing this CWN with the EUC CPN shown in Figure 3, several differences can be observed. First of all, some subtasks shown in the EUC CPN

Table 2. Places in a CWN need to be of type `Case` or `Resource`

```
colset CaseID = union C:INT;
colset AttName = string;
colset AttValue = string;
colset Attribute = product AttName * AttValue;
colset Attributes = list Attribute;
colset Case = product CaseID * Attributes timed;
colset ResourceID = union R:INT;
colset Role = string;
colset Roles = list Role;
colset OrgUnit = string;
colset OrgUnits = list OrgUnit;
colset Resource = product ResourceID * Roles * OrgUnits timed;
```

are not included in the CWN because they will not be supported by the IT system. Subtask 1b (`Add companion`) and Subtask 2q (`Consolidate plans`) are not included because of this reason. Secondly, Figure 5 includes more explicit references to the resource and data/case perspectives. Note that Figure 5 shows three resource places of type `Resource` defined in Table 2. These resource places hold information on the availability and capabilities of people. Using the concept of a fusion place [14, 27], these places together form one logical entity. Places of type `Case` hold information on cases. Cases have several attributes such as `patient name`, `patient id`, `address`, `birth date`, `preliminary diagnosis`, etc. In Figure 5, the relevant attributes are only shown for the task `Register patient`, but, for the sake of readability, not shown for all other tasks.

One of the advantages of using Petri nets is the availability of a wide variety of analysis techniques. In CPN Tools it is possible to simulate models and to do state-space analysis. We have used both facilities. For the state-space analysis we have abstracted from time and color to asses soundness [4]. Initially, we discovered a minor error (a deadlock because we did not connect Subtask 4d properly). However, after repairing this, the CWN was sound. Note that reachability graph of the CWN shown in Figure 5 for one patient has only 14 nodes and 29 arcs, so it is easy to verify its correctness by hand. However, for more complicated CWNs, automated state-space analysis of CPN Tools is indispensable to asses correctness before implementation.

5 Realization of the System Using YAWL

In [3], it was shown that for some CWNs it is possible to automatically generate BPEL template code [7]. The *Business Process Execution Language for Web Services* (BPEL4WS or short BPEL) [7] is a textual XML based language that has been developed to form the “glue” between webservices. Although it is an expressive language, it tends to result in models that are difficult to under-

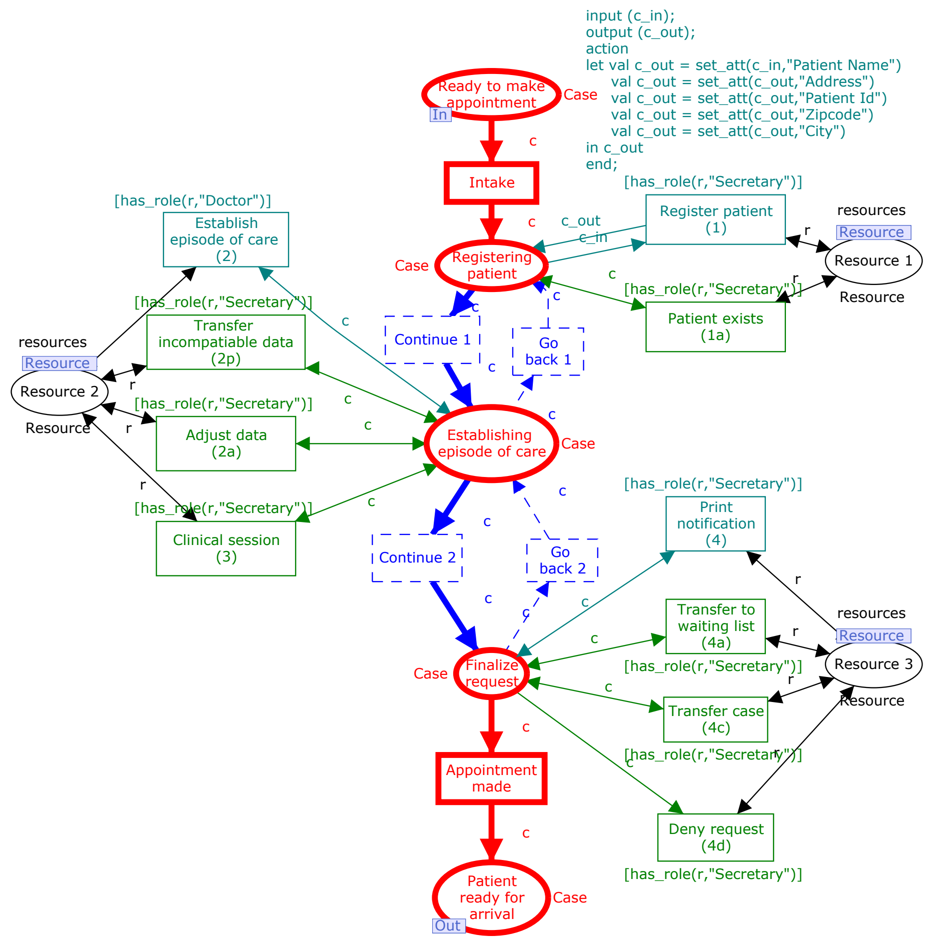

input (c_in);
output (c_out);
action
let val c_out = set_att(c_in,"Patient Name")
val c_out = set_att(c_out,"Address")
val c_out = set_att(c_out,"Patient Id")
val c_out = set_att(c_out,"Zipcode")
val c_out = set_att(c_out,"City")
in c_out
end;
Ready to make appointment
In
Case
Intake
Registering patient
Register patient (1)
Patient exists (1a)
[has_role(r,"Secretary")]
Resource 1
Resource 2
Resource 3
resources
Resource
Establish episode of care (2)
[has_role(r,"Doctor")]
Transfer incompatiable data (2p)
Adjust data (2a)
Clinical session (3)
Continue 1
Go back 1
Establishing episode of care
Continue 2
Go back 2
Print notification (4)
Transfer to waiting list (4a)
Transfer case (4c)
Deny request (4d)
Finalize request
Appointment made
Patient ready for arrival
Out
c_out
c_in
c
r

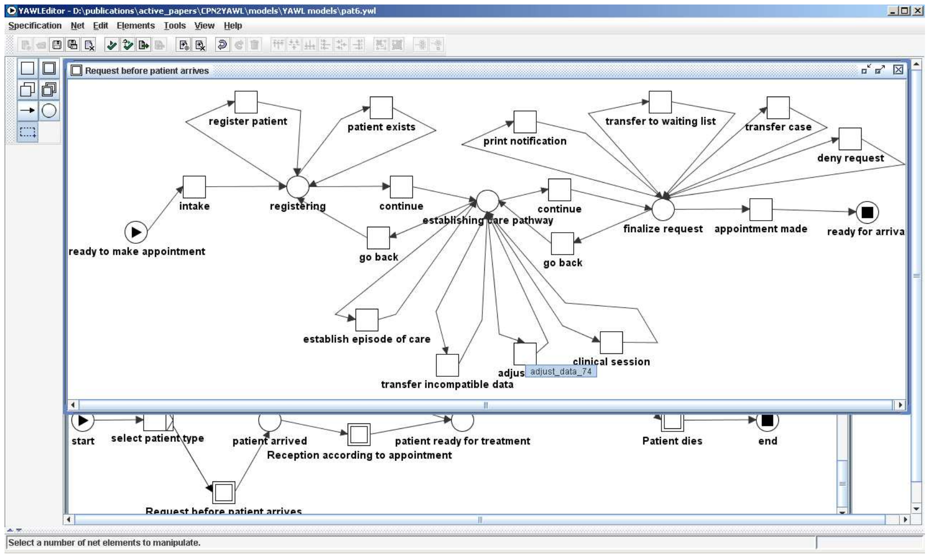
Fig. 5. CWN for the task `Request before patient arrives`

stand and maintain. For example, it is not possible to show BPEL code to end users (e.g., to visualize management information or to allow for dynamic change [24]). Moreover, BPEL offers little flexibility and no support for the resource perspective.[3] Therefore, we decided to use YAWL [1] rather than BPEL.

YAWL (Yet Another Workflow Language) [1] is based on the well-known workflow patterns (www.workflowpatterns.com, [2]) and is more expressive than any of the other languages available today. Because of its native and unrestricted support of the *deferred choice* pattern [2], it is possible to leave the selection of

[3] Note that only recently people started to investigate adding the resource perspective to BPEL, cf. the *WS-BPEL Extension for People (BPEL4People)* initiative http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.

the next task to the user. This offers more flexibility than BPEL, because it is possible to define for each state what tasks are possible without selecting one (in BPEL this is restricted to the inside of a `pick` activity [7]). Moreover, YAWL also supports the resource perspective (in addition to the control-flow and data perspectives). The language YAWL is also supported by an open source workflow management system that can be downloaded from www.yawl-system.com.


YAWLEditor - D:\publications\active_papers\CPN2YAWL\models\YAWL models\pat6.ywl
Specification Net Edit Elements Tools View Help
Request before patient arrives
register patient
patient exists
print notification
transfer to waiting list
transfer case
deny request
intake
registering
continue
establishing care pathway
continue
finalize request
appointment made
ready for arriva
ready to make appointment
go back
go back
establish episode of care
transfer incompatible data
adjus
adjust_data_74
clinical session
start
select patient type
patient arrived
Reception according to appointment
patient ready for treatment
Patient dies
end
Request before patient arrives

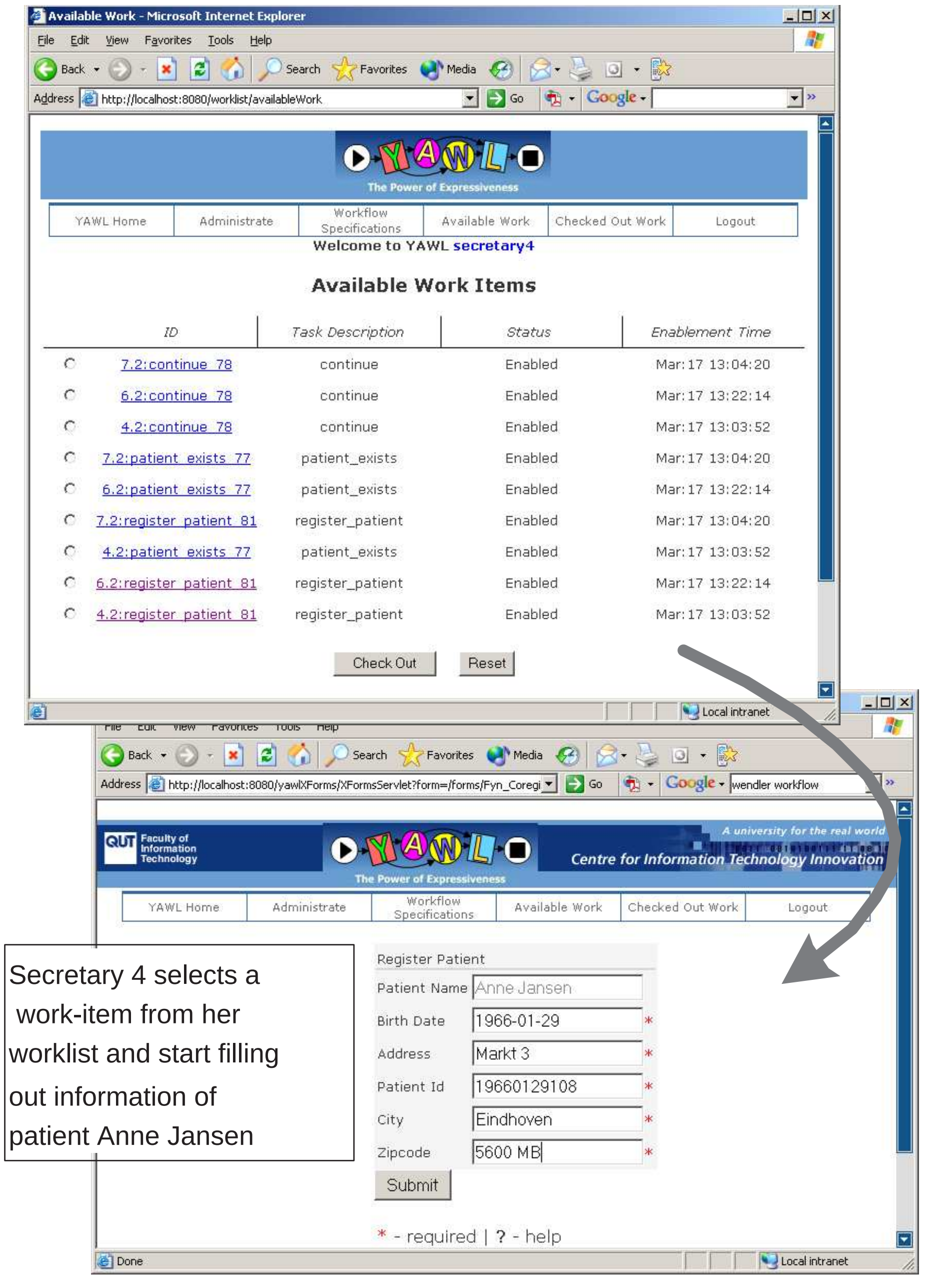
Select a number of net elements to manipulate.

Fig. 6. Screenshot of YAWL editor

Given the fact that YAWL can be seen as a superset of CWNs, it was easy to translate the running example from CPN in YAWL. Figure 6 shows the top-level workflow and the composite task `Request before patient arrives`. Although both models look quite different, a fairly direct mapping was possible from the CWN shown in Figure 5 to the YAWL model shown in Figure 6. All places of type `Case` in Figure 5 are mapped onto conditions in YAWL and transitions in Figure 5 are mapped onto YAWL tasks.[4]

After mapping the CWNs onto a YAWL specification, it is possible to enact the associated workflows. Figure 7 shows a work-list and a form generated by YAWL. The left-hand side of the figure shows the work-list of the secretary with user code `secretary4`. It shows work-items associated to three cases. Each of these three cases is in the state `registering` where three tasks are enabled. Therefore, there are 3*3=9 possible work-items. After selecting a work-item related to task `register patient`, three work-items disappear from the work-

[4] Note that *subtasks* in Task Descriptions correspond to *transitions* in CWNs and *tasks* in YAWL.


Available Work - Microsoft Internet Explorer
http://localhost:8080/worklist/availableWork
The Power of Expressiveness
YAWL Home
Administrate
Workflow Specifications
Available Work
Checked Out Work
Logout
Check Out
Reset
Local intranet
http://localhost:8080/yawlXForms/XFormsServlet?form=/forms/Fyn_Coregi
wendler workflow
Faculty of Information Technology
A university for the real world
Centre for Information Technology Innovation
Submit
Done

Welcome to YAWL secretary4

Available Work Items

ID	Task Description	Status	Enablement Time
7.2:continue_78	continue	Enabled	Mar:17 13:04:20
6.2:continue_78	continue	Enabled	Mar:17 13:22:14
4.2:continue_78	continue	Enabled	Mar:17 13:03:52
7.2:patient_exists_77	patient_exists	Enabled	Mar:17 13:04:20
6.2:patient_exists_77	patient_exists	Enabled	Mar:17 13:22:14
7.2:register_patient_81	register_patient	Enabled	Mar:17 13:04:20
4.2:patient_exists_77	patient_exists	Enabled	Mar:17 13:03:52
6.2:register_patient_81	register_patient	Enabled	Mar:17 13:22:14
4.2:register_patient_81	register_patient	Enabled	Mar:17 13:03:52

Register Patient

Patient Name	Anne Jansen
Birth Date	1966-01-29 *
Address	Markt 3 *
Patient Id	19660129108 *
City	Eindhoven *
Zipcode	5600 MB *

* - required | ? - help

Secretary 4 selects a work-item from her worklist and start filling out information of patient Anne Jansen

Fig. 7. Screenshot of the YAWL worklist and the form associated to task `register patient`

list (the competing tasks become disabled for this patient) and `secretary4` can fill out a form with patient data. After completing the form there are again nine work-items, etc.

The realization of the workflow process in YAWL completes the overall approach shown in Figure 1, i.e., we moved from informal task descriptions, then to EUCs, after that to CWNs, and finally realized the task descriptions in terms of YAWL. Note that, given the availability of a running YAWL system and a CWN, it is possible to construct a running system in a very short period, e.g., in a few hours it is possible to make the process shown in Figure 6 operational. This does include the generation of user forms as shown in Figure 7 but does not include system integration or the development of dedicated applications. The task of mapping a CWN onto YAWL can be partly automated by using the automatic translation provided by ProM (cf. www.processmining.org). ProM is able to automatically map Petri nets in PNML format onto various other formats, including YAWL. However, this translation does not take data and resources into account, so some manual work remains to be done. Nevertheless, it shows that the overall process shown in Figure 1 is feasible. Moreover, we would like to argue that *initially more time is spent on the requirements, but considerably less time is spent on the actual realization and testing.* The intermediate steps (i.e., EUCs and CWNs) enable an efficient implementation. Moreover, less time needs to be spent on testing the system because its design has been validated and verified earlier. Also, the system is more likely to be accepted by the end-users.

6 Related Work

This paper builds on the work presented in [3], where we also apply CPN Tools to model EUCs and CWNs. However, in [3], EUCs are not linked to task descriptions and we used BPEL as target language instead of YAWL. The extension with task descriptions was inspired by the work of Lauesen [18, 19]. Compared to existing approaches for requirements engineering and use case design [9, 11, 13, 25], our approach puts more emphasis on the two intermediate steps. First of all, we make EUCs with both an animation and formal tier. Second, we use CWNs to link these EUCs to concrete implementations.

Today, workflow technology is used in areas such as radiology [26]. However, there is no systematic and broader support for workflows in health-care organizations. Vendors and researchers are trying to implement “careflow systems” but are often confronted with the need for more flexibility [22, 23]. The state concept in CPN and YAWL (e.g., places with multiple outgoing arcs modeling a choice which is resolved by the organization rather than the system) allows for more flexibility than classical workflow systems. We know of one other application of YAWL in the health-care domain. Giorgio Leonardi, Silvana Quaglini et al. from the University of Pavia have used YAWL to build a careflow management system for outpatients. However, they did not use task descriptions, EUCs, and CWNs. Instead they directly implemented the system in YAWL.

7 Conclusions

In this paper, we realized a small careflow system using the four-step approach depicted in Figure 1 and motivated the added value of each of the three transformation steps in our approach. Obviously, the system made using YAWL is not the full EPR for Fyn County. It is just a prototype illustrating the viability of our approach. To come from an extensive and detailed set of task descriptions — as the seven task descriptions we have been considering — to their implementation requires large amounts of work and extensive involvements of the stakeholders. A weaknesses of the work presented in this paper is the unavailability of stakeholders in coming from the task descriptions to the EUC, the CWN, and the YAWL implementation (stakeholders have been extensively involved in the writings of the task descriptions, but this is beyond the scope of this paper).

The language we used both for Executable Use Cases (EUCs) and Colored Workflow Nets (CWNs) is CPN. For the actual realization of the system we used YAWL which can be seen as a superset of CPNs (extended with OR-joins and cancellation sets [1]) dedicated towards the implementation of workflows. The state-based nature of these modeling languages fits well with task descriptions, i.e., in a given state it is possible to enable multiple tasks and let the environment select one of these tasks. This is not possible in many workflow systems because there the system selects the next step to be executed.

Although CPNs and YAWL allow for more flexibility than classical workflow management systems, we would like to argue that in the health-care domain more flexibility is needed than what is provided by YAWL as it has been used in this paper. Work on computer-interpretable guidelines [21] shows that classical workflow languages tend to be too restrictive. Health-care workers should be allowed to deviate and select alternative pathways if needed.

To conclude this paper, we would like to discuss three extensions to allow for more flexibility.

- **Dynamic change.** The basic idea of dynamic change is to allow for changes while cases are being handled [24]. A change may affect one case (e.g., changing the standard treatment for an individual patient) or many cases (e.g., a new virus forcing a hospital to deviate from standard procedures). Although this approach is very flexible, it requires end-users to be able and willing to change process models.
- **Case handling.** Case handling [5] comprises a set of concepts to enable more flexibility without the need for adapting processes. The basic idea is that there are several mechanisms to deviate from the standard flow, e.g., unless explicitly disabled people can skip and roll-back tasks. Moreover, the control-flow perspective is no longer dominating, i.e., based on the available data the state is constantly re-evaluated and the collection and visualization of data is no longer bound to specific tasks.
- **Worklets.** Worklets [6] allow for the late binding of process fragments, e.g., based on the condition of a patient the appropriate treatment is selected. YAWL supports the uses of worklets, i.e., based on ripple-down rules an

appropriate subprocess is selected. The set of ripple-down rules and the repertoire of worklets can be extended on-the-fly thus allowing for a limited form or dynamic change.

Each of these approaches can be combined with the four-step approach depicted in Figure 1. However, further work is needed to develop EUCs and CWNs that can capture the degree of required flexibility and link this to concrete workflow languages allowing for more flexibility. Currently, even the most innovative systems support only one form of flexibility. For example, Adept [24] only supports dynamic change, FLOWer [5] only supports case handling, and YAWL [6] only supports worklets. Hence, future work will aim at an analysis of the various forms of flexibility in the context of the approach presented in this paper.

Acknowledgements We thank Søren Lauesen for permission to use the task descriptions for the Fyn County EPR as basis for this paper. We also thank Søren for fruitful discussions and feedback on this paper.

References

1. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
3. W.M.P. van der Aalst, J.B. Jørgensen, and K.B. Lassen. Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System. In *Proc. of 13th International Cooperative Information Systems Conf.*, volume 3760 of *LNCS*, pages 22–39, Agia Napa, Cyprus, 2005. Springer.
4. W.M.P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.
5. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
6. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating Flexibility and Dynamic Exception Handling in Workflows. In O. Belo, J. Eder, O. Pastor, and J. Falcao e Cunha, editors, *Proceedings of the CAiSE'05 Forum*, pages 45–50. FEUP, Porto, Portugal, 2005.
7. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
8. C. Bossen and J.B. Jørgensen. Context-descriptive Prototypes and Their Application to Medicine Administration. In *Proc. of Designing Interactive Systems (DIS) 2004*, pages 297–306, Cambridge, Massachusetts, 2004. ACM Press.
9. A. Cockburn. *Writing Effective Use Cases*. Addison-Wesley, 2000.
10. Krav til Fyns Amts EPJ-system (udkast) — Requirements to Fyn County's EPR System (Draft). Fyns Amt, 2003.

11. P. Grünbacher, A. Egyed, and N. Medvidovic. Reconciling software requirements and architectures with intermediate models. *Software and Systems Modeling*, 3(3):235–253, 2004. Springer.
12. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.
13. M. Jackson. *Problem Frames — Analyzing and Structuring Software Development Problems.* Addison-Wesley, 2001.
14. K. Jensen. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts.* Monographs in Theoretical Computer Science. An EATCS Series. Springer, 1992.
15. J.B. Jørgensen and C. Bossen. Executable Use Cases as Links Between Application Domain Requirements and Machine Specifications. In *Proc. of 3rd International Workshop on Scenarios and State Machines (at ICSE 2004)*, pages 8–13, Edinburgh, Scotland, 2004. IEE.
16. J.B. Jørgensen and C. Bossen. Executable Use Cases: Requirements for a Pervasive Health Care System. *IEEE Software*, 21(2):34–41, 2004.
17. 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.
18. S. Lauesen. *Software Requirements — Styles and Techniques.* Addison-Wesley, 2002.
19. S. Lauesen. Task Descriptions as Functional Requirements. *IEEE Software*, 20(2):58–65, 2003.
20. OMG Unified Modeling Language Specification, Version 1.4. Object Management Group (OMG); UML Revision Taskforce, 2001.
21. M. Peleg and et al. Comparing Computer-interpretable Guideline Models: A Case-study Approach. *Journal of the American Medical Informatics Association*, 10(1):52–68, 2003.
22. S. Quaglini, M. Stefanelli, A. Cavallini, G. Micieli, C. Fassino, and C. Mossa. Guideline-based Careflow Systems. *Artificial Intelligence in Medicine*, 20(1):5–22, 2000.
23. S. Quaglini, M. Stefanelli, G. Lanzola, V. Caporusso, and S. Panzarasa. Flexible Guideline-based Patient Careflow Systems. *Artificial Intelligence in Medicine*, 22(1):65–80, 2001.
24. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
25. I. Sommerville. *Software Engineering — Seventh Edition.* Addison-Wesley, 2004.
26. T. Wendler, K. Meetz, and J Schmidt. Workflow Automation in Radiology. In H.U. Lemke, editor, *Proceedings of Computer Assisted Radiology and Surgery (CAR98)*, pages 364–369. Elsevier, 1998.
27. CPN Tools. www.daimi.au.dk/CPNTools.

A Coloured Petri Net Model of the Dynamic MANET On-demand Routing Protocol

Cong Yuan and Jonathan Billington

Computer Systems Engineering Centre
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
Email: yuacy003@students.unisa.edu.au, jonathan.billington@unisa.edu.au

Abstract. An important characteristic of Mobile Ad Hoc Networks (MANETs) is rapidly changing topologies due to the movement of wireless mobile devices that we shall refer to as nodes. Traditional routing approaches cannot be used directly in such environments. Thus a number of experimental routing protocols have been designed to suit the new conditions. One of these is the Dynamic MANET On-demand (DYMO) routing protocol being developed by the Internet Engineering Task Force. Compared to other on-demand routing protocols, it only requires the most basic route discovery and maintenance procedures. To determine whether there are any subtle errors or design flaws, we create a Coloured Petri Net (CPN) model of DYMO, as the first step towards its verification. The model utilizes a common net structure, with functions in arc expressions representing the routing algorithms. This common CPN pattern can be used to model other on-demand routing protocols for MANETs. We do not attempt to model the changing topology of the network directly, but instead model the node routing tables. Our experiments show that this model can simulate the required routing operations defined in DYMO, as well as the dynamically changing topologies of a MANET.

Keywords: DYMO, Coloured Petri Nets, MANETs, modelling and simulation, verification.

1 Introduction

A Mobile Ad Hoc Network (MANET) [1,8] is an autonomous system of mobile nodes. It does not use any preexisting infrastructure and there is no centralized administration. The network topology may change rapidly at unpredictable times, due to the arbitrary movement of nodes. The purpose of the Internet Engineering Task Force (IETF) MANET Working Group (WG) [11] is to standardize IP routing protocols that are suited to wireless ad hoc environments. Several routing protocols [7, 14, 23, 25] have been developed by this WG, the latest being the Dynamic MANET On-demand (DYMO) routing protocol [5], a new reactive routing protocol. In reactive routing protocols, routes are only discovered when they are needed (i.e. *on-demand*), a very different approach to that used by conventional fixed network routing protocols. DYMO only requires the most basic route discovery and maintenance procedures, based on a simplified combination of two previous reactive routing protocol procedures [14,25]. Three implementations of DYMO [15,20,30] have been announced [10], and some preliminary results given in [12]. To determine whether there are subtle errors or design flaws, formal verification of the operations of the protocol is necessary [32]. This paper presents a first attempt in this direction by formalising the dynamic operations of DYMO using Coloured Petri Nets (CPNs) [13,16].

In previous work [37], we provided an abstract CPN model of a proactive protocol, known as the Destination-Sequenced Distance-Vector Routing (DSDV) protocol [26]. Proactive protocols attempt to establish and maintain routes as soon as they are activated in the MANET, rather than finding routes on demand. Thus the routing mechanisms of DSDV and DYMO are very different. We discussed the literature related to modelling highly dynamic systems using CPNs in [37]. Briefly, Findlow and Billington [9] used High-Level Petri Nets [3] to model dynamic dining philosophers, which only considers a circular topology. Xiong et al. [33, 34] have presented a timed CPN model for the Ad-hoc On-demand Distance Vector (AODV) routing protocol [24]. Due to the perceived difficulty of modelling arbitrary changing topologies, Xiong et al. only consider an approximation. Kristensen et al [18] have used hierarchical CPNs to model MANETs. The network topology is directly modelled, but no routing protocol is considered. Then in [17], the Edge Router Discovery Protocol is modelled and analysed. Similarly, [19] models the interoperability protocol, by which packets flow between a core network and an ad hoc network via the closest gateway connecting the two networks. The dynamically changing topologies of the entire network are not considered. In contrast, our work focuses on modelling the routing protocol in detail, and expressing the dynamically changing topology via updating routing table entries. Each node updates its routing table by creating, updating or invalidating the route entries, after receiving the DYMO control messages transmitted in the system. Thus, the route entries, contained in the node routing tables, record the current connection between nodes, which reflects the topology of the network learned from the view of nodes, as is done in a real MANET.

The main aim of this paper is to extend the modelling approach developed in [36], by providing a CPN model of a highly dynamic MANET, with nodes running DYMO. We employ a generic net structure suited to modelling reactive protocols in general. The specifics of the DYMO routing algorithms are then realised by functions in the arc expressions of the model. We also consider some optimization operations of DYMO, such as an originator node using an *expanding ring search* (see Section 2) to search for routes to the destination. Finally, we also perform some simulation experiments to demonstrate the way the model captures the dynamic changes in network topology.

There are several contributions of this paper. Firstly, following on from our previous work [36, 37], we model the dynamically changing topologies of MANETs running reactive routing protocols including message loss and retransmission. Secondly, we provide the first formal specification of the DYMO routing procedures. Thirdly, we discover incorrect routing information in the route discovery procedure due to ambiguous statements in the specification [5] by simulation experiments. Finally, we suggest modifications to the DYMO procedures in an attempt to overcome this problem.

The rest of the paper is organised as follows. Section 2 describes the basic operations of DYMO. In Section 3, we present our CPN model of DYMO. Simulation experiments are conducted in Section 4 and discussed in Section 5. Finally, conclusions and future work are presented in Section 6.

2 Dynamic MANET On-demand Routing Protocol

In this section, we present the background of DYMO in the field of routing algorithms. Then the basic operations of DYMO are described by a simple example that illustrates the route discovery and maintenance procedures.

2.1 Background

Distance vector routing algorithms [22] are based on the shortest path computation algorithm presented by Bellman [2]. The Routing Information Protocol (RIP) [21] is a simple and practical distance vector protocol. However, RIP cannot be used for MANETs directly, because it lacks the ability to handle rapid changes in topology [27], and it also suffers from very slow convergence (the *counting to infinity* problem [31]). For these reasons, DSDV [26] and then AODV [28, 29] were developed for ad hoc networks. These protocols preserve the distance vector routing algorithms, and utilise sequence numbers to ensure freedom from looping behaviour. To overcome the complexity in AODV, IETF started to develop a new reactive routing protocol, known as the Dynamic MANET On-demand Routing Protocol (DYMO) [4] in 2005. A brief introduction to the basic routing algorithms of DYMO is given in the next subsection (according to [5]) in order to understand the CPN model in Section 3.

2.2 Protocol Overview

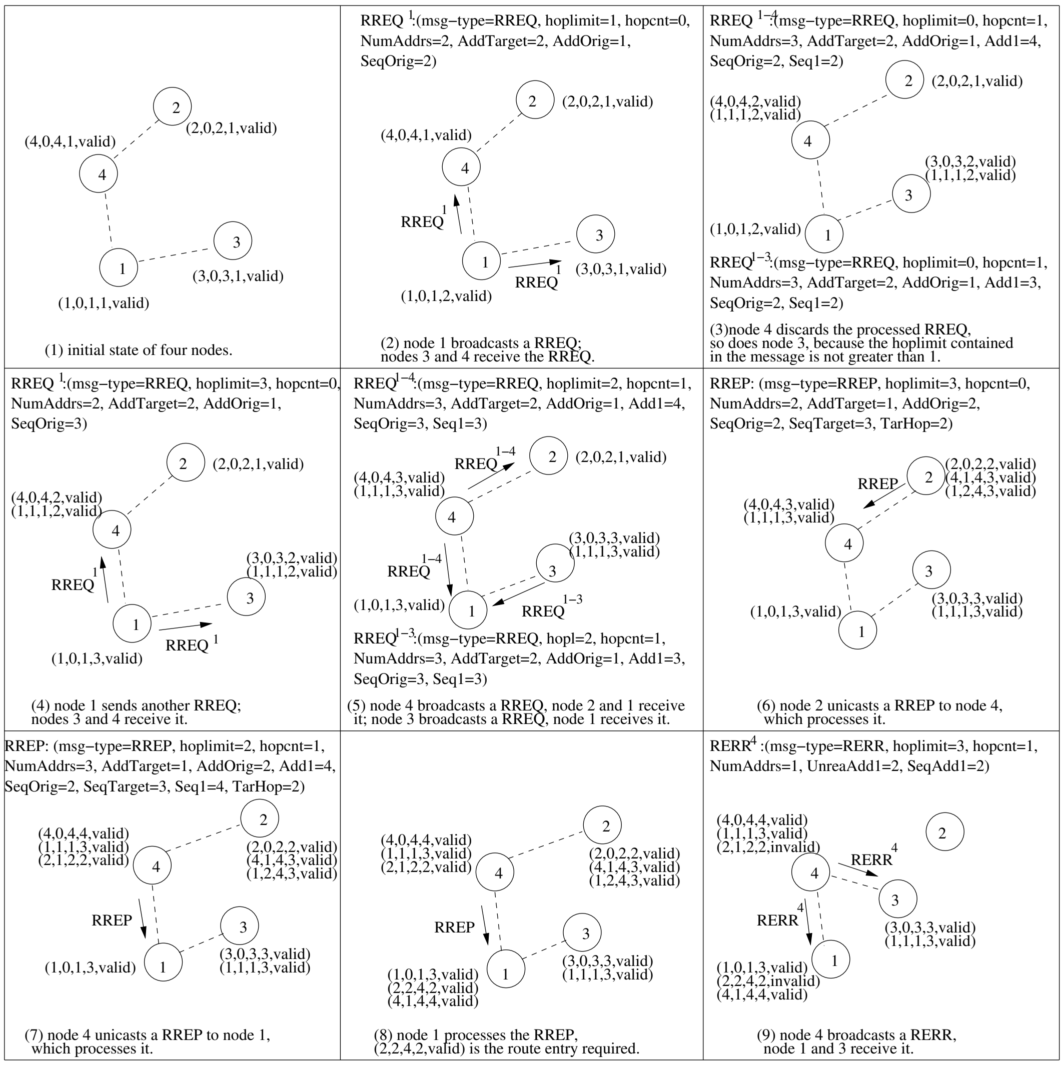
In reactive routing protocols, routes are only attempted to be found when a data packet arrives at a node and its destination address is not present in the node's routing table (or the routing table entry for that address is no longer valid). In this situation, DYMO uses a procedure for route discovery comprising broadcasting a route request message and (hopefully) receiving a reply containing a discovered path. Each route entry in the routing table contains the following fields:

- **Route.Address**: the IP address of the destination node.
- **Route.SeqNum**: the last known sequence number of the destination.
- **Route.NextHopAddress**: the IP address of the next node on the path toward the destination.
- **Route.NextHopInterface**: the particular interface over which the packets are sent toward the destination must be known, because it is likely that DYMO will be used with multiple wireless interfaces.
- **Route.ValidTimeout**: the time at which the route entry is no longer considered valid.
- **Route.DeleteTimeout**: the time after which the route entry must be deleted, so that the invalid route entries are purged from the routing table.

In addition, there are some optional fields defined. *Route.HopCnt* contains the number of intermediate node hops traversed to reach the destination. *Route.IsInternetGateway* indicates whether the destination is an Internet gateway. *Route.Prefix* indicates the destination is a network address, rather than a host address. If prefix is set to zero, unkown, or equal to the address length in bits, this destination has a host address. *Route.Used* indicates whether this route entry has been used to forward data toward the destination.

Consider 4 nodes running DYMO in a MANET from the initial state as time increases, as shown in Fig. 1. Each circle represents a node with its identity inside the circle, the solid arrow indicates a DYMO message sent by a node with its type next to the arrow, and the dash line is used to connect two nodes which can communicate with each other. This example provides an introduction to the DYMO routing mechanisms defined in [5]. The route entry described above is represented as a 7-tuple in the model (see Section 3.1). Here to save the space, two zero fields *Route.prefix* and *Route.IsGateway* are omitted. Thus, each route entry shown in Fig. 1 is described by a 5-tuple: (*Route.Address, Route.HopCnt, Route.NextHopAddress, Route.SeqNum, Route.REstate*), where Route.Address indicates the identity of the destination, Route.HopCnt represents the number of hops traversed to reach this destination, Route.NextHopAddress indicates the identity of the next node towards the destination, Route.SeqNum is the last known sequence number of the destination or *zero* if this is unknown, and Route.REstate is set to *invalid* if the Route.ValidTimeout is exceeded and *valid* otherwise.

Moreover, to present a complete picture of the system, we assume the initial state of the MANET in Fig. 1(1) is that: for each node, there is one route entry to itself, in which the destination and the next hop are itself, the number of hops is assigned as 0, and its initial sequence number is assumed to be 1 (since the sequence number 0 is reserved and is used to represent an unknown sequence number).



(4,0,4,1,valid)
(2,0,2,1,valid)
(3,0,3,1,valid)
(1,0,1,1,valid)
(1) initial state of four nodes.
RREQ 1:(msg−type=RREQ, hoplimit=1, hopcnt=0, NumAddrs=2, AddTarget=2, AddOrig=1, SeqOrig=2)
(2,0,2,1,valid)
(4,0,4,1,valid)
RREQ 1
RREQ 1
(3,0,3,1,valid)
(1,0,1,2,valid)
(2) node 1 broadcasts a RREQ; nodes 3 and 4 receive the RREQ.
RREQ 1−4:(msg−type=RREQ, hoplimit=0, hopcnt=1, NumAddrs=3, AddTarget=2, AddOrig=1, Add1=4, SeqOrig=2, Seq1=2)
(2,0,2,1,valid)
(4,0,4,2,valid)
(1,1,1,2,valid)
(3,0,3,2,valid)
(1,1,1,2,valid)
(1,0,1,2,valid)
RREQ1−3:(msg−type=RREQ, hoplimit=0, hopcnt=1, NumAddrs=3, AddTarget=2, AddOrig=1, Add1=3, SeqOrig=2, Seq1=2)
(3)node 4 discards the processed RREQ, so does node 3, because the hoplimit contained in the message is not greater than 1.
RREQ 1:(msg−type=RREQ, hoplimit=3, hopcnt=0, NumAddrs=2, AddTarget=2, AddOrig=1, SeqOrig=3)
(2,0,2,1,valid)
(4,0,4,2,valid)
(1,1,1,2,valid)
(3,0,3,2,valid)
(1,1,1,2,valid)
RREQ 1
RREQ 1
(1,0,1,3,valid)
(4) node 1 sends another RREQ; nodes 3 and 4 receive it.
RREQ1−4:(msg−type=RREQ, hoplimit=2, hopcnt=1, NumAddrs=3, AddTarget=2, AddOrig=1, Add1=4, SeqOrig=3, Seq1=3)
RREQ 1−4
(2,0,2,1,valid)
(4,0,4,3,valid)
(1,1,1,3,valid)
(3,0,3,3,valid)
(1,1,1,3,valid)
RREQ 1−4
(1,0,1,3,valid)
RREQ 1−3
RREQ1−3:(msg−type=RREQ, hopl=2, hopcnt=1, NumAddrs=3, AddTarget=2, AddOrig=1, Add1=3, SeqOrig=3, Seq1=3)
(5) node 4 broadcasts a RREQ, node 2 and 1 receive it; node 3 broadcasts a RREQ, node 1 receives it.
RREP: (msg−type=RREP, hoplimit=3, hopcnt=0, NumAddrs=2, AddTarget=1, AddOrig=2, SeqOrig=2, SeqTarget=3, TarHop=2)
RREP
(2,0,2,2,valid)
(4,1,4,3,valid)
(1,2,4,3,valid)
(4,0,4,3,valid)
(1,1,1,3,valid)
(1,0,1,3,valid)
(3,0,3,3,valid)
(1,1,1,3,valid)
(6) node 2 unicasts a RREP to node 4, which processes it.
RREP: (msg−type=RREP, hoplimit=2, hopcnt=1, NumAddrs=3, AddTarget=1, AddOrig=2, Add1=4, SeqOrig=2, SeqTarget=3, Seq1=4, TarHop=2)
(4,0,4,4,valid)
(1,1,1,3,valid)
(2,1,2,2,valid)
(2,0,2,2,valid)
(4,1,4,3,valid)
(1,2,4,3,valid)
RREP
(1,0,1,3,valid)
(3,0,3,3,valid)
(1,1,1,3,valid)
(7) node 4 unicasts a RREP to node 1, which processes it.
(4,0,4,4,valid)
(1,1,1,3,valid)
(2,1,2,2,valid)
(2,0,2,2,valid)
(4,1,4,3,valid)
(1,2,4,3,valid)
RREP
(1,0,1,3,valid)
(2,2,4,2,valid)
(4,1,4,4,valid)
(3,0,3,3,valid)
(1,1,1,3,valid)
(8) node 1 processes the RREP, (2,2,4,2,valid) is the route entry required.
RERR4 :(msg−type=RERR, hoplimit=3, hopcnt=1, NumAddrs=1, UnreaAdd1=2, SeqAdd1=2)
(4,0,4,4,valid)
(1,1,1,3,valid)
(2,1,2,2,invalid)
RERR 4
RERR 4
(3,0,3,3,valid)
(1,1,1,3,valid)
(1,0,1,3,valid)
(2,2,4,2,invalid)
(4,1,4,4,valid)
(9) node 4 broadcasts a RERR, node 1 and 3 receive it.

Fig. 1. A simple example of a route discovery and maintenance procedure of DYMO

Assume that node 1 needs a route for node 2. It broadcasts a route request (RREQ) message. The RREQ message format [5] is given in Fig. 2(1). A RREQ message contains an *IP Header*, a *Message Header*, an *Address Block* and a corresponding *Address tlv-block* (i.e. "Address TLVs" in Fig. 2(1)) [6]. The IP Header of a RREQ contains the IP source address (not shown in Fig. 2(1)), and the IP destination address (*IP.DestinationAddress*), which is set

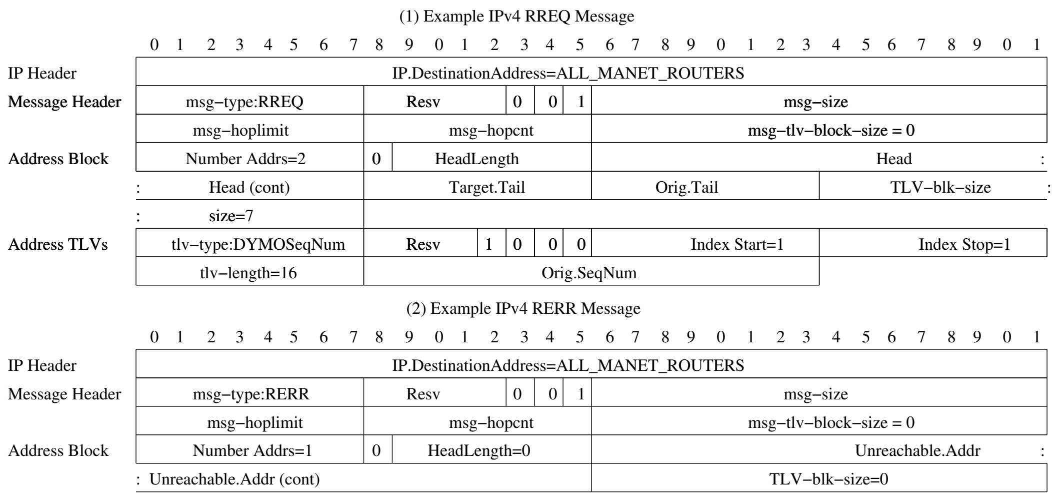


Fig. 2. DYMO message formats

to *ALL_MANET_ROUTERS*, indicating that a RREQ is broadcast in the MANET. In the Message Header, the *msg-type* field is set to specify a RREQ message; the *msg-semantics* field is defined as $(Resv, 0, 0, 1)$, which indicates that *msg-hoplimit* and *msg-hopcnt* are included in this message; and *msg-size* specifies the size of the entire message counted in octets (each 8 bits). In the second line of the Message Header, the msg-hoplimit represents the maximum number of hops over which a message will be transmitted; the msg-hopcnt indicates the number of hops a message has traversed. The *msg-tlv-block-size* field is set to 0, indicating that a DYMO message does not contain the message tlv-block (we describe the tlv-block later). Next, lines 4 to 6 of Fig. 2(1) is an *Address Block*, which is a block of addresses about which the originator of the message wishes to convey information. The *Number Addrs* indicates the number of addresses contained in the message. In DYMO messages, addresses contained in an address block share the same *Head* and have different *Tails*. Thus, the *HeadLength* represents the length of the following *Head* field. When concatenated to the Head, *Target.Tail* represents the IP address of the Target destination node indicated in RREQ, *Orig.Tail* represents the IP address of the originator of the RREQ. *TLV-blk-size* in the address block indicates the total length (in octets) of the immediately following tlv(s), namely Type-Length-Value (TLV). An Address TLV specifies some attribute(s), which are associated with address(es) contained in the preceding address block. As shown in Fig. 2(1), the *tlv-type* is set to *DYMOSeqNum*. This means that this address tlv contains the sequence number of some address(es) if known. The *tlv-semantics* is defined as $(Resv, 1, 0, 0, 0)$, indicating the elements included in the tlv block are: *index-start*, *index-stop*, *tlv-length* and *value*. The index-start and index-stop are used to specify the addresses whose sequence numbers are contained in this tlv. The tlv-length specifies the size of each sequence number contained, and value indicate these sequence numbers. In Fig. 2(1), the index-start and index-stop are both set to 1, indicating the second address *Orig.Tail* in preceding address block, where the first address *Target.Tail* is indicated by zero. Hence, the sequence number of the originator node *Orig.SeqNum*, with the size defined as tlv-length, is included.

As shown in Fig. 1(2), node 1 broadcasts a RREQ message (represented as $RREQ^1$) and then waits for a reply. At this time, only nodes 3 and 4 are in range of node 1 and receive $RREQ^1$. To introduce the routing procedures simply, the message only contains main information without considering its concrete data structure, and elements concerned with *size* or *length* are ignored. The $RREQ^1$ is shown in Fig. 1(2). The type of the message is set to *RREQ*, the *hopcnt* is set to 0, and the initial *hoplimit* is set to 1, according to the *expanding ring search* described in AODV [25], which is to prevent unnecessary network-wide dissemination of RREQs. Node 1 increments its sequence number by one, and sets the originator address (*AddOrig*) and sequence number (*SeqOrig*) to its own value. Because the sequence number of node 2 (*AddTarget*) is unknown, it is not included in the message.

In Fig. 1(3), nodes 3 and 4 process the received RREQ, where the hoplimit is decremented by one and the hopcnt is incremented by one. Then each node creates a route entry to node 1. To facilitate the future route discovery, it also appends its IP address (*Add1*) and incremented sequence number (*Seq1*) to the processed RREQ. The RREQ processed by node 4 is represented as $RREQ^{1-4}$, that by node 3 as $RREQ^{1-3}$. Because the hoplimit of $RREQ^{1-4}$ and $RREQ^{1-3}$ is not greater than one, nodes 4 and 3 discard the processed RREQs. Their updated routing tables are shown in Fig. 1(3).

As shown in Fig. 1(4), node 1 does not receive the reply within its timeout period, so it broadcasts another RREQ (as $RREQ^1$). The hoplimit of this RREQ is incremented to 3 (the suggested value in AODV). To distinguish this RREQ and the older one, we assume that node 1 increases the originator sequence number (this point is not specified in [5] and we discuss it in Section 5). Nodes 3 and 4 still receive this RREQ.

Then in Fig. 1(5), they process the received RREQ. They update their routing tables and then broadcast the processed RREQ because the hoplimit is larger than one. Nodes 1 and 2 receive the $RREQ^{1-4}$ sent by node 4, and node 1 also receives the $RREQ^{1-3}$ sent by node 3. Because node 1 is the node indicated as the originator contained in the received RREQs, it drops them. In contrast, node 2 processes the received RREQ.

As shown in Fig. 1(6), node 2 creates route entries to nodes 1 and 4. Then it generates a route reply (RREP) and discards the processed RREQ, since it is the node indicated as the target in the RREQ. According to DYMO, Node 2 increases its own sequence number before creating the RREP, because the target sequence number is not included in the received RREQ. The RREP message format is the same as that of RREQ given in Fig. 2(1), except that its IP destination address only contains one address, which is the next hop for the target of the RREP. Thus the RREP is unicast in the MANET, rather than broadcast. In the created RREP, the type of the message is *RREP*, the *hoplimit* is set to *Net-Diameter* (assumed as 3 here) and the *hopcnt* is set to 0. Node 2 as the originator inserts its own address and sequence number in *AddOrig* and *SeqOrig* respectively. The address and sequence number of node 1 contained in the received RREQ are set to *AddTarget* and *SeqTarget*, as the routing information of the target. The *TarHop* is set to the hop count for node 1, as 2 hops. Then node 2 unicasts the RREP to node 4, which is the next hop of the route entry for node 1. Node 4 receives this RREP and processes it, where the hoplimit is decremented by one and hopcnt is incremented by one.

As shown in Fig. 1(7), node 4 inserts a route entry to node 2 in its routing table, and appends its routing information, *Add1* and *Seq1*, to the RREP after incrementing its own sequence number. Then the RREP is unicast to node 1.

Node 1 processes the received RREP. The updated routing table of node 1 is presented in Fig. 1(8), in which $(2, 2, 4, 2, valid)$ is the route entry required to be able to deliver data packets to node 2. Meanwhile, node 1 discards the processed RREP, because it is the node indicated as the target of the message.

Now, assume that when receiving data packets from node 1, node 4 cannot deliver them to node 2 because of a broken link. Node 4 generates a route error (RERR) message, as shown in Fig. 1(9). The RERR message format is given in Fig. 2(2). The IP header and the message header of a RERR message are same as those of a RREQ. There is only one unreachable destination address (*Unreachable.Addr*) contained in a RERR, and no tlv(s)) is present in the message, so HeadLength is 0, *msg-tlv-blk-size* and *TLV-blk-size* both are set to 0. In the RERR created by node 4 (as $RERR^4$), the hoplimit is set to Net-diameter and the hopcnt is set to 1. The unreachable destination is node 2 and its last known sequence number is 2 kept by node 4, so two elements are written as *UnreaAdd1* and *SeqAdd1* respectively. Then the RERR is broadcast in the MANET and nodes 1 and 3 receive it (node 3 has moved into range). Node 1 processes the unreachable node address contained in the RERR by marking the affected route entry in its routing table as invalid. In contrast, node 3 drops the received RERR, because no route entry in its routing table is affected by this unreachable node address.

3 CPN model of reactive routing

In this section we firstly explain the abstract representation of DYMO elements in the model. Then, we describe all assumptions used in our design of the model and the reasons they are necessary. We then present the limitations of the modelling in this model. Lastly, the CPN model is introduced in detail, and we explain the complicated operations run by this model, even though it may seem deceptively simple.

3.1 Assumptions

The purpose of the CPN model is not to implement but to specify the protocol, so the route entry Route.NextHopInterface is ignored. To simplify the problem, assume that each node is not an Internet gateway, and it is a host address, so Route.IsGateway and Route.Prefix are both set to zero in its routing table. Furthermore, we do not model the deletion of invalid route entries, so Route.DeleteTimeout is omitted. Any valid route entry can be used to deliver the data packets, so Route.Used is abstracted away. Thus each route entry in the model is described by a 7-tuple: (Route.Address, Route.HopCnt, Route.IsGateway, Route.NextHopAddress, Route.Prefix, Route.SeqNum, Route.REstate).

Moreover, we represent the DYMO message formats in an abstract way. The elements related to size or length and the message semantics are omitted in the messages used in our model. The TLV is a very flexible and extensible information carrier, by which information such as the hopcount, prefix or gateway may be included in a message or not. To cover the most general case, we keep all this routing information for each address contained in a message. To simplify computation, we do not use the defined formats, like the address block and address tlv-block, instead we list each address and its routing information together. Similarly, the target is stored in the last address rather than the first one (according to DYMO), since each address contained in a message, except the target, should be processed during the transmission.

In the model, we define some additional parameters which are not included in the protocol and are only relevant to the model itself. For instance, to realise the retransmission of a route request, we assume that each node keeps a pair, including a target IP address for which the node has generated a RREQ and is waiting for a reply, and a counter of the number of times this RREQ has been transmitted. Initially, these two elements both are set to zero, which means that the nodes do not generate a route discovery procedure. When a node creates a route discovery, it inserts the IP address of the target requested in the first field and sets the counter to one. If it does not receive a reply within a certain predefined period, it broadcasts another RREQ for this target and increments the counter by one. This can continue, up to the maximum number of retries permitted in the system. If the node receives a reply for this target, it resets both fields to zero again. Hence, the zero IP address is only used in this case of the model. A node with a target IP address other than zero is regarded as a waiting originator node in the model, and can retransmit a RREQ again.

When designing the model, we found some statements given in DYMO [5] are not complete and explicit, so we made assumptions to make them more reasonable. For example, no initial state is presented in DYMO. As described in Section 2.2, we assume that each node knows nothing about the environment initially, and only has a route entry to itself in its routing table, where the hop count to itself is set to zero. This does not conform to DYMO, where the hop count is only assumed to be zero if unknown. However, we believe the hop count to a node itself should be included, to ensure routing operations defined in DYMO work correctly. To cover this difference, an unknown hop count is represented as the value *unknown* directly in our model. Similarly, we assume that a node increases its sequence number before rebroadcasting the RREQ (these two points will be discussed in Section 5). In addition, we assume that sequence numbers are always increased by one in the model, although this is not explicitly defined in DYMO.

We do not model the broadcast completely in the model. In a MANET, the broadcast of a node may be received by: no nodes (the node is isolated); one node (only one node can communicate with this node); or any number of nodes. In the model of Fig. 3, a broadcast message may be lost, indicating no nodes receive it; or may be received by an arbitrary node. Thus, this receiving node may be the only receiver, or one of several receivers. Then, the node transmits the message after processing if it is not the target. In the case of a route discovery, this procedure continues until a RREP is generated. Next, the RREP is unicast along the nodes which have transmitted the RREQ we modelled, until it reaches the originator node. In this way, we just consider the bidirectional dissemination of DYMO control messages along a group of nodes, each consisting of one hop in the route found by this route discovery procedure. Meanwhile, we do not model operations of other receivers of a broadcast, as we interpret that they are not related to the route found, although they may update route entries in their routing tables which form their knowledge of the topology of the network. Thus, the current model is sufficient for observing operations of the protocol, but it does not model a complete broadcast in the system. In addition, we do not model data packets waiting for a route. If the route entry required is found, the successful transmission of the data packets is not considered. However, we do model the route maintenance procedure.

3.2 CPN model

Based on these assumptions, we create the CPN diagram of DYMO given in Fig. 3. The lower part models route discovery, and the upper part models route maintenance.

The nodes are stored in the place Nodes, which is typed by the colour set Mnode as shown in line 22 in Fig. 4. Mnode is a product of NodeId, RT and RREQTries. We represent NodeId (see line 8 in Fig. 4) as an integer from zero to the maximum number of nodes in the MANET, MaxNodes (see line 1 in Fig. 4). The zero is reserved and only used in RREQTries. RT (see line 19 in Fig. 4) is a list of route entries, one for each destination. RTEntry, indicating a route entry, is a 7-tuple comprising DestAddress, HopCnt, IsGateway, NextHopAddress, Prefix, SeqNum and REstate. The DestAddress and NextHopAddress are node addresses, represented

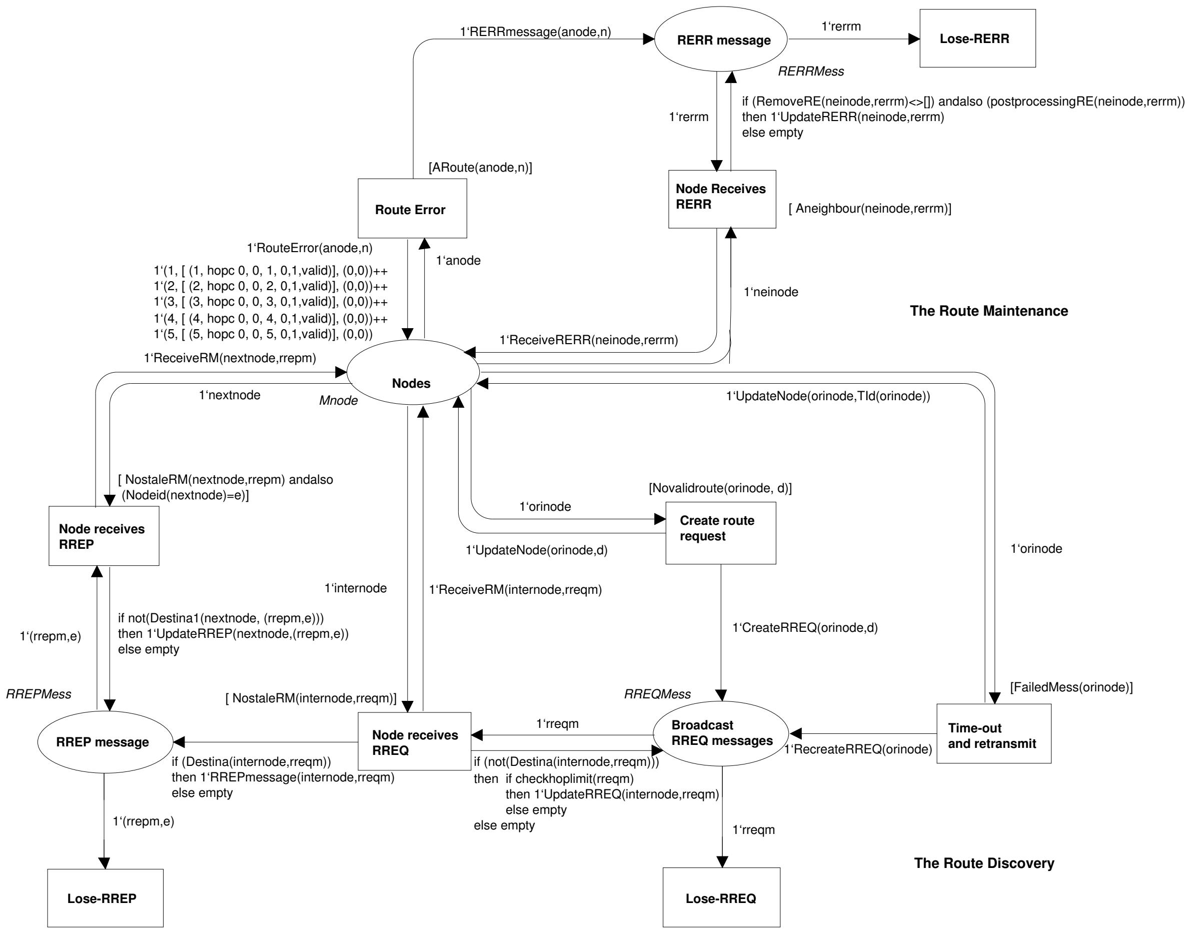

1'RERRmessage(anode,n)
RERR message
1'rerrm
Lose-RERR
RERRMess
1'rerrm
if (RemoveRE(neinode,rerrm)<>[]) andalso (postprocessingRE(neinode,rerrm))
then 1'UpdateRERR(neinode,rerrm)
else empty
[ARoute(anode,n)]
Route Error
Node Receives RERR
[Aneighbour(neinode,rerrm)]
1'RouteError(anode,n)
1'anode
1'(1, [(1, hopc 0, 0, 1, 0,1,valid)], (0,0))++
1'(2, [(2, hopc 0, 0, 2, 0,1,valid)], (0,0))++
1'(3, [(3, hopc 0, 0, 3, 0,1,valid)], (0,0))++
1'(4, [(4, hopc 0, 0, 4, 0,1,valid)], (0,0))++
1'(5, [(5, hopc 0, 0, 5, 0,1,valid)], (0,0))
1'neinode
The Route Maintenance
1'ReceiveRERR(neinode,rerrm)
1'ReceiveRM(nextnode,rrepm)
Nodes
1'nextnode
Mnode
1'UpdateNode(orinode,TId(orinode))
[NostaleRM(nextnode,rrepm) andalso (Nodeid(nextnode)=e)]
[Novalidroute(orinode, d)]
1'orinode
Create route request
Node receives RREP
1'UpdateNode(orinode,d)
1'orinode
1'internode
1'ReceiveRM(internode,rreqm)
if not(Destina1(nextnode, (rrepm,e)))
then 1'UpdateRREP(nextnode,(rrepm,e))
else empty
1'(rrepm,e)
1'CreateRREQ(orinode,d)
RREPMess
[NostaleRM(internode,rreqm)]
RREQMess
[FailedMess(orinode)]
1'rreqm
RREP message
Node receives RREQ
Broadcast RREQ messages
Time-out and retransmit
1'RecreateRREQ(orinode)
if (Destina(internode,rreqm))
then 1'RREPmessage(internode,rreqm)
else empty
if (not(Destina(internode,rreqm)))
then if checkhoplimit(rreqm)
then 1'UpdateRREQ(internode,rreqm)
else empty
else empty
1'(rrepm,e)
1'rreqm
The Route Discovery
Lose-RREP
Lose-RREQ

Fig. 3. CPN model of DYMO routing procedures

by the NodeId. IsGateway and Prefix, indicating whether the destination is an Internet gateway or a host respectively, are each represented by an integer. SeqNum (see line 15 in Fig. 4) is represented as an integer from zero to the largest possible number MaxSeqNr (see line 6 in Fig. 4) given by DYMO. HopCnt is normally the Hopcount, an integer from zero to MaxNodes, representing the number of intermediate node hops before reaching the destination. However, when it is unknown, this field is assigned the value *unknown*. Thus HopCnt is a union of the Hopcount and *unknown* as seen in line 11 in Fig. 4. In line 16, REstate is used to denote the state of a route entry. It may either be *valid* indicating that the route entry can be utilized to forward data packets, or *invalid* indicating that the route entry is no longer usable. RREQTries (see line 21 in Fig. 4) is a product set, including the target IP address (DestAddress) and the number of times of transmission (Tries), which is represented as an integer from zero to the maximum number of transmission, MaxTries (see line 4 in Fig. 4).

We consider a configuration of 5 nodes in the MANET for our initial experiments (see Fig. 3). Thus Nodes is initially marked by 5 nodes, where each routing table has a single entry to itself. Initially, the target address is set to zero, a value indicating that no target address has been allocated. We also set the number of tries to zero, indicating that at the beginning, no RREQs have been broadcast.

```
-----------------------------------------------------------------------
val MaxNodes = 5;
val TTLstart = 1;
val TTLincrement =2;
val MaxTries = 3;
val Netdiameter = 4;
val MaxSeqNr = 65535;
      (* each node: IP address, the routing table, retransmission of RREQ *)
color NodeId = int with 0..MaxNodes;
color DestAddress = NodeId;
color Hopcount = int with 0..MaxNodes;
color HopCnt = union hopc:Hopcount + unknown declare of_hopc;
color IsGateway = int;
color NextHopAddress = NodeId;
color Prefix = int;
color SeqNum = int with 0..MaxSeqNr;
color REstate= with valid | invalid;
color RTEntry = product DestAddress * HopCnt * IsGateway * NextHopAddress*
             Prefix * SeqNum * REstate;
color RT = list RTEntry;
color Tries = int with 0..MaxTries;
color RREQTries = product DestAddress * Tries;
color Mnode = product NodeId * RT * RREQTries;
      (*msg-header*)
color msgtype = with RREQ | RREP | RERR;
color msghoplimit = int with 1..Netdiameter;
color msghopcnt = Hopcount;
color msgheader= product msgtype * msghoplimit * msghopcnt;
      (*address-block and address-tlv-block *)
color Numadds = int;
color Add = NodeId;
color Seqadd = SeqNum;
color Hopadd = HopCnt;
color Gatewtlv = IsGateway;
color Prefixtlv = Prefix;
color Addtlv = product Add * Seqadd * Hopadd * Gatewtlv * Prefixtlv;
color Addtlvs= list Addtlv;
color TarAddtlv = Addtlv;
color Addinfo = product Numadds * Addtlvs * TarAddtlv;
      (* Generalize message format: RREQ/RREP  *)
color RMessage = product msgheader * Addinfo;
color SourceAdd = NodeId;
color NextAdd = NodeId;
color RREQMess = product SourceAdd * RMessage;
color RREPMess = product  RREQMess * NextAdd;
      (*RERR message*)
color UnreAddtlv = product Add * Seqadd * Hopadd;
color UnreAddtlvs = list UnreAddtlv;
color AddinfoRERR = product Numadds * UnreAddtlvs;
color RERRM = product msgheader * AddinfoRERR;
color RERRMess = product SourceAdd * RERRM;
      (* Variable Declarations *)
var n, m, e, d, h: NodeId;
var orinode, internode, nextnode, anode, neinode: Mnode;
var rreqm, rrepm : RREQMess;
var rerrm : RERRMess;
-----------------------------------------------------------------------
```

Fig. 4. Data types of the CPN model

As shown in line 27 in Fig. 4, msgheader indicates the message header of a DYMO message. It is a product set, including msgtype, msghoplimit and msghopcnt, where msgtype is the set of routing messages: RREQ, RREP and RERR; msghoplimit is an integer up to NetDiameter (see line 2 in Fig. 4), to prevent messages cycling in the MANET; and msghopcnt (Hopcount) represents the number of hops a message has traversed. As shown in line 38 in Fig. 4, Addinfo indicates the main body of a DYMO message. It is a product set, comprising Numadds, Addtlvs and TarAddtlv, where Numadds indicates the number of addresses contained in the message; Addtlvs contains a list of addresses contained in the message except the target and its routing information; TarAddtlv contains only the target address and its routing information. Addtlv (see line 35 in Fig. 4) records an address Add and its routing information known by the originator of the message.

As shown in line 40 in Fig. 4, RMessage represents a general DYMO message (i.e., RREQ or RREP), comprising msgheader and Addinfo. RREQMess (see line 43 in Fig. 4) forms the RREQ, comprising the IP source address (SourceAdd) and the message RMessage. The IP destination address of a RREQ is not set, because a RREQ is broadcast and the receiver(s) cannot to be predicted. The RREQ messages disseminated in the MANET are stored in the place, Broadcast RREQ messages, which is typed by the colour set RREQMess. In contrast, a RREP is unicast to the target (which is the originator node of the corresponding RREQ), so its IP destination address is the next hop of the route entry for the target in its routing table. Thus, RREPMess (see line 44 in Fig. 4) consists of RREQMess and its IP destination address (NextAdd). The place RREP message stores the RREPs transmitted in the system and is typed by the colour set RREPMess. RERR messages are stored in the place RERR message, which is typed by RERRMess, as shown in line 50 in Fig. 4. It is a product set of the IP source address (SourceAdd) of a RERR and the RERR message RERRM. RERRM comprises the message header (msgheader) and the message body (AddinfoRERR). AddinfoRERR (see line 48 in Fig. 4) contains the number of unreachable destinations (Numadds) and their addresses and routing information (UnreAddtlvs), which is a list of UnreAddtlv (see line 46 in Fig. 4).

Transition Create route request models the process of a node originating a RREQ. The variable, orinode, in the arc inscription from the place Nodes to this transition, represents an arbitrary node which originates the RREQ. The guard Novalidroute(orinode,d) ensures that orinode does not have a valid route entry for the destination d in its routing table. The value of d represents the identity of an arbitrary node except orinode. The arc inscription from this transition to the place Nodes is a function, UpdateNode(orinode). The function increments the sequence number of orinode by one, and inserts d as the target of the created RREQ and increments the times of retransmission counter of this RREQ by one. The arc expression from this transition to the place broadcast RREQ messages is the function CreateRREQ(orinode,d). The function returns the RREQ created by orinode that requires a route entry to d. We interpret the occurrence of transition Create route request to mean that at any time any node (orinode) can originate a route request, when it receives a data packet with a destination that does not have a valid entry in its routing table.

Transition Node receives RREQ models the process of a node handling a received RREQ. The arc expression from the place Nodes to this transition has a variable, internode, representing an arbitrary 'intermediate' node along the path to the target destination. The arc inscription from broadcast RREQ messages to this transition has a variable, rreqm, indicating the received RREQ. The guard NostaleRM(internode, rreqm) ensures that internode determines that the routing information of the originator, contained in rreqm is fresh. Otherwise, rreqm is dropped. If internode does not have a route entry for the originator, rreqm can be regarded as fresh. On the other hand, if it has such a route entry, it compares its routing information for the originator with that contained in rreqm. The RREQ is fresh if it contains a higher sequence number for the originator, or shorter hop count when the sequence numbers are identical. The arc inscription from this transition to the place Nodes is the function, ReceiveRM(internode, rreqm). This function returns internode after updating its routing table based on rreqm. As shown in the arc expression from this transition to the place broadcast RREQ messages, the function not(Destina(internode, rreqm)) ensures that internode is not the target desired by rreqm. The hoplimit contained in the RREQ is checked by the function checkhoplimit(rreqm). If the limit has not been reached, the processed RREQ returned by UpdateRREQ(internode, rreqm) is deposited in the place broadcast RREQ messages. Otherwise, if internode is the target, as shown in the arc expression from this transition to the place RREP message, it generates a RREP in response to the received RREQ using function RREPmessage(internode,rreqm). The created RREP is stored in the place RREP message. We interpret the occurrence of transition Node receives RREQ to mean that a broadcast RREQ, rreqm, can be received and processed by any node, internode, at any time. In contrast, transition Mess-loss models that no node receives a broadcast RREQ. In this case, the sender of the RREQ is regarded as an isolated node (no one can hear it).

Transition Time-out and retransmit models that a waiting originator orinode sends another RREQ, when its timer expires. The arc expression from the place Nodes to this transition is orinode. The guard FailedMess(orinode) ensures that orinode is a waiting originator node by checking that it has a non zero target address. It also makes sure that its retransmission of the same RREQ is less than MaxTries. In the arc inscription from this transition to broadcast RREQ messages, the RREQ returned by function RecreateRREQ(orinode) is deposited in the place broadcast RREQ messages. Similarly, as shown in the arc expression from this transition to the place Nodes, after orinode broadcasts a RREQ again, orinode is deposited in the place Nodes, with its sequence number and the number of retransmissions updated by function UpdateNode(orinode, Tid(orinode)), where Tid(orinode) returns the target currently kept by orinode. According to DYMO, the waiting time utilizes a binary exponential backoff from the initial time. In our model, the occurrence of this transition represents the current waiting time is over, regardless of its concrete value. Hence, our model is more general and covers any backoff scheme.

Transition Node receives RREP models the processing associated with receiving a RREP. The arc inscription nextnode from Nodes to this transition represents the next node in the path back to the originator of the route discovery (i.e. the target contained in the RREP). The arc expression (rrepm, e) from RREP message to this transition represents a RREP received by nextnode, in which e denotes the IP destination address of the RREP, indicating the next hop address towards the target. The guard of this transition contains two functions: NostaleRM(nextnode, rrepm) as before, ensures that the routing information of the originator contained in the RREP is fresh; Nodeid(nextnode) = e ensures that nextnode is the node indicated by the IP destination address of the RREP. As shown in the arc inscription from this transition to the place RREP message, if nextnode is not the node indicated by the target in rrepm, determined by the function not(Destina1(nextnode, (rrepm, e))), it updates (rrepm,e) by UpdateRREP(nextnode, (rrepm, e)) and then unicasts the processed RREP to the next hop to the target. It also updates its routing table based on the received RREP by the function ReceiveRREP(nextnode, (rrepm, e)), as shown in the arc expression from this transition to the place Nodes. Otherwise, if nextnode is the target, it does not forward the processed RREP. The route entry required to the newly found destination is stored in its updated routing table.

We interpret the occurrence of transition Node receives RREP to mean at any time an node nextnode can receive and process a unicast RREP, (rrepm,e), to itself, then it unicast the processed RREP to the next hop forwards the target. If nextnode is the target node indicated in the received RREP, that means the route discovery generated by this node is completed, the node gets the route entry it requests. On the other hand, transition Lose-RREP models that the RREP is losting during the transmission. The occurrence of this transition indicates a failure of route discovery, which leads to the originator to broadcast another RREQ if its transmission counter has not reached MaxTries.

Transition Route Error models the process of a node detecting a broken link to the destination when transmitting data packets, and then initiating a RERR. The arc expression from the place Nodes to this transition (anode) represents an arbitrary node. The guard Aroute(anode, n) ensures that there is a previously valid route entry for n in the routing table of anode, where n represents an arbitrary node (except anode itself). The arc inscription from this transition to the place Nodes is the function, RouteError(anode, n). This function returns anode after it handles the route entries for n (and for additional unreachable destinations if they exist in its routing table) by marking them as invalid. The arc inscription from this transition to place RERR Message is the function, RERRmessage(anode, n). This function returns a RERR which is deposited in RERR Message. This represents the RERR being broadcast in the MANET.

Transition Node Receives RERR models the process of a node receiving a RERR. The arc expression neinode from Nodes to this transition represents a neighbour node receiving the RERR. The arc expression rerrm from RERR message to this transition represents a RERR message transmitted in the MANET. The guard Aneighbour(neinode, rerrm) ensures that the transmitter of the rerrm is a neighbour of neinode. The arc expression from this transition to the place Nodes is the function, ReceiveRERR(neinode, rerrm)). This function returns neinode with its routing table updated based on rerrm, including marking the route entry for any affected destination as invalid. The function RemoveRE(neinode, rerrm) determines whether at least one route entry in the routing table of neinode is affected by rerrm, while, postprocessingRE(neinode,rerrm) checks whether rerrm is permitted to transmit after processed. If both functions are satisfied, neinode processes rerrm, including handling the message header, and removing each unreachable destination that does not result in a change to its routing table. This procedure is realised by the function UpdateRERR(neinode,rerrm) and the result is stored in place RERR message. Otherwise, if any function is not satisfied, the RERR is not rebroadcast. We interpret the occurrence of Node Receives RERR mean that any node that has not previously received the RERR, receives a RERR, updates its routing table based on the RERR, then rebroadcasts the processed RERR if necessary. Conversely, the transition Lose-RERR models the loss of the RERR during the transmission.

3.3 Functions designed in the model

In the CPN model, the routing algorithms of DYMO are mainly realised by the functions contained in the arc inscriptions. All functions are given in a technical report [35]. However, they are too extensive to be included in this paper. Hence, we just give a brief illustration of one of the functions ReceiveRM() (see Fig. 5) as follows (all functions used by this function can be found in [35]).

By this function, a node updates its routing information based on the received message (a RREQ or a RREP). The node comprises its IP address n, its routing table rt and a transmission counter for each target tries typed by the colour set RREQtries. The received

```
------------------------------------------------------------------------
1  fun ReceiveRM((n,rt,tries),(h,(mheader1,(num1,adds,tadd))))=
2  let
3  val PrePro = preprocessingQ(mheader1)
4  val IncreHop = UpdateHop(adds)
5  val InorderAd= InorderRM(IncreHop,[])
6  val proRM = (h,(PrePro,(num1,InorderAd,tadd)))
7  val InorderRT1 = InorderTable(rt,[])
8  val RTs = UpdateRT((n,InorderRT1,tries),proRM)
9  val TarG= Getadd(tadd)
10 val TypeRM= GetType(proRM)
11 val NewRTSeq=UpdateSeq(n,RTs,tries)
12 val NodeQ = IncreSeq((n,RTs,tries),proRM)
13 in
14 if TypeRM=RREQ
15 then  NodeQ
16 else if (n<>TarG)
17      then (n,NewRTSeq,tries)
18      else (n,RTs,(0,0))
19 end;
------------------------------------------------------------------------
```

Fig. 5. Function ReceiveRM()

message contains the IP source address h, the message header mheader1 and the message body comprising the number of addresses num1, addresses (except the target) and their routing information adds and the target address and its routing information tadd. There is a local declaration, *let/in/end*. By function preprocessingQ(), the message header is bound to a variable PrePro (see line 3 in Fig. 5), after its hoplimit is decreased by one and its hop count is incremented by one. Then the hop count of each address contained in the message, except the target, is increased by one by the function UpdateHop(), and the result is bound to IncreHop. To facilitate the comparison, the addresses contained in adds are arranged in ascending order by the function InorderRM() and the result is bound to InorderAd (line 5). Similarly, the route entries in the node's routing table are arranged in ascending order by the function InorderTable(), and bound to InorderRT1 (line 7). Then the node updates its routing table based on the received message proRM by function UpdateRT, and the result is bound to RTs (line 8). Moreover, the IP address of the target is obtained by function Getadd and is bound to a variable TarG (line 9). Similarly, the type of the processed message is got by function GetType(), and is bound to a variable TypeRM (line 10). As shown in line 14 in Fig. 5, if TypeRM is represented as RREQ, that means the node receives a RREQ. In this case, if the node is an intermediate node, it increases its sequence number by one because of appending its routing information to the RREQ. Or if the node is the target, whether it increases its sequence number depends on the comparison of its sequence number with that contained in the RREQ. This procedure is realised by function IncreSeq(), and the result bound to variable NodeQ is returned (line 15). Otherwise, if the message is a RREP, and if the node is an intermediate node, its updated routing formation is returned (line 17), where its own sequence number is incremented by function UpdateSeq() (the result is bound to variable NewRTSeq, see line 11). Or if it is the target, its updated routing information is returned with tries reset as (0, 0), as shown in line 18 in Fig. 5.

4 Model Simulation

The purpose of the following simulation of the CPN model is to provide some insight into the operations of DYMO. We consider the operation of the CPN with 5 nodes, which addresses (node IPs) 1 to 5. Suppose the initial state of the MANET is given as the initial marking of place Nodes in Fig. 3. We consider an execution of the CPN model which includes route discovery and route maintenance. The markings of the model during the simulation are shown in Fig. 6. Places that are not shown are ignored or are empty.

Consider that Create route request occurs, with orinode bound to node 1 and d bound to IP address 5. Hence, node 1 creates a RREQ for node 5 and broadcasts it. The markings are shown in (1) of Fig. 6. Then the transition Node receives RREQ occurs, with node 3 being bound to internode and rreqm bound to the created RREQ. That means node 3 receives the RREQ broadcast in the network. Node 3 processes the RREQ but drops it, because the hoplimit of the processed RREQ is equal to zero. The markings are shown in (2) of Fig. 6. Time-out and retransmit occurs in the next step with orinode bound to node 1. Thus node 1 rebroadcasts RREQ for node 5 again. The markings are shown in (3) of Fig. 6. Transition Node receives RREQ occurs again, with internode bounding to node 3 and rreqm bound to the new RREQ. Node 3 processes the RREQ and updates its routing table based on the RREQ, then broadcasts the processed RREQ. The current markings are shown in (4) of Fig. 6. Transition Node receives RREQ occurs again, with node 5 being bound to internode and the RREQ sent by node 3 being bound to rreqm. Node 5 is the target in the received RREQ. Hence, it processes the RREQ and creates a RREP in response, and then drops the RREQ. The markings are given in (5) of Fig. 6.

Now we consider the transmission of the RREP created by node 5 in the network. The transition Node receives RREP occurs, where the next node can only be bound to node 3, because it is the next hop for node 1 in the routing table of node 5. Thus, node 3 processes the RREP, updates its routing table, and unicasts the processed RREP to node 1. The markings are shown in (6) of Fig. 6. Transition Node receives RREP occurs, with node 1 being bound to nextnode and the RREP sent by node 3 being bound to (rrepm,e). Node 1 updates its routing table, and does not transmit the processed RREP. As shown in (7) of Fig. 6, *(5, hopc(2), 0, 3, 0, 2, valid)* is the route entry required.

Assume in the next step the transition Route Error occurs, with node 3 being bound to anode and IP address 5 being bound to n. We interpret this as node 3 detecting that the link to node 5 is broken when transmitting data packets. So node 3 sets the route entry for node 5 as invalid, and generates a RERR. Then the transition Node Receives RERR occurs, with node 1 being bound to neinode, and the created RERR being bound to rerrm. So node 1 updates its routing table based on the received RERR. The marking of the place Nodes is shown in (8) of Fig. 6.

5 Discussion

The execution sequence described above shows that the CPN model can simulate the DYMO operations in the MANET, where the topology of the network can change arbitrarily. While specifying DYMO using CPNs, we found some ambiguous statements that could lead to incorrect routes. We describe them in detail as follows.

- **Hop Count (HopCnt)**: there are ambiguous statements about the hop count in DYMO, which may lead to incorrect routes. For instance, when processing a RREQ/RREP, a node

```
(1) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 2, valid)], (5,1)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(3, hopc(0), 0, 3, 0, 1, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(5, hopc(0), 0, 5, 0, 1, valid)], (0,0))
RREQ message: 1‘(1, ((DYMORREQ, 1, 0), (2, [(1, 2, hopc(0), 0, 0)], (5, 0, unknown, 0, 0))))
```

```
(2) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 2, valid)], (5,1)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 2, valid), (3, hopc(0), 0, 3, 0, 2, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(5, hopc(0), 0, 5, 0, 1, valid)], (0,0))
```

```
(3) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid)], (5,2)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 2, valid), (3, hopc(0), 0, 3, 0, 2, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(5, hopc(0), 0, 5, 0, 1, valid)], (0,0))
RREQ message: 1‘(1, ((DYMORREQ, 3, 0), (2, [(1, 3, hopc(0), 0, 0)], (5, 0, unknown, 0, 0))))
```

```
(4) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid)], (5,2)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 3, valid), (3, hopc(0), 0, 3, 0, 3, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(5, hopc(0), 0, 5, 0, 1, valid)], (0,0))
RREQ message: 1‘((3,((DYMORREQ,2,1),(3,[(1,3,hopc(1),0,0),(3,3,hopc(1),0,0)],(5,0,unknown,0,0))))
```

```
(5) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid)], (5,2)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 3, valid), (3, hopc(0), 0, 3, 0, 3, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(1, hopc(2), 0, 3, 0, 3, valid), (3, hopc(2), 0, 3, 0, 3, valid), (5, hopc(0), 0, 5, 0, 2, valid)], (0,0))
RREP message: 1‘((5, ((DYMORREP, 4, 0), (2, [(5, 2, hopc(0), 0, 0)], (1, 3, hopc(2), 0, 0)))), 3)
```

```
(6) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid)], (5,2)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 3, valid), (3, hopc(0), 0, 3, 0, 4, valid), (5, hopc(1), 0, 5, 0, 2, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(1, hopc(2), 0, 3, 0, 3, valid), (3, hopc(2), 0, 3, 0, 3, valid), (5, hopc(0), 0, 5, 0, 2, valid)], (0,0))
RREP message: 1‘((3, ((DYMORREP, 3, 1), (3, [(5, 2, hopc(1), 0, 0), (3, 4, hopc(1), 0, 0)],
(1, 3, hopc(2), 0, 0)))), 1)
```

```
(7) Nodes: 1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid), (3, hopc(2), 0, 3, 0, 4, valid), (5, hopc(2), 0, 3, 0, 2, valid)],
(0,0)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 2, valid), (3, hopc(0), 0, 3, 0, 3, valid), (5, hopc(1), 0, 5, 0, 1, valid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(1, hopc(2), 0, 3, 0, 3, valid), (3, hopc(2), 0, 3, 0, 3, valid), (5, hopc(0), 0, 5, 0, 2, valid)], (0,0))
```

```
(8) Nodes:1‘(1, [(1, hopc(0), 0, 1, 0, 3, valid), (3, hopc(2), 0, 3, 0, 4, valid), (5, hopc(2), 0, 3, 0, 2, invalid)],
(0,0)) ++
1‘(2, [(2, hopc(0), 0, 2, 0, 1, valid)], (0,0)) ++
1‘(3, [(1, hopc(1), 0, 1, 0, 3, valid), (3, hopc(0), 0, 3, 0, 4, valid), (5, hopc(1), 0, 5, 0, 2, invalid)], (0,0)) ++
1‘(4, [(4, hopc(0), 0, 4, 0, 1, valid)], (0,0)) ++
1‘(5, [(1, hopc(2), 0, 3, 0, 3, valid), (3, hopc(2), 0, 3, 0, 3, valid), (5, hopc(0), 0, 5, 0, 2, valid)], (0,0))
```

Fig. 6. The markings of the CPN model for one simulation

increments the hopcount of each address except the target, then it creates or updates a route entry for each address except the target. If it is an intermediate node, it may append its routing information to the processed message, in which the *Node.Hopcnt* if included is set to **one**. The definition of AddTLV.Node.Hopcnt [5] is "The number of IP hops to reach the associated Node.address.". But the source node from which the number of hops is counted is not mentioned explicitly. It may either be the originator of this message, or the intermediate node which is processing the message. In our understanding, this definition, if the first case is true, is similar to that of *msg-hopcnt*: "the number of hops a message has travelled" [6]. Hence, that *Node.Hopcnt* is set to *msg-hopcnt* rather than one seems more reasonable. Furthermore, the other possible problem exists in creating or updating a route entry when processing a message, in which "the Route.HopCnt is set to the Node.HopCnt". The definition of Route.HopCnt, the hop count of the route entry, is that:"the number of intermediate node hops traversed before reaching the Route.address node" [5]. That means the number of hops from a node to the destination. For instance, as shown in (3) of Fig. 6, there is a route entry for node 1 in node 3's routing table, where the Route.HopCnt is set to hopc(1) indicating there is one hop from node 3 to node 1. In the example described in section 2, Node.HopCnt is not included when an intermediate node appends its address. Now, we consider in the case of Fig. 1(5), that when node 4 appends its address and sequence number to the processed RREQ, the Node.Hopcnt is included, indicating the hops from node 1 (the originator node) to node 4. Then node 2 receives and processes the RREQ send by node 4. When creating a route entry to node 4, the corresponding Route.HopCnt indicates the hops from node 2 to node 4. Even in this case, they both are one hop. But we can know that they indicate the distance from different nodes to the same destination. On the other hand, if the second case is true, that means the Node.HopCnt appended by node 4 is the hop count from node 4 to itself. Then when node 2 processes the RREQ sent by node 4, it increases the Node.HopCnt associated with address 4 by one. Thus the updated Node.HopCnt records the hops from node 2 to node 4. In this case, the Route.HopCnt is set to the Node.HopCnt. Unfortunately, if the initial Node.HopCnt is one, incorrect routing information still exists. For instance, as shown in (4) of Fig. 6, node 3 appends its routing information $(3, 3, \mathit{hopc}(1), 0, 0)$ to the received RREQ, where hopc(1) indicates the Node.Hopcnt set to one. When node 5 receives and processes the RREQ sent by node 3, it creates a route entry to node 3. As shown in (5) of Fig. 6, the route entry for node 3 has two hops in the routing table of node 5, which is incorrect. Similarly, an incorrect Route.HopCnt occurs in the route entry for node 3 in the routing table of node 1 in (7) of Fig. 6. The reason for the error is that node 5 increments the Node.Hopcnt of node 3, then it creates a route entry to node 3 using the incremented Node.Hopcnt as the Route.HopCnt. Why is the route entry to node 1 correct? Because we set the initial Node.HopCnt of node 1 as zero (this is not mentioned in DYMO, but we think it is necessary), so that node 5 can use incremented Node.Hopcnt of node 1 as the Route.HopCnt of the route entry for node 1. In this case, if an intermediate node appends its Node.Hopcnt as **zero**, the errors can be avoided.

– **definition of Net-diameter** : for the creation of a RREP/RERR, the initial *msg-hoplimit* (see Fig. 2) is set to *Net-Diameter*. The suggested value of Net-Diameter is 10, but there is no definition of this parameter in DYMO. For RREQ, the msg-hoplimit may be set in accordance with an expanding ring search as described in AODV [25], but, it is not mentioned whether the suggested values of parameters in this search given in AODV are suitable for DYMO.

- **assumptions about the originator node**: to simplify the routing algorithms compared with other reactive protocols, DYMO only gives a general statement of its operations. Hence, some operations require assumptions to be made. For instance, there is no mention that the originator discards its own RREQs, echoed by its neighbours, so it appears that they should be rebroadcast. Since sequence numbers are identical, but the next operation is hard to do without the HopCnt information. However, there is no statement about that whether the originator node has its own HopCnt, so we have to assume that the originator has its HopCnt as zero. In addition, we know in DYMO that a node increments its sequence number by one before creating a RREQ, and may send another RREQ if it does not receive a reply before its retransmission timer expires. But it does not mention whether the originator updates the originator sequence number in this RREQ. If it does not increment its sequence number then the following problem may occur. An intermediate node, which has received the same RREQ before, may determine this RREQ is stale and hence discard it. Thus the retransmission will be useless and the route discovery procedure will fail no matter how many times the originator retransmits a RREQ. To solve this possible problem, we assume that the originator increments its sequence number each time it retransmits a RREQ.
- **propagation of the RERR**: when appending additional unreachable destinations to a new RERR, DYMO states:"Additional unreachable nodes that required the same unavailable link (routes with the same Route.NextHopAddress and Route.NextHopInterface) MAY be added to the RERR" [5]. If this unreachable destination is not a next hop node (namely a neighbour), it is hard to determine if any other route entry with the same next hop is unreachable or not. Hence in this case, we just include the unreachable destination itself in the RERR, similar to AODV.
- **the use of sequence numbers**: similar to AODV, in DYMO the higher sequence number indicates fresher routing information. Both protocols accomplish sequence number "rollover" [5]: when a sequence number is assigned to the largest number (i.e. 65535), then it returns to a previous number (256 in DYMO) when incremented. The comparison between sequence numbers use signed 16-bit arithmetic [5]. However, no algorithm is given in DYMO to distinguish the number after rollover and the number without undergoing rollover. For example, in DYMO, if there are two route entries to the same destination, and the destination sequence number of the first route entry is 257 after rollover and that of the second route entry is 300 without rollover, then the second route entry will be regarded as fresher than the first one, which is incorrect.

6 Conclusions and Future Work

This paper has presented the first formal specification of the DYMO routing protocol [5] using CPNs. Retransmission of messages and an optimization of the protocol known as the expanding ring search are included. By simulating the model, we discovered errors due to ambiguous definitions in the document, and suggest modifications to eliminate these errors. This is the first time these errors have been reported as far as we are aware. We plan to model the complete broadcast in the further work that is, a broadcast may be received by any number of nodes in the system, as is done in a real MANET.

Acknowledgements

The authors would like to thank Guy Gallasch for his assistance with some of the detailed technical parts of the model. We also gratefully acknowledge the constructive comments of the anonymous referees and our colleague, Nimrod Lilith, for helping to improve the quality of this paper. Cong Yuan is supported by an Australian Government International Postgraduate Scholarship.

References

1. S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic. Mobile Ad Hoc Networking. IEEE Press, New York, 2004.
2. R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.
3. J. Billington. Many-sorted High-Level Nets. In *the Third International Workshop on Petri Nets and Performance Models*, pages 11–13, Kyoto, Japan, December 1989. IEEE CS Press, Washington, D. C., USA, 1989.
4. I. D. Charkeres, E. M. Belding-Royer, and C. E. Perkins. *Dynamic MANET On-demand (DYMO) Routing Protocol.* IETF Internet Draft, draft-ietf-manet-dymo-00.txt, February 2005.
5. I. D. Charkeres and C. E. Perkins. *Dynamic MANET On-demand (DYMO) Routing Protocol.* IETF Internet Draft, draft-ietf-manet-dymo-05.txt, June 2006.
6. T. Clausen, C. Dearlove, J. Dean, and C. Adjih. *Generalized MANET Packet/Message Format.* http://www.ietf.org/internet-drafts/draft-ietf-manet-packetbb-01.txt, June 2006.
7. T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol.* Request for Comments 3626, http://www.ietf.org/rfc/rfc3626.txt, October 2003.
8. S. Corson and J. Macker. *Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations (RFC 2501).* http://www.ietf.org/rfc/rfc2501.txt.
9. G. Findlow and J. Billington. High-Level Nets for Dynamic Dining Philosophers Systems. In M.Z. Kwiatkowska, M.W. Shields, and R.M. Thomas, editors, *Semantics for Concurrency*, pages 185–222. Springer-Verlag, 1990.
10. http://www.ietf.org/proceedings-directory.html, Paris, France. *Proceedings of the 63rd Internet Engineering Task Force*, August 2005.
11. IETF. Mobile Ad-hoc Networks (manet). http://www.ietf.org/html.charters/manet-charter.html.
12. IETF. Online Proceedings. http://www.ietf.org/proceeding-directory.html.
13. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1–3. Springer-Verlag, 1997.
14. D. B. Johnson, D. A. Maltz, and Y. C. Hu. *The Dynamic Source Routing Protocol for Mobile Ad hoc Networks (DSR).* IETF MANET Working Group, Internet Draft, draft-ietf-manet-dsr-09.txt, April 2003.
15. L. Klein-Berndt. *National Institute of Standards and Technology (NIST).* http://sourceforge.net/projects/nist-dymo/.
16. 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.
17. L. M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad Hoc Networking. In H. Ehrig and W. Damm et al, editors, *Integration of Software Specification Techniques for Applications in Engineering, LNCS 3147*, pages 248–269. Springer-Verlag, 2004.
18. L. M. Kristensen, J. B. Jørgensen, and K. Jensen. Application of Coloured Petri Nets in System Development. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri nets - Advanced in Petri Nets, LNCS3098*, pages 626–685. Springer-Verlag, 2004.
19. L. M. Kristensen, M. Westergaard, and P.C. Nørgaard. Model-based Prototyping of Interoperability Protocol for Mobile Ad-hoc Networks. In J. Romijn, G. Smith, and J. Pol, editors, *Proceedings of 5th International Conference on Integrated Formal Methods (IFM2005), LNCS3771*, pages 266–286, Eindhovem, the Netherlands, November 2005. Springer-Verlag.
20. K. Kuladintihi and C. Gorg. *DYMO for OPNET.* University of Murcia, http://www.fb1.uni-bremen.de/comnets/opnet/.
21. G. Malkin. RIP Version 2- Carrying Additional Information. RFC 1723(draft standard). Technical report, Internet Engineering Task Force, November 1994.

22. G. S. Malkin and M. E. Steenstrup. Distance-Vector Routing. In *Routing in Communications Networks*, pages 83–98. Prentice-Hall, 1995.
23. R. Ogier, F. Templin, and M. Lewis. *Topology Dissemination Based on Reverse-path Forwarding (TBRPF)*. Request for Comments 3684, http://www.ietf.org/rfc/rfc3684.txt, February 2004.
24. C. E. Perkins, E. Belding-Royer, and S. Das. *Ad Hoc On-Demand Distance Vector (AODV) Routing*. IETF Internet-Draft, draft-ietf-manet-aodv-09.txt, Dec. 2001.
25. C. E. Perkins, E. Belding-Royer, and S. Das. *Request for Comments 3561: Ad hoc On Demand Distance Vector (AODV) Routing*. http://www.ietf.org/rfc/rfc3561.txt, July 2003.
26. C. E. Perkins and P. Bhagwat. Highly Dynamic Destination Sequenced Distance Vector (DSDV) for Mobile Computers. In *ACM SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.
27. C. E. Perkins and P. Bhagwat. *Ad Hoc Networking, Chapter 3: DSDV Routing over a Multihop Wireless Network of Mobile Computers*. Addison-Wesley, 2001.
28. C. E. Perkins and E. M. Royer. Ad Hoc On Demand Distance Vector(AODV)Routing. Technical report, Internet-Draft,Version 2, IETF, March 1998.
29. C. E. Perkins and E. M. Royer. Ad-hoc On-demand Distance Vector Routing. In *the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.
30. F. Ros and P. Ruiz. *DYMOUM*. University of Murcia, http://masimum.dif.um.es/?Software:DYMOUM.
31. A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Upper Saddle River, NJ, 4th edition, 2003.
32. O. Wibling, J. Parrow, and A. Pears. Automatized Verification of Ad Hoc Routing Protocols. In D. Frutos-Escrig and M. Nu, editors, *Formal Techniques for Networked and Distributed Systems (FORTE 2004), LNCS 3235*, pages 343–358. Springer-Verlag, 2004.
33. C. Xiong, T. Murata, and J. Leigh. An Approach for Verifying Routing Protocols in Mobile Ad Hoc Networks Using Petri Nets. In *Proceedings of IEEE 6th CAS Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication*, pages 537–540, Shanghai, China, 2004.
34. C. Xiong, T. Murata, and J. Tsai. Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Wireless Networks Using Colored Petri Nets. In *Proceedings of Workshop on Formal Methods Applied to Defence Systems in Formal Methods in Software Engineering and Defence Systems, Conferences in Research and Practice in Information Technology*, volume 12, pages 145–153, 2002.
35. C. Yuan and J. Billington. A Highly Abstract Model of Dynamic MANET On-demand Routing in Mobile Ad hoc Networks using CPNs. Draft technical report, Computer Systems Engineering Centre, University of South Australia, 2006.
36. C. Yuan and J. Billington. A Modelling Approach for Reactive Routing in Mobile Ad hoc Networks using CPNs. Draft technical report, Computer Systems Engineering Centre, University of South Australia, 2006.
37. C. Yuan, J. Billington, and J. Freiheit. An Abstract Model of Routing in Mobile Ad hoc Networks. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 137–156, Aarhus, Denmark, October 2005. Department of Computer Science Technical Report, DAIMI PB 576.

Mining CPN Models

Discovering Process Models with Data from Event Logs

A. Rozinat, R.S. Mans, and W.M.P. van der Aalst

Department of Information Systems, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.rozinat,r.s.mans,w.m.p.v.d.aalst}@tue.nl

Abstract. Process-aware information systems typically log events (e.g., in transaction logs or audit trails) related to the actual execution of business processes. Analysis of these execution logs may reveal important knowledge that can help organizations to improve the quality of their services. Starting from a process model, which can be discovered by conventional process mining algorithms, we analyze how data attributes influence the choices made in the process based on past process executions using decision mining, or decision point analysis. In this paper we describe how the resulting model (including the discovered data dependencies) can be represented as a *Colored Petri Net* (CPN), and how further perspectives, such as the performance and organizational perspective, can be incorporated. We also present a *CPN Tools Export* plug-in implemented within the ProM framework. Using this plug-in simulation models in ProM obtained via a combination of various process mining techniques can be exported to CPN Tools. We believe that the combination of automatic discovery of process models using ProM and the simulation capabilities of CPN Tools offers an innovative way to improve business processes. The initially discovered process model describes reality better than most hand-crafted simulation models. Moreover, the simulation models are constructed in such a way that it is easy to explore various redesigns.

1 Introduction

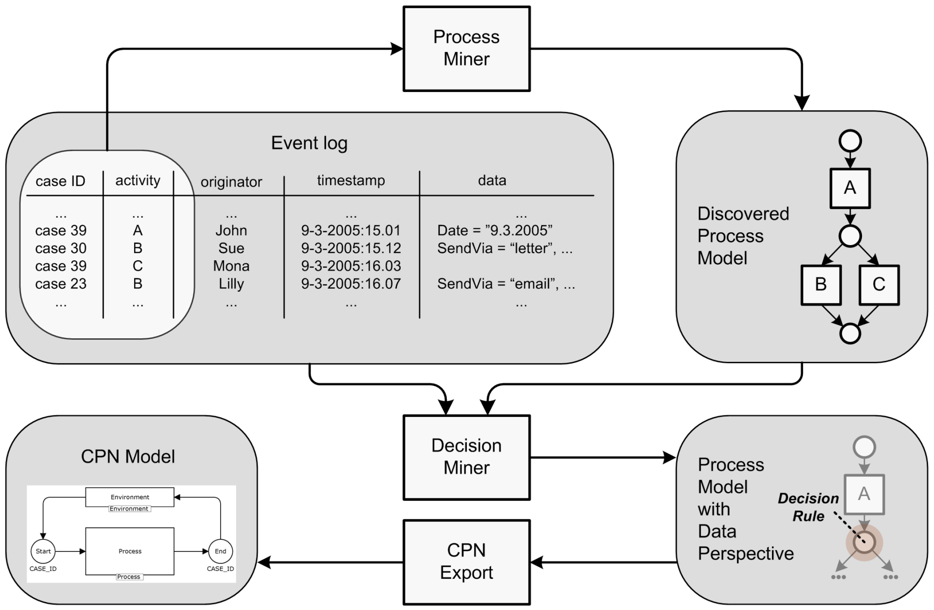
Process mining has proven to be a valuable approach that provides new and objective insights into the way business processes are really handled within organizations. Taking a set of real process executions (the so-called "event logs") as the starting point, these techniques can be used for *process discovery* and *conformance checking*. Process discovery [4, 6] can be used to automatically construct a process model reflecting the behavior that has been observed and recorded in the event log. Conformance checking [1, 16] can be used to compare the recorded behavior with some already existing process model to detect possible deviations. Both may serve as *input* for designing and improving business processes, e.g., conformance checking can be used to find problems in existing processes, and process discovery can be used as a starting point for process analysis and system configuration. While there are several process mining algorithms that deal with the control flow perspective of a business process [4] *less attention has been paid to how data attributes affect the routing of a case.* Classical process mining approaches consider all choices to be non-deterministic. The approach presented in this paper investigates how values of data attributes influence particular choices in the model.

Most information systems (cf. WFM, ERP, CRM, SCM, and B2B systems) provide some kind of *event log* (also referred to as transaction log or audit trail) [4] where an event refers to a case (i.e., process instance) and an activity, and, in most systems, also a timestamp, a performer, and some additional data. Nevertheless, many process mining techniques only make use of the first two attributes in order to construct a process model which reflects the causal relations that have been observed among the activities. In this paper we start from a discovered process model (i.e., a model discovered by conventional process mining algorithms), and we try to enhance the model by integrating patterns that can be observed from data modifications, i.e., a *decision point analysis* [18] will be carried out in order to find out which properties (i.e., valuations of data attributes) of a case might lead to taking certain paths in the process. Colored Petri Nets (CPNs) [13] are used as a representation for the enhanced model because of their expressiveness and the good tool support provided through CPN Tools [19] (which, for example, has strong simulation capabilities). Furthermore, the hierarchy concept allows for the composition of a CPN model in a modular way. The time concept and the availability of many probability distributions in CPN Tools allow for the modeling of performance aspects. Moreover, by introducing resource tokens also organizational and work distribution aspects can be modeled.

Figure 1 illustrates the overall approach. First of all, some process mining algorithm is used to discover a process model in terms of a Petri net (e.g., the α-algorithm [6]). Note that conventional process mining techniques (e.g., based on the α-algorithm) only use the first two columns of the event log depicted in Figure 1. However, the event log may also contain information about the people performing activities (cf. originator column), the timing of these activities (cf. timestamp column), and the data involved (cf. data column). In the next step we make use of the additional information, the data column to be precise. The Decision Miner presented in this paper uses this information to discover rules for taking alternative paths based on values of the data attributes present in the process. Finally, the process model including the data perspective is exported as a CPN model. The CPN model may be extended with additional information about time and resources. This information may be manually included or is extracted from the log based on the originator column and timestamp column.

To directly support the generation of a CPN model for business processes we have implemented a *CPN Tools 2.0 Export* plug-in (in the remainder of this paper referred to as CPN Export plug-in) in the context of the ProM framework[1], which offers a wide range of tools related to process mining and process analysis.

[1] Both documentation and software (including the source code) can be downloaded from *www.processmining.org.*



case ID	activity	originator	timestamp	data
...	...	...	...	...
case 39	A	John	9-3-2005:15.01	Date = "9.3.2005"
case 30	B	Sue	9-3-2005:15.12	SendVia = "letter", ...
case 39	C	Mona	9-3-2005:16.03	
case 23	B	Lilly	9-3-2005:16.07	SendVia = "email", ...
...	...	...	...	...

Fig. 1. The approach pursued in this paper

The paper is organized as follows. First, Section 2 introduces a simple example process that is used throughout the paper. Then, the decision mining approach is explained briefly in Section 3. Subsequently, we describe how a business process (including multiple perspectives) can be represented as a CPN in Section 4. Section 5 presents the CPN Export plug-in of the ProM framework. Finally, related work is discussed in Section 6, and the paper concludes by pointing out directions for future research.

2 Running Example

As pointed out in Figure 1, the first step in the decision mining process is to obtain a process model without data through some classical *Process Miner*, e.g., a Petri net discovered using the α-algorithm. Figure 2(a) shows an event log in a schematic way, i.e., as a set of event traces. Note that this information can be extracted from the first two columns of the event log shown in Figure 1. Based on this information the α-algorithm automatically constructs the process model shown in Figure 2(b).

The example process used throughout the paper sketches the processing of a liability claim within an insurance company: first, some data related to the claim is registered (cf. activity A in Figure 2), and then either a full check or a policy-only check is performed (B or C). Afterwards, the claim will be evaluated (D), and then it is either rejected (F) or approved (E and G). Finally, the case is archived and closed (H).

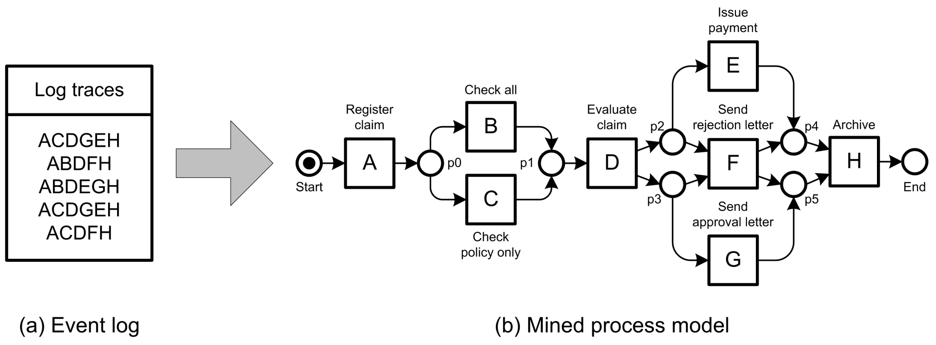
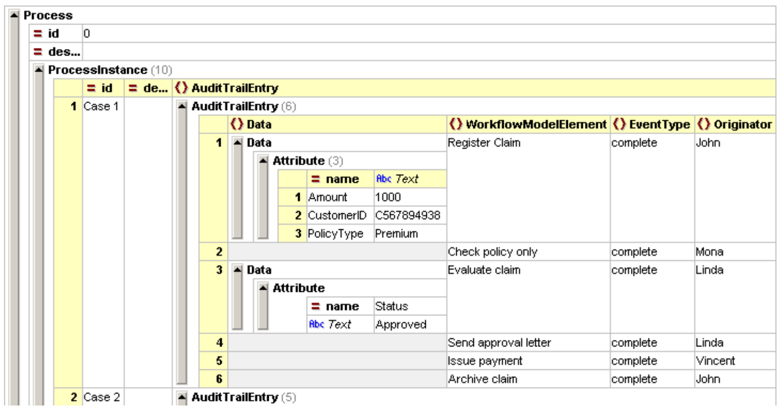


Fig. 2. Process mining phase

Now we have discovered the control flow perspective of the process. But the process execution log contains much more valuable information. In order to generate a simulation model that reflects as close as possible the process that has been observed, case data attributes, timestamps, and originator information can be analyzed to reveal characteristics related to the *data*, *performance*, and *organizational* perspectives. Figure 3 depicts a screenshot of the event log in MXML[2] format, and in the following we will have a closer look at it, considering these perspectives.



```
Process
  = id     0
  = des...
  ProcessInstance (10)
      = id     = de...  () AuditTrailEntry
    1 Case 1            AuditTrailEntry (6)
                            () Data                              () WorkflowModelElement  () EventType  () Originator
                          1 Data                                 Register Claim           complete      John
                              Attribute (3)
                                    = name        Abc Text
                                  1 Amount        1000
                                  2 CustomerID    C567894938
                                  3 PolicyType    Premium
                          2                                      Check policy only        complete      Mona
                          3 Data                                 Evaluate claim           complete      Linda
                              Attribute
                                  = name     Status
                                  Abc Text   Approved
                          4                                      Send approval letter     complete      Linda
                          5                                      Issue payment            complete      Vincent
                          6                                      Archive claim            complete      John
    2 Case 2            AuditTrailEntry (5)
```

Fig. 3. Fragment of the example log in MXML format viewed using XML Spy

(a) *Data perspective.* Here a data item within an audit trail entry (i.e., an event) is interpreted as a case attribute that has been created, or modified.

[2] Both the corresponding schema definition and the ProMimport framework, which converts logs from existing (commercial) process-aware information systems to the XML format used by ProM, can be downloaded from *www.processmining.org*.

In Figure 3 one can observe that only activities *Register claim* and *Evaluate claim* have associated data items. During the execution of activity *Register claim* information about the amount of money involved (*Amount*), the corresponding customer (*CustomerID*), and the type of policy (*PolicyType*) are provided, while after handling the activity *Evaluate claim* the outcome of the evaluation is recorded (*Status*). Semantically, *Amount* is a numerical attribute, *CustomerID* is an attribute which is unique for each customer, and both *PolicyType* and *Status* are enumeration types (being either "Normal" or "Premium", or either "Approved" or "Rejected", respectively).

(b) *Performance perspective.* In the example, for simplicity, activities are considered as being atomic and carry no time information. However, information systems dealing with processes typically log events on a more fine-grained level, e.g., they may record *schedule*, *start*, and *complete* events (including timestamps) for each activity. Thus, time information can be used to infer, e.g., activity durations, or the arrival rate of new cases. Furthermore, the frequency of alternative paths represents quantitative information that is implicitly contained in the event log. For example, the event log shown in Figure 3 contains 10 process instances, of which 7 executed activity *Check policy only* and only 3 performed the full check procedure *Check all*.

(c) *Organizational perspective.* In Figure 3 one can observe an event carry information about the resource that executed the activity. In the insurance handling example process 7 different persons have worked together: Howard, Fred, Mona, Vincent, Robert, Linda, and John.

As illustrated in Figure 1, the discovered process model and the detailed log are the starting point for the *Decision Miner*, which analyzes the data perspective of the process in order to discover data dependencies that influence the routing of a case. The idea of decision mining is briefly explained in the next section (see [17] for further details). The Decision Miner constructs a simulation model incorporating the data perspective and passes this on to the CPN Export plug-in. However, in addition to the control-flow and data perspective the simulation model may also contain information about resources, probabilities, and time (i.e., the performance and organizational perspectives).[3] The representation of all these perspectives in terms of a CPN model and the configuration possibilities of the CPN Export plug-in in ProM are described in Section 4 and Section 5.

3 Decision Mining

In order to analyze the choices in a business process we first need to identify those parts of the model where the process splits into alternative branches, also

[3] These perspectives can be added by hand or through additional process mining techniques. We are currently working on integrating the information from various plug-ins, focusing on integrating the performance and organizational perspectives with the information from the Decision Miner.

called *decision points*. Based on data attributes associated to the cases in the event log we subsequently want to find rules for following one route or the other [18].

In terms of a Petri net, a decision point corresponds to a place with multiple outgoing arcs. Since a token can only be consumed by one of the transitions connected to these arcs, alternative paths may be taken during the execution of a process instance. The process model in Figure 2(b) exhibits three such decision points: *p0* (if there is a token, either B or C can be performed), *p2* (seen from this place, either E or F can be executed) and *p3* (seen from this place, either F or G may be carried out). In order to analyze the choices that were made in past process executions, we need to find out which alternative branch was taken by a certain process instance. Therefore, the set of possible decisions must be described with respect to the event log. Starting from the identification of a choice in the process model (i.e., a decision point) a decision can be detected if the execution of an activity in the respective alternative branch of the model has been observed, which requires a mapping from that activity to its "occurrence footprint" in the event log. So, if a process instance contains the given "footprint", this means that there was a decision for the associated alternative path in the process. For simplicity we examine the occurrence of the *first* activity per alternative branch in order to classify the possible decisions. However, in order to make decision mining operational for real-life business processes several challenges posed by, for example, *invisible activities*, *duplicate activities*, and *loops* need to be met. We refer the interested reader to our technical report [17], where these issues are addressed in detail.

After identifying a decision point in a business process and classifying the decisions of the process instances in the log, the next step is to determine whether this decision might be influenced by case data, i.e., whether cases with certain properties typically follow a specific route. The idea is to convert every decision point into a *classification problem* [14, 15, 21], where the *classes* are the different decisions that can be made. As training examples we use the process instances in the log (for which it is already known which alternative path they followed with respect to the decision point). The attributes to be analyzed are the case attributes contained in the log, and we assume that all attributes that have been written *before* the choice construct under consideration are relevant for the routing of a case at that point[4]. In order to solve such a classification problem, various algorithms are available [14, 21]. We decided to use an algorithm based on decision trees (the C4.5 algorithm [15] to be precise). Decision trees are a popular tool for inductive inference and the corresponding algorithms have been extended in various ways to improve practical applicability. For example, they are able to deal with continuous-valued attributes, missing attribute values, and they include effective methods to avoid *over-fitting* the data (i.e., that the tree is too much tailored towards the particular training examples).

[4] We also allow the user to set other scoping rules, e.g., only the data set in a directly preceding activity, or all case data including the data that is set later.

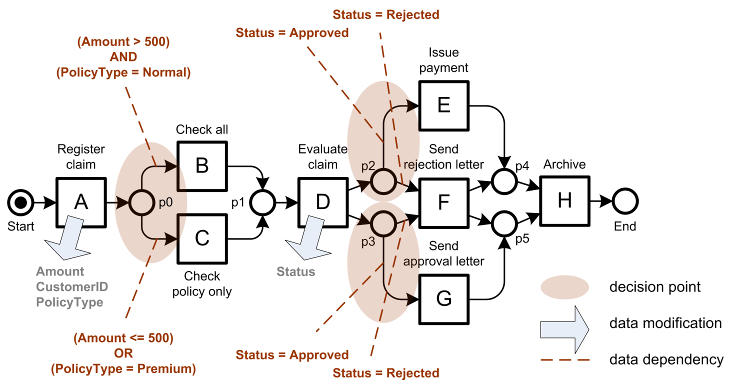


Fig. 4. Enhanced process model

Using decision point analysis we can extract knowledge about decision rules as shown in Figure 4. Each of the three discovered decision points corresponds to one of the choices in the running example. With respect to decision point *p0* the extensive check (activity *B*) is only performed if the *Amount* is greater than 500 and the *PolicyType* is "Normal", whereas a simpler coverage check (activity *C*) is sufficient if the *Amount* is smaller than or equal to 500, or the *PolicyType* is "Premium" (which may be due to certain guarantees given by "Premium" member corporations). The two choices at decision point *p2* and *p3* are both guided by the *Status* attribute, which is the outcome of the evaluation activity (activity *D*).

Now that we have automatically discovered a model integrating both the control-flow and data perspective of the example process, we describe how this information (and information about the performance and organizational perspective) can be represented in a CPN model (Section 4), and show how such a CPN model can be generated in ProM (Section 5). Recall that we have also developed mining techniques for discovering these additional perspectives. However, a detailed description is beyond the scope of this paper.

4 CPN Model of a Business Process

Since we want to make use of the simulation facilities of CPN Tools, we provide the actual process model together with a simulation environment. The top-level page in the hierarchical CPN model is shown in Figure 5. For each process model this page will look identical; the environment generates cases and puts them into the *Start* place, and removes those that have finally reached the *End* place. We assume that the discovered process—represented by the sub-page *Process*—is sound, i.e., any case that enters the sub-page via place *Start* leaves the sub-page via place *End*.

Figure 6 depicts the simulation environment in more detail. One can observe that the CASE_ID color set is used to refer to particular process instances (i.e.,

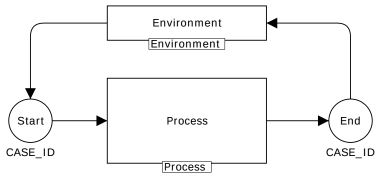


Fig. 5. Overview page

cases). To give each case a unique ID a counter is simply incremented for each generated process instance. For the data perspective, a separate token containing the case ID and a record of case attributes (defined via the DATA color set) is created and initialized. The place *Case data* is modeled as a fusion place as activities may need to inspect or modify data attribute values on different pages in the hierarchical model. Furthermore, the *Resources* fusion place contains the available resources for the process, and therefore determines the environment from an organizational perspective. Finally, each time a token is put back in the *next case ID* place a time delay[5] is added to it, which is used to steer the generation of new cases. In Figure 6 a constant time delay of 3 implements that every 3 time units a new case arrives. Note that the inter-arrival times may also be sampled from some probability distribution discovered by ProM.

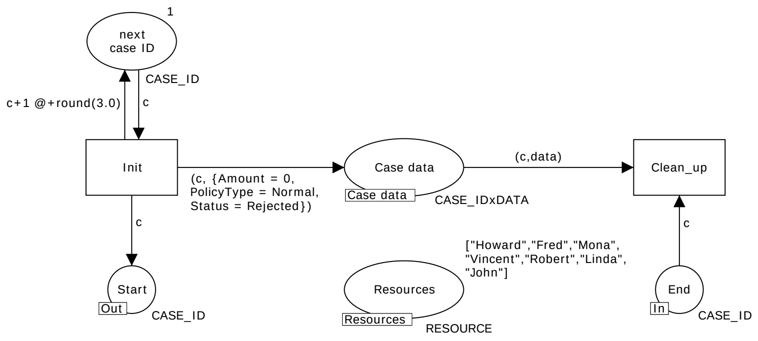


Fig. 6. Environment page

Figure 7 shows the sub-page containing the actual process model, which looks exactly like the original, low-level Petri net. Note that the tokens routed from the *Start* to the *End* place are of type CASE_ID, so that tokens belonging to different instances are not mixed up.

[5] Note that in our simulation model the time delay is always attached to an arc (depending on the token that should be delayed) rather than using the time delay of a transition in order to avoid side effects on other tokens that should actually not be delayed (such as the *Case data* token).

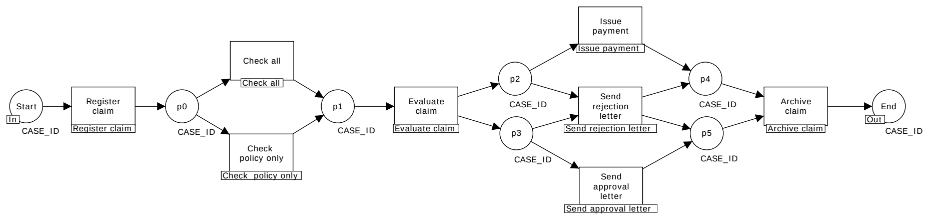
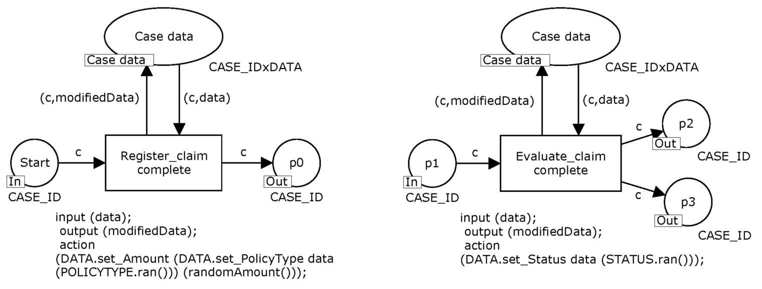


Fig. 7. Process page

Every activity on the process page has its own sub-page containing the actual simulation information. Depending on the covered perspectives these activity sub-pages may look very different. In the following sub sections we will present how simulation information from several dimensions can be represented in terms of a CPN sub-page.

4.1 Data

Taking the enhanced model from Figure 4 as the starting point, we now want to incorporate the discovered data dependencies in the simulation model. The discovered decision rules are based on attributes provided by activity *Register claim* and *Evaluate claim* respectively (see the result described in Section 3). Since the attribute *CustomerID* is not involved in the discovered rules, we discard it from the process model and define process-specific data types for each of the remaining attributes (i.e., AMOUNT, POLICYTYPE, and STATUS).



```
input (data);
 output (modifiedData);
 action
(DATA.set_Amount (DATA.set_PolicyType data
(POLICYTYPE.ran())) (randomAmount()));
```

```
input (data);
 output (modifiedData);
 action
(DATA.set_Status data (STATUS.ran()));
```

(a) Sub page *Register claim*

(b) Sub page *Evaluate claim*

Fig. 8. Writing data items using random values

Figure 8 shows how the provision of case data can be modeled using random values. While a random value for a nominal attribute can be generated by applying the *ran()* function directly to the color set[6] a dedicated random function

[6] Note that—for performance reasons—the *ran()* function can only be used for enumerated color sets with less than 100 elements.

is needed for numeric attributes. In the action part of transition *Register_claim complete* function *POLICYTYPE.ran()* is used to select the policy type ("Normal" or "Premium") and a dedicated function *randomAmount()* is used to set the amount. In this case, the amount is sampled from a uniform distribution generating a value between the lowest and the highest attribute value observed in the event log. However, many other settings are possible.

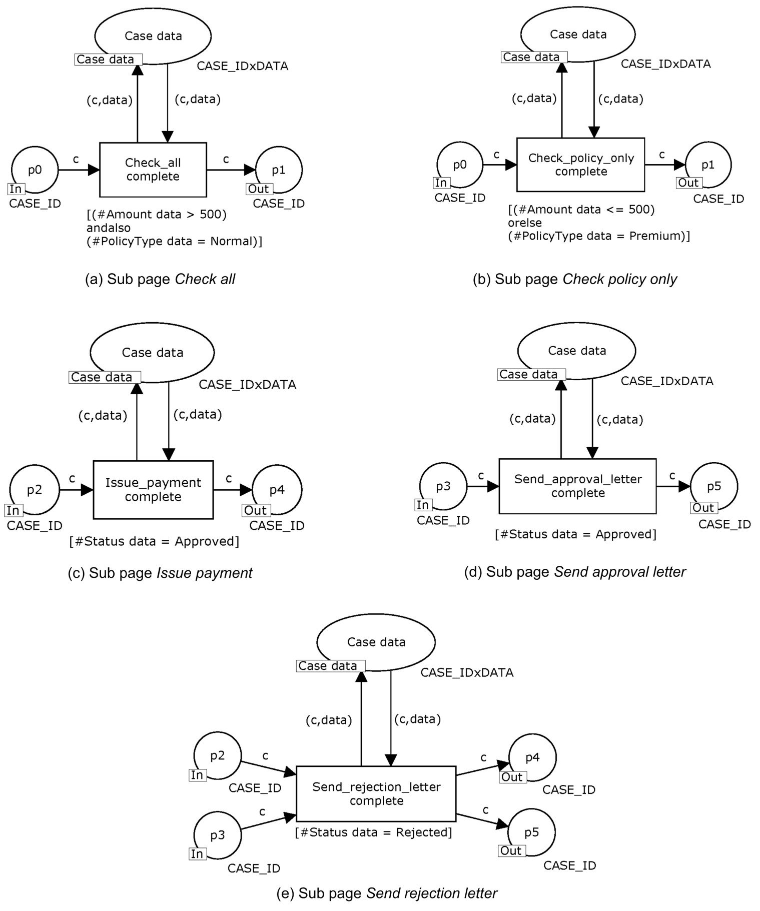

Case data
Case data
CASE_IDxDATA
(c,data)
(c,data)
p0
In
CASE_ID
c
Check_all complete
c
p1
Out
CASE_ID
[(#Amount data > 500) andalso (#PolicyType data = Normal)]
(a) Sub page Check all
Check_policy_only complete
[(#Amount data <= 500) orelse (#PolicyType data = Premium)]
(b) Sub page Check policy only
p2
Issue_payment complete
p4
[#Status data = Approved]
(c) Sub page Issue payment
p3
Send_approval_letter complete
p5
[#Status data = Approved]
(d) Sub page Send approval letter
Send_rejection_letter complete
[#Status data = Rejected]
(e) Sub page Send rejection letter

Fig. 9. Modeling data dependencies using transition guards

Figure 9 shows how the discovered data dependencies can then be modeled with the help of transition guards. If the transition is enabled from a control-flow perspective, it additionally needs to satisfy the given guard condition in order to be fired.

4.2 Time

Although there is no time information in the example event log, we want to include the time dimension in our simulation model because it is relatively easy to extract from most real-life logs. Moreover, this perspective is of utmost importance from a practical point of view. To explain this perspective we assume that—in contrast to the policy-only check, which takes between 3 and 8 time units—the full check procedure needs between 9 and 17 time units to complete. Furthermore, the time between the point where the activity *could have been started* (i.e., all required previous activities were completed) and the point where someone *actually starts* working on it may vary from 3 to 5 time units. Whereas the sub-page shown in Figure 9(a) models the activity *Check all* in an atomic way, one can distinguish between *schedule*, *start*, and *complete* transitions in order to incorporate the *waiting time* and *execution time* of this activity. Figure 10 shows three ways to model this for activity *Check all*.

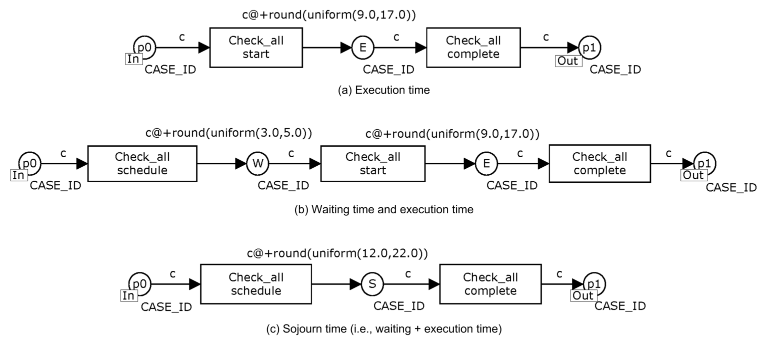


Fig. 10. Different variants of modeling time on sub-page *Check all* depending on the event types (i.e., schedule, start, and complete) present in the log

In Figure 10(a) only the execution time of the activity is modeled. When transition *Check_all start* is fired, a token is produced with the indicated time delay. Similar to the case generation scheme in Figure 6, the token will remain between 9 and 17 time units in place *E* (i.e., the activity is in the state *Executing*) before transition *Check_all complete* may fire.

In Figure 10(b) both the execution time and the waiting time are explicitly modeled. Analogously to the execution time, the waiting time is realized by a time delay that forces the token to reside in place *W* (i.e., the activity is in the state *Waiting*) between 3 and 5 time units before transition *Check_all start* may fire.

In Figure 10(c) the sum of the waiting time and the execution time is modeled. This may be useful if no information is available about the actual start of an activity (i.e., only the time when it becomes enabled and when it is finished is known).

4.3 Resources

In order to gain insight into the organizational perspective we can, for example, analyze the event log with the *Social Network miner* of ProM [3]. One possible analysis is to find resources that perform *similar work* [3], i.e., two people are linked in the social network if they execute similar activities. The more similar their execution profiles are the stronger their relationship. As a result, one can observe that Vincent and Howard execute a set of activities which is disjoint from those executed by all other employees. More precisely, they only execute the activity *Issue payment* and, therefore, might work, e.g., in the *Finance* department of the insurance company. Furthermore, the work of Fred and Linda seems rather similar and quite different from the other three people; they are the only people performing the *Evaluate claim* activity, although they also execute other activities (such as *Send rejection letter* and *Archive claim*). One explanation could be that the activity *Evaluate claim* requires some *Manager* role, whereas all the remaining activities can be performed by people having a *Clerk* role.

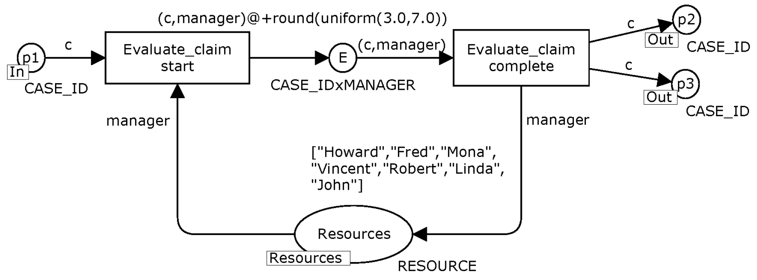


Fig. 11. Sub-page *Evaluate claim* including resource modeling

A simple way to incorporate this information in our simulation model is to create three groups of resources (Finance = {Howard, Vincent}, Manager = {Fred, Linda}, Clerk = {Fred, Linda, John, Robert, Mona}) and to specify for each activity which kind of resource is required (if no particular group has been specified for an activity, it can be performed by any resource). As indicated, this resource classification can be discovered semi-automatically. However, it could also result from some explicit organizational model. Figure 11 depicts how the fact that activity *Evaluate claim* requires the role *Manager* is modeled in the corresponding CPN model. The role is modeled as a separate color set MANAGER, which contains only "Linda" and "Fred". Because the variable *manager* is of type MANAGER, only the resource token "Linda" or "Fred" can be consumed by transition *Evaluate_claim start*. As soon as transition *Evaluate_claim start* is fired, the corresponding resource token resides in the place *E*, i.e., it is not available for concurrent executions of further activities, until transition *Evaluate_claim complete* fires and puts the token back.

4.4 Probabilities and Frequencies

Closely related to the modeling of time aspects is the likelihood of taking a specific path. Both may be of a stochastic nature, i.e., a time duration may be sampled from some probability distribution, and similarly, the selection of an alternative branch may be selected randomly (if there are no data attributes clearly influencing the choice). Hence, the probabilistic selection of a path also needs to be incorporated in the CPN model. Figure 12 shows how often each arc in the model has been used, determined through the log replay analysis carried out by the *Conformance Checker* in ProM[7]. Looking at the first choice it becomes clear that activity *Check policy only* has been executed 7 (out of 10) times and activity *Check all* was performed only 3 times. Similarly, activity *Send rejection letter* happened for 4 (out of 10) cases, while in 6 cases both activity *Send approval letter* and activity *Issue payment* were executed.

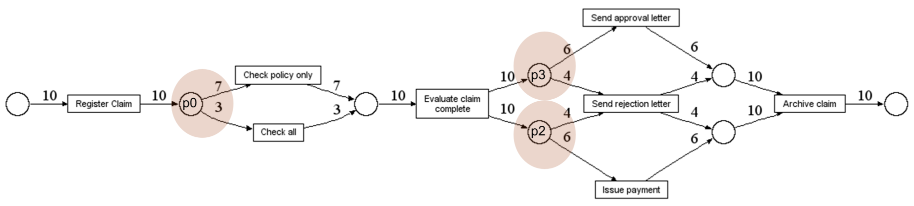


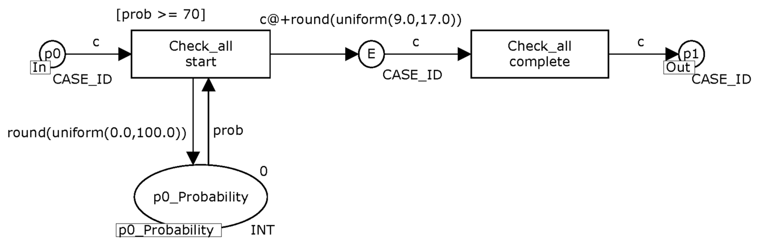
Fig. 12. Frequencies of alternative paths in the example model

In order to reflect frequencies of alternative paths in the simulation model we use two different approaches, depending on the nature of the choice.

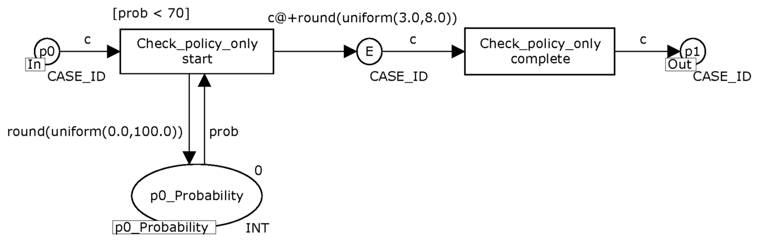
Simple choice The first choice construct in the example model is considered to be a so-called *simple choice* as it is only represented by one single place. We can model such a simple choice using a probability token that is shared among all the activities involved in this choice via a fusion place.

Figure 13 shows this solution for the choice at place *p0*. Both sub-pages *Check all* and *Check policy only* contain a fusion place *p0_Probability* that initially contains a token with the value 0, but after each firing of either transition *Check_all start* or transition *Check_policy_only start* a random value between 0 and 100 is generated. Because of the guard condition, the decision at the place *p0* is then determined for each case according to the current value of the token in place *p0_Probability*. For example, the transition *Check_all start* needs to bind the variable *prob* to a value greater than or equal to 70 in order to be enabled, which will only happen in 30% of the cases.

[7] Note that the place names and the markup of the choices have been added to the diagnostic picture obtained from ProM for explanation purposes.



(a) Sub page *Check all*



(b) Sub page *Check policy only*

Fig. 13. Using a probability token for simple choices

Dependent choices The second choice construct in the example model actually consists of two *dependent choices*[8] (i.e., the choices represented by places *p2* and *p3*) that need to be coordinated in order to either approve or reject a claim. It is clear that two dependent choices cannot be steered properly by two independently generated probability tokens, because the process model will deadlock as soon as the values of the probability tokens indicate contrasting decisions (e.g., the probability token in *p2* indicates a reject while the other probability token in *p3* suggests to approve the claim).

Figure 14 shows a solution for modeling the dependent choices at place *p2* and *p3*. The idea is to increase the likelihood of choosing a certain activity through activity duplication (using the fact that during simulation in CPN Tools all enabled transitions will be fired with an equal probability). This way, the observed relative frequency[9] of the transitions involved in the dependent choices can be incorporated in the simulation model. Figure 14(a) shows an intermediate sub-page for activity *Issue payment*, where three substitution transitions *Issue payment* point to different instances of the same sub-page *Issue payment* (i.e., the actual sub-page is only modeled once). Figures 14(b) and (c) show similar intermediate sub-pages for the activities *Send approval letter* (also duplicated three times) and *Send rejection letter* (duplicated twice).

[8] Similar to the Decision Miner we consider each place as a choice (or decision point) if it contains more than one outgoing arc (cf. Figure 4).

[9] In order to obtain the relative frequency, the absolute frequency is divided by the greatest common divisor (i.e., $6/2 = 3$ and $4/2 = 2$).

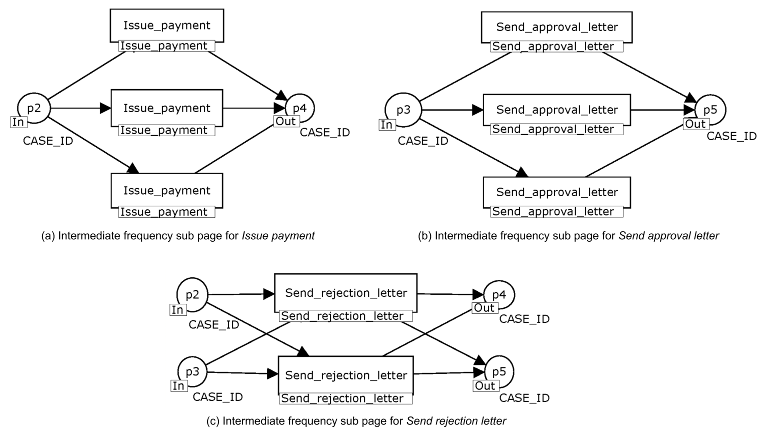

Issue_payment
p2
In
CASE_ID
p4
Out
(a) Intermediate frequency sub page for *Issue payment*
Send_approval_letter
p3
p5
(b) Intermediate frequency sub page for *Send approval letter*
Send_rejection_letter
(c) Intermediate frequency sub page for *Send rejection letter*

Fig. 14. Modeling dependent choices via activity duplication

4.5 Logging and Monitoring Simulation Runs

The CPN models described in this section deal with time, resources, and data. When running a simulation in CPN Tools we are interested in statistics (e.g. average, variance, minimum, and maximum) related to (a) the utilization of resources and (b) the throughput times of cases during the simulation run. We obtain this information via pre-defined monitors as described in the following.

(a) *Resource utilization.* If resources have been specified for the process, all the available resources are contained in a *Resources* fusion place, which is located on the *Environment* page and on every activity sub-page. For obtaining statistics about the resource utilization during the simulation we can define a *marking size monitor* [9] for this Resources fusion place, which records the number of available resources plus the current time (and step) as soon as a resource becomes (un-)-available.

(b) *Throughput time.* If the time perspective is covered, tokens are created with a timestamp. We record the timestamp of each case's creation together with the case ID token that is routed through the process. This way, we can determine the throughput time of a case by defining a *data collector monitor* [9] for the *Clean up* transition on the *Environment* page (cf. Figure 6), which simply calculates the difference between the current model time and the start time of a case[10], and records the throughput time, the end time and end step for each case. Note that—due to recording its creation time—the current run time of a case could be easily determined at any stage in the process via adding some custom monitor.

[10] Because the type of the current model time is infinite integer and in order to not lose precision when calculating the difference between the current model time and the start time of a case, the model time is mapped onto a STRING value, i.e., color set START_TIME is of type STRING and is used to encode infinite integers.

Moreover, we want to generate process execution logs for the business process in the CPN model. This can be very useful for, e.g., the creation of artificial logs that are needed to evaluate the performance of process mining algorithms.

For each firing of a transition on an activity sub-page an event is logged, which includes case ID, the type of transition (i.e., *schedule*, *start*, or *complete*), current timestamp, originator, and additional data (each if available). For generating these process execution logs we use the logging functions that have been described in [10]. However, in contrast to [10]—where the code segments of transitions have to be modified to invoke these logging functions—we decided to use *user defined monitors* [9] in order to clearly separate the logging from the rest of the simulation model.

5 Exporting CPN Models from ProM

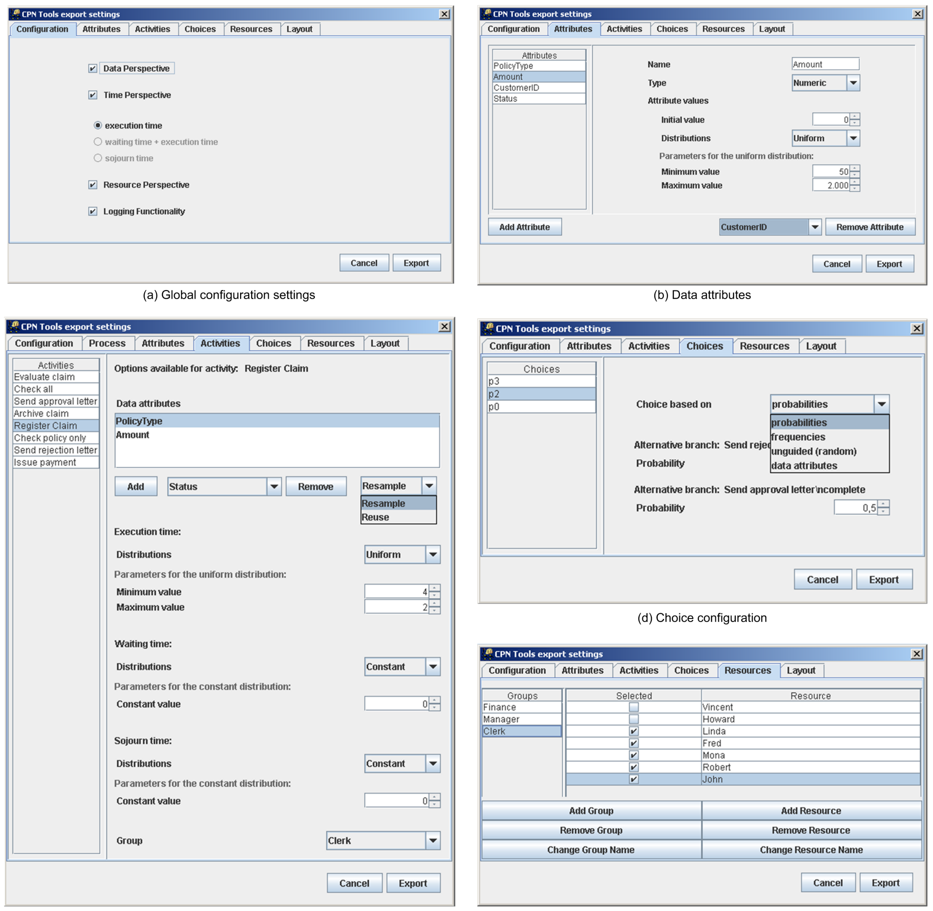
We are able to generate CPN models as presented in the previous section (i.e., including simulation environment and the described monitors) using the *CPN Tools 2.0 Export* plug-in in the ProM framework[11]. It either accepts a simulation model that has been provided by another plug-in in the framework, or a simple, low-level Petri net (in which case an empty simulation model is created and filled with default values). Before the actual export takes place, it allows for the manipulation of the simulation information in the model. As illustrated in Figure 1, we have discovered a process model including the data perspective (provided by the *Decision Miner*), and we can now manually integrate information about further dimensions, such as the performance and organizational dimension.

Figure 15(a) shows the global *Configuration settings*, where the user can choose which dimensions should be included in the generated CPN model. In fact, although the relevant simulation information may be provided, it will be ignored if the corresponding configuration option was not chosen. Note that, since the waiting time of an activity typically results from the unavailability of resources, the explicit modeling of a resource scheme such as in Figure 11 is in conflict with modeling the time dimension including waiting time (cf. figures 10(b) and (c)). Therefore, only the execution time option is available if the resource dimension is selected.

Figure 15(b) depicts the *Attribute settings* of the process. New data attributes can be provided by specifying their name, type (nominal or numeric), possible values (a list of String values for a nominal attribute, and some probability distribution[12] for a numeric one), and initial value. Note that for our example process the available data attributes were already discovered by the Decision

[11] Note that the layout of the generated models was slightly adjusted in order to improve the readability.

[12] The CPN Export plug-in supports all probability distributions currently available in CPN Tools, i.e., constant, bernoulli, binomial, chi square, discrete, erlang, exponential, normal, poisson, student, and uniform.


CPN Tools export settings

Configuration | Attributes | Activities | Choices | Resources | Layout

[✓] Data Perspective

[✓] Time Perspective

- [● execution time]
- ○ waiting time + execution time
- ○ sojourn time

[✓] Resource Perspective

[✓] Logging Functionality

(a) Global configuration settings

CPN Tools export settings

Configuration | Attributes | Activities | Choices | Resources | Layout

Attributes
PolicyType
Amount
CustomerID
Status

Name: Amount

Type: Numeric

Attribute values

Initial value: 0

Distributions: Uniform

Parameters for the uniform distribution:

Minimum value: 50

Maximum value: 2.000

Add Attribute | CustomerID | Remove Attribute

(b) Data attributes

CPN Tools export settings

Configuration | Process | Attributes | Activities | Choices | Resources | Layout

Activities
Evaluate claim
Check all
Send approval letter
Archive claim
Register Claim
Check policy only
Send rejection letter
Issue payment

Options available for activity: Register Claim

Data attributes

PolicyType
Amount

Add | Status | Remove | Resample (Resample, Reuse)

Execution time:

Distributions: Uniform

Parameters for the uniform distribution:

Minimum value: 4

Maximum value: 2

Waiting time:

Distributions: Constant

Parameters for the constant distribution:

Constant value: 0

Sojourn time:

Distributions: Constant

Parameters for the constant distribution:

Constant value: 0

Group: Clerk

(c) Activity settings

CPN Tools export settings

Configuration | Attributes | Activities | Choices | Resources | Layout

Choices
p3
p2
p0

Choice based on: probabilities (probabilities, frequencies, unguided (random), data attributes)

Alternative branch: Send rejec

Probability

Alternative branch: Send approval letter\ncomplete

Probability: 0,5

(d) Choice configuration

CPN Tools export settings

Configuration | Attributes | Activities | Choices | Resources | Layout

Groups
Finance
Manager
Clerk

Selected	Resource
☐	Vincent
☐	Howard
✓	Linda
✓	Fred
✓	Mona
✓	Robert
✓	John

Add Group | Add Resource | Remove Group | Remove Resource | Change Group Name | Change Resource Name

(e) Resource settings

Fig. 15. CPN Tools export settings

Miner. For this particular example it makes sense to delete the *CustomerID* attribute, since it is not involved in any of the discovered data dependencies.

In Figure 15(c), a screenshot of the *Activity settings* for activity *Register claim* is displayed. In this view, the provided data attributes, the execution time, waiting time, sojourn time, and the required resource group may be specified for each of the activities in the process. The data attributes were already provided by the Decision Miner, and we can decide whether the old value of the attribute should be kept (i.e., reuse) or whether a random value will be generated (i.e., re-sample). Furthermore, we can assign some execution time (note the uniform distribution between 2 and 4 time units), and choose the suitable group of resources from the list of groups available in the process (note the *Clerk* role).

Figure 15(d) shows the *Choice configuration* view, where the user can determine for each decision point in the process whether it should be based on either probabilities or frequencies (cf. Section 4.4), or on data attributes, or whether it should not be guided by the simulation model (one of the alternative paths is then randomly chosen by CPN Tools). In Figure 15(d) the probability settings are displayed. For every alternative branch a probability may be provided between 0.0 and 1.0. Before the actual export takes place, each value is normalized by the sum of all specified values, and if the values sum up to 0.0, default values (i.e., equal probabilities for each alternative branch) are used. As discussed in Section 4.4, for dependent choices one should specify a relative frequency value instead. Finally, we can provide a dependency value based on data attributes (in the case of our example process the discovered dependency has already been filled in by the Decision Miner). In the current version of the export plug-in this dependency value is simply a String containing the condition to be placed in the transition guard of the corresponding transition.

In Figure 15(e) the *Resource settings* are depicted. Here, one can add groups and resources, and assign resources to groups. This way, the CPN Export plug-in also supports the specification of information about resources. This information is then used to create the sub-pages shown earlier.

6 Related Work

The work reported in this paper is related to earlier work on process mining, i.e., discovering a process model based on some event log. The idea of applying process mining in the context of workflow management was first introduced in [7]. Cook and Wolf have investigated similar issues in the context of software engineering processes using different approaches [8]. Herbst and Karagiannis also address the issue of process mining in the context of workflow management using an inductive approach [12]. They use stochastic task graphs as an intermediate representation and generate a workflow model described in the ADONIS modeling language. Then there are several variants of the α algorithm [6, 20]. In [6] it is shown that this algorithm can be proven to be correct for a large class of processes. In [20] a heuristic approach using rather simple metrics is used to construct so-called "dependency/frequency tables" and "dependency/frequency graphs". This is used as input for the α algorithm. As a result it is possible to tackle the problem of noise. For more information on process mining we refer to a special issue of Computers in Industry on process mining [5] and a survey paper [4]. However, as far as we know, this is the first attempt to mine process models including other dimensions, such as data. (Note that [3] only considers the social network in isolation and does not use it to provide an integrated view.)

In [11] the authors present a translation of Protos simulation models to CPN Tools. In addition, three types of data collector monitors (measuring the total flow time per case, the waiting time per task, and the resource availability/utilization per resource type), and configuration features enabling the dynamic elimination of unnecessary parts of the process model are generated. Besides the work in [11], we are not aware of further attempts to export business process models to CPN Tools. The work reported in this paper has a different starting point as it is not limited by the simulation information present in a Protos model, but aims at discovering the process characteristics to be simulated from the event logs of real process executions.

7 Future Work

Future work includes the refinement of the generated CPN models. For example, a more realistic resource modeling scheme may allow for the specification of a working scheme per resource (e.g., whether the person works half-time or full-time) and include different allocation mechanisms. Moreover, we plan to apply our approach to real-life data. Finally, the discovery of further perspectives of a business process will be integrated in the mined process models. Currently, we are able to discover data dependencies via the Decision Miner in ProM. But existing plug-ins in ProM deliver also time-related characteristics of a process (such as the case arrival scheme, and execution and waiting times) and frequencies of alternative paths, or organizational characteristics (such as the roles of the employees involved in the process). All these different pieces of aggregate information (discovered from the event log) need then to be combined in one holistic simulation model, which may be exported to CPN Tools, or, e.g., translated to an executable YAWL model [2]. Note that a YAWL model can be used to enact a business process using the YAWL workflow engine. For enactment all perspectives play a role and need to be taken into account. Hence, successfully exporting to YAWL is another interesting test case for the mining of process models with data and resource information.

Acknowledgements

This research is supported by EIT and the IOP program of the Dutch Ministry of Economic Affairs. Furthermore, the authors would like to thank Ton Weijters, Boudewijn van Dongen, Ana Karla Alves de Medeiros, Minseok Song, Laura Maruster, Christian Günther, Eric Verbeek, Monique Jansen-Vullers, Hajo Reijers, Michael Rosemann, Huub de Beer, Peter van den Brand, et al. for their on-going work on process mining techniques. We would also like to thank Lisa Wells and Kurt Jensen for their support in using CPN Tools.

References

1. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis. In J. Grundspenkis and M. Kirikova, editors, *Proceedings of the 5th Workshop on Business Process Modeling, Development and Support (BPMDS'04)*, volume 2 of *Caise'04 Workshops*, pages 138–145. Riga Technical University, 2004.
2. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
3. W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work*, 14(6):549–593, 2005.

4. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
5. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
6. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
7. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
8. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
9. CPN Group, Aarhus. CPN Tools Help. http://wiki.daimi.au.dk/cpntools-help/.
10. A.K. Alves de Medeiros and C.W. Guenther. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In K. Jensen, editor, *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.
11. F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and H.M.W. Verbeek. Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes. Accepted for the Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2006.
12. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
13. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Springer-Verlag, 1997.
14. T. M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.
15. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.
16. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *Business Process Management 2005 Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
17. A. Rozinat and W.M.P. van der Aalst. Decision Mining in Business Processes. BPM Center Report BPM-06-10, BPMcenter.org, 2006.
18. A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *BPM 2006*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer-Verlag, Berlin, 2006.
19. A. Vinter Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In W.M.P. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer Verlag, 2003.
20. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
21. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition.* Morgan Kaufmann, 2005.

Business Process Simulation - A Tool Survey

M.H. Jansen-Vullers and M. Netjes

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{m.h.jansen-vullers, m.netjes}@tm.tue.nl

Abstract. In the nineties, more and more attention was raised for process oriented analysis of the performance of companies. Nowadays, many process aware information systems are implemented (e.g., workflow management systems) and business processes are evaluated and redesigned. The discipline related to this field of study is called Business Process Management (BPM). An important part of the evaluation of designed and redesigned business processes is Business Process Simulation (BPS). Although an abundance of simulation tools exist, the applicability of these tools is diverse. In this paper we discuss a number of simulation tools that are relevant for the BPM field, we evaluate their applicability for BPS and formulate recommendations for further research.

Keywords: Business Process Management, Simulation, Petri nets.

1 Introduction

Business Process Management (BPM) is attracting attention more than a decade now, and its attention is now shifting from the enactment of business processes towards improving business processes. The field of BPM now supports the design, enactment, control, and analysis of business processes [6]. Companies are improving their performance by a constant evaluation of the value added in all parts of their processes. Business processes are in a continuous improvement cycle in which design and redesign play an important role. Various possibilities to change a process are present and the best alternative design should replace the current process. Making an intuitive choice may lead to unpleasant surprises and lower process performance instead of yielding the expected gains. In [16] simulation is mentioned as one of the techniques suitable for the support of redesign. The simulation of business processes helps in understanding, analyzing, and designing processes. With the use of simulation the (re)designed processes can be evaluated and compared. Simulation provides quantitative estimates of the impact that a process design is likely to have on process performance and a quantitatively supported choice for the best design can be made.

Simulating business processes is, to a large extent, overlapping with the simulation of other discrete event systems. In [32] an overview is provided of the steps that are carried out in the context of Business Process Simulation (BPS).

Regarding the simulation of business processes a number of steps can be distinguished.

First the business process is mapped onto a process model, possibly supplemented with process documentation facilities. Then the sub processes and activities are identified. The control flow definition is created by identifying the entities that flow through the system and describing the connectors that link the different parts of the process. Lastly, the resources are identified and assigned to the activities where they are necessary. The process model should be verified to ensure that the model does not contain errors.

Before simulation of a business process, the performance characteristics, such as throughput time and resource utilization, need to be included. For statistically valid simulation results a simulation run should consists of multiple sub runs and each of these sub runs should have a sufficient run length. During the simulation, the simulation clock advances. The simulation tool may show an animated picture of the process flow or real-time fluctuations in the key performance measures. When the simulation has been finished, the simulation results can be analyzed. To draw useful and correct conclusions from these results, statistical input and output data analysis is performed.

Although the steps in BPS will be the same irrespective of the simulation tool used, each simulation tool will have a different applicability. There is an abundance of simulation tools available of which some are applicable to the BPM field. In this paper we discuss several simulation tools taken from three relevant areas: business process modelling, business process management and general simulation tools. We evaluate the modelling, simulation and output analysis capabilities and we aim at providing insights in the advantages and disadvantages of each simulation tool.

The remainder of this paper is organized as follows. First, we discuss related work on evaluation criteria of BPM tools. Then, in Section 3, we describe tools for BPS. The criteria used for the evaluation of BPS tools are listed and explained in Section 4. In Section 5 we compare the described BPS tools, and in Section 6 we present our conclusions.

2 Related work

In this paper, we aim to evaluate several software packages for suitability of BPS. Hardly any package explicitly advertises as BPS tool, however, many of them provide simulation functionality and may be suitable. Bradley *et al* defined seven different categories to evaluate business process re-engineering software tools [8]. The seven categories are as follows:

1. Tool capabilities, including a rough indication of modelling, simulation and analysis capabilities.
2. Tool hardware and software, including, e.g., the type of platform, languages, external links and system performance.

3. Tool documentation, covering the availability of several guides, online-help and information about the learning curve of the tool.
4. User features: amongst others user friendliness, level of expertise required, and existence of a graphical user interface.
5. Modelling capabilities, such as identification of different roles, model integrity analysis, model flexibility and level of detail.
6. Simulation capabilities, summarizing the nature of simulation (discrete vs. continuous), handling of time and cost aspects and statistical distributions.
7. Output analysis capabilities such as output analysis and BPR expertise.

In this paper we elaborate on the categories as defined by Bradley *et al* in the direction of BPS. Especially the last three categories are of interest when evaluating BPS.

With respect to modelling capabilities, the patterns research is used to evaluate the possibility to model various control flow patterns [5], data patterns [26] and resource patterns [27]. The patterns research is used to evaluate the modelling capabilities of a tool with respect to complexity.

The complexity of modern business processes is increasing. In order to manage this complexity, Becker *et al* have formulated six main quality criteria for business process models [7]. These criteria are:

1. Correctness, the model needs to be syntactically and semantically correct.
2. Relevance, the model should not contain irrelevant details.
3. Economic efficiency, the model should serve a particular purpose that outweighs the cost of modelling.
4. Clarity, the model should be (intuitively) understandable by the reader.
5. Comparability, the models should be based on the same modelling conventions within and between models.
6. Systematic design, the model should have well-defined interfaces to other types of models such as organizational charts and data models.

Many authors have proposed requirements for business process modelling tools (for example [10, 24, 31, 35]) or have tested these requirements empirically [11, 16]. Although this requirement building frequently took place in the context of BPS only one explicit list with evaluation criteria for simulation or output analysis capabilities is present. Law and Kelton describe desirable software features for the selection of general purpose simulation software [19]. They identify the following groups of features:

1. General capabilities, including modelling flexibility and ease of use.
2. Hardware and software considerations.
3. Animation, including default animation, library of standard icons, controllable speed of animation, and zoom in and out.
4. Statistical capabilities, including random number generator, probability distributions, independent runs (or replications), determination of warm up period, and specification of performance measures.

5. Customer support and documentation.
6. Output reports and plots, including standard reports for the estimated performance measures, customization of reports, presentation of average, minimum and maximum values and standard deviation, storage and export of the results, and a variety of (static) graphics like histograms, time plots, and pie charts.

3 Tools for Business Process Simulation

Many software tools exist to simulate processes. When simulating business processes, some specific requirements are applicable. The nature of the business process requires sufficient modelling power of the tool. When particular choices or a synchronization cannot be implemented, the simulation result loosens its strengths. On the other hand, simulation of business processes aims to support process owners or process managers. When the tool or the simulation output can hardly be understood by the client, the tool overreaches itself. In this section, we describe three different categories of software tools that may be applicable for BPS:

– business process modelling tools,
– business process management tools,
– general purpose simulation tools.

For each type a general introduction and the description of two specific tools are given.

3.1 Business process modelling tools

Business Process Modelling tools are developed to describe and analyze business processes. The analysis part may provide data useful for the management of these processes. The tool supports the process to establish the control flow of business processes, the resource roles involved, documents being used and it documents instructions for the execution of steps in the business process. As a result, reports can be generated for process documentation, manuals, instructions, functional specifications, etc. For the evaluation of the simulation functionality we consider two different tools, one based on Petri Nets (Protos) and one based on Event-driven Process Chains (ARIS Toolset).

Protos
Protos is a modelling and analysis tool developed by Pallas Athena and it is mainly applied for the specification of in-house business processes. Protos is suitable to model well-defined Petri Net structures. Nevertheless, it also permits free hand specifications of business processes without formal semantics, e.g. to support initial and conceptual modelling [34]. When formal Petri Net semantics have been applied, translation to various other process-based systems is feasible as well, e.g. to the workflow management system COSA and the workflow analyzer Woflan [33].

The main use of Protos is to define models of business processes as a step towards either the implementation of quality management systems, the redesign of a business process, communication enhancement between process stake holders or the implementation of workflow management systems. The process can be analyzed with respect to data, user and control logic perspective, and by making use of simulation [34].

The simulation engine is implemented in Protos version 7.0. The existing engine of the Petri Net based tool ExSpect has been integrated in the Protos environment and it facilitates the simulation of the business process as has been specified in the Protos model before. In addition to the standard process specification, simulation data can be added for tasks, connections and resources such as the (stochastic) processing time and the number of resources required. Furthermore, process characteristics are added such as the arrival pattern for cases and the number and length of simulation runs. The simulation result can be obtained from an Excel spreadsheet and includes mean and 90% and 99% confidence interval of utilization rates, waiting times, service times, throughput times and costs.

ARIS
ARIS Simulation is a professional tool for the dynamic analysis of business processes. It is an integral part of the ARIS Toolset; processes recorded in the ARIS Toolset are used as the data basis for business process simulation. ARIS Toolset is developed by IDS Scheer AG (see www.ids-scheer.nl) and can be classified as an enterprise modelling tool with a strong emphasis on business processes. Enterprise modelling is supported by a number of different views (process, function, data, organization and product) and the modelling approach called ARIS House [30, 29].

The process modelling part supports the definition of business processes represented in Event-driven Process Chains (EPCs). Other modelling techniques supported in the ARIS House are, e.g. value chains (also to model the control flow), organization charts (to model relationships between resources), EPCs and and function allocation diagrams (for supplementary information such as data and systems). The simulation functionality shows whether the specified processes are executable at all and it answers questions about throughput times and utilization levels of the resources, etc.

When starting a simulation, the simulation module of the tool is started and the model is transferred. The simulation toolbar shows buttons for start and stop, one time step and simulation steps and options for animations. The simulation results are available in Excel spreadsheets and include statistics on events, functions, resources, processes and costs. Only raw data is available.

3.2 Business process management tools

Business process management (BPM) systems can be seen as successors of Workflow Management (WFM) systems. The core functionality of a WFM system is automating the “flow of work”. With the introduction of BPM the functionality is broadened to support the whole proces life-cycle. BPM is defined as *Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information* [6]. Some BPM tools offer a simulation tool to support the design phase. Most BPM tools, however, do not provide simulation facilities and we use FLOW*er* as a representative of this group of BPM tools. Further, we will evaluate FileNet, one of the most advanced BPM tools. FileNet is evaluated to show what most likely will be the best simulation functionality provided by a BPM tool. The FLOW*er* tool is evaluated, regardless his lack of simulation facilities, to illustrate the other end of the simulation spectrum of BPM tools.

FLOW*er*

FLOW*er* is a flexible, case-based BPM system. When handling cases the system only prevents actions for which it is specified that these are not allowed. This results in a flexible process where activities for a case can be executed, skipped or redone.

The FLOW*er* systems consists of a FLOW*er* Studio, FLOW*er* Case Guide, FLOW*er* CFM (Configuration Management), FLOW*er* Integration Facility, and FLOW*er* Management Information and Case History Logging.

The graphical design environment, Studio, is used to define processes, activities, precedences, data objects and forms. Work queues are used to provide work to users (defined with CFM) and to find cases satisfying specified search criteria. Case Guide is the client application which is used to handle individual cases. FLOW*er* Integration Facility provides the functionality to interface with other applications. FLOW*er* Management Information and Case History Logging can be used to store and retrieve management information at various levels of detail.

BPM systems, like FLOW*er*, focus on the configuration of the system, and the execution and control of the workflow. Additional features like the FLOW*er* Management Information and the FLOW*er* Integration Facility are provided. However, FLOW*er* does not provide explicit simulation or output analysis functionality. We will not be able to evaluate the simulation and output analysis capabilities of FLOW*er*, but we can evaluate the modelling capabilities [3, 5].

FileNet

FileNet is considered to be one of the leading commercial BPM systems[1]. We have evaluated the strengths and weaknesses of the FileNet P8 BPM Suite and its ability to support the various parts of the process life-cycle [23].

The FileNet system includes a FileNet Process Designer, a FileNet Process Simulator, a FileNet Process Engine, a FileNet Process Administrator, and a FileNet Analysis Engine.

First, a process structure is modelled graphically with the Process Designer and tasks are assigned to work queues. These work queues and the associated users are created outside the Process Designer. Then, the created process defini-

[1] www.gartner.com

tion is feeded to the Process Engine to start the execution of the workflow. The execution data for individual cases is logged by the Process Engine and can be accessed with the Process Administrator. Further, execution data is aggregated and parsed to the Analysis Engine. Reporting and analysis of the aggregated data is facilitated by twenty out-of-the-box reports; each graphically presenting the data related to one performance indicator.

The Process Simulator in FileNet can be used to evaluate the performance of a created design. The Process Simulator is a separate tool, which can partly import the created process definition. Other parts of the process definition have to be reentered. Simple arrival patterns of cases are defined, i.e. a fixed number of cases arrives at fixed time points. Also historic, execution arrival data can be used. Other performance characteristics should be added manually and can only have constant values. After simulation an animation and a summary of the simulation results are provided. Simulation data can also be presented in Excel reports. However, performing what-if analysis (comparing scenarios) is not possible.

3.3 General purpose simulation tools

Simulation tools may be tailored towards a specific domain, such as logistics (e.g., Enterprise Dynamics) or SPEEDES in the military domain. In this section we consider simulation tools that are not tailored towards a particular domain and we evaluate their suitability for the domain of business processes. The first tool, Arena, has an industrial background and shows industry successes in manufacturing, supply chain management, military/defense, health care, contact centers and process reengineering (see www.arenasimulation.com). The second tool, CPN Tools, has been developed in a university environment and has been applied in more technical engineering domains (see http://www.daimi.au.dk/CPnets/).

Arena
Arena is a general purpose simulation tool developed by Rockwell Automation. The Arena product family consists of a Basic Edition for uncomplicated processes and a Professional Edition for more complex large scale projects in manufacturing, distribution, processes, logistics, etc. The Professional Edition also provides (and allows definition of) templates for complex repetitive logic, e.g., for packaging and contact centers.

When opening the tool, a number of process panels are available, e.g., for basic and advanced processes and for reporting. The model can be created by drag and drop from the process panel to the model window. By double-clicking on the icons, options for the different building blocks can be set such as delay types, time units and the possibility to report statistics. Many more building blocks are available and can be attached when necessary.

When a model has been created and is completely specified (from the Arena viewpoint) and it is syntactically correct, it can be simulated. Warm-up and cool-down periods can be specified, as well as run length and confidence intervals.

Several statistics are provided by default, but the larger part needs to be added manually by adding record building blocks where necessary [15].

In a previous study, de Vreede et al considered the suitability of Arena to simulate business processes [35]. They stated that a weak point in simulating business processes is the time consuming and complicated process to create simulation models. They took advantage of the possibility to develop their own template with predefined building blocks, which they considered to be successful in several simulation studies they carried out.

CPN Tools

CPN Tools is developed by the computing science group of Aarhus University in Denmark. CPN Tools is a tool for editing, simulating and analyzing Colored Petri Nets. The tool attracts attention with respect to its user interface which has been designed in cooperation with leading HCI experts, and includes a number of novel interaction mechanisms such as the use of two-handed input by means of a mouse and a trackball. During editing a net (a process model), feedback facilities provide contextual error messages and indicate dependency relationships between net elements. The tool features incremental syntax checking and code generation which take place while a net is being constructed. A fast simulator efficiently handles both untimed and timed nets. Untimed nets are generally not applicable for modelling and simulation of (realistic) business processes, but several earlier projects already showed that timed CP-nets can model business processes [20, 13, 22]. Correctness of the developed model can be researched by existing Petri Net techniques such as the generation of state spaces and the analysis of boundedness and liveness properties, which are all implemented in CPN Tools.

The industrial use of CPN Tools (and its predecessor Design CPN) can be found starting from the home page (http://www.daimi.au.dk/CPnets/). Design CPN is the most widespread software package for modelling and analysis by means of Colored Petri Nets. The overview shows a wide variety of mainly technical domain areas such as protocols and networks, hardware and control systems. Also some projects are listed with a more business oriented focus, though these are exceptions.

4 Evaluation criteria for BPS Tools

When evaluating BPS tools, the modelling, simulation and output analysis capabilities of the tool are important. In this section we present our view on these capabilities and specify criteria to evaluate each capability in detail.

4.1 Modelling capabilities

The purpose of the modelling capabilities criteria is to evaluate how well and how precise a business process can be represented. The modelling evaluation criteria are:

- Ease of model building [8, 19]
  Model building should be easy to allow users to be involved in the modelling of their processes. A graphical user interface with predefined business objects which can be dragged and dropped facilitates the model building. The hard coding of process parts is hard to perform or understand for users and should be avoided.

- Formal semantics [4] and verification of correctness [21, 33]
  Formal semantics provide a precise and unambiguous description of the behavior of the modelled process. Van der Aalst concludes that many modelling techniques lack formal semantics and thus powerful analysis methods and tools [4]. In [2] he summarizes three good reasons for using a Petri-net based workflow management system which appear to be critical in large BPM projects. These reasons are: (1) the existence of formal semantics despite the graphical nature, (2) the state based diagrams instead of event based diagrams (as can be encountered in many workflow products) and (3) the abundance of analysis techniques.

- Workflow patterns [5]
  The workflow patterns [5], or control flow patterns, are used to evaluate the expressive power of modelling languages. The patterns identify both basic and complex modelling constructs. The number of supported patterns indicate how well a modelling language can give a good representation of the actual business process.

- Resource and data perspective [8, 26, 27]
  The process model should include the resource and data perspective and not just the process structure to provide a good representation of the real world situation. Resource and data patterns capture the various ways in which respectively resources and data are represented and utilized in processes [26, 27].

- Level of detail, transparency and suitability for communication [7, 8]
  Both senior management as well as end users need to be informed about the process (alternatives), they shouldddddd be able to validate the model and should be able to make decisions based on these models. These stake holders have a different need for information, senior management wants a high level overview, while the end users need detailed work descriptions. Through the use of, for instance, hierarchical layers processes are be modelled in detail, but without loosing overview.

4.2 Simulation capabilities

The purpose of the simulation capabilities is to evaluate in which way a simulation can be carried out and which parameter settings can be made. The simulation evaluation criteria are:

- Performance dimensions [8, 25, 19]
  A simulation model should incorporate the performance dimensions one is interested in. In most cases it should be possible to simulate several different time and/or costs aspects. Other relevant performance dimensions are quality and flexibility [25].

- Distributions [19, 24]
  The average performance of a simulated process may seem fine while in real life many problems would occur because of its variability. Queues may be empty at some moments and overloaded at other moments, creating employee and customer dissatisfaction [24]. Taking into account the distributions of performance characteristics will not only show the average behavior of the process, but also its extremities.

- Animation [19, 35]
  With simulation not only the final simulation results but also the simulation itself can give useful insights in the simulated process. A replay or animation of the simulation will show the states the simulation model has been in during simulation. This visualization might reveal bottlenecks and other problems in the execution of the process.

- Scenarios [14, 18]
  With the use of scenarios the consequences of changes can be investigated. While the process stays the same, different configurations of the simulation model reflect potential changes in, i.e., the arrival pattern or resource availability. With the use of scenarios the effects of changes can be predicted and counter measures can be taken to avoid bad performance once the change occurs in reality.

4.3 Output analysis capabilities

The output analysis capabilities aim to evaluate the outcome of a simulation, which data can be analyzed and which representation styles are provided. The output analysis evaluation criteria are:

- Statistics [17, 19]
  Simulation should provide statistically proper results and it should be clear how these results are calculated. Simulation settings (e.g. simulation length, number of replications, start and stop conditions [9, 24]) should be indicated to or even better be set by the user. A random generator should be used for the generation of cases. For each performance measure not only the mean, but also the standard deviation and confidence intervals should be presented.

- Format [8, 19]
  The tool should have an easy to read format for the presentation of the re-

sults and possibilities for animation, storing and reuse of results.

- What-if analysis [18, 9, 14]
  Before a process design is chosen what-if analysis is performed. In this analysis different scenarios (of the same simulation model) are compared. The comparison of confidence intervals of a performance measure shows which scenarios perform significantly better than others on this measure. It also indicates under which conditions a certain process design will perform within its requirements and under which conditions a performance level can not be reached.

- Conclusion-making support [8, 35]
  Conclusion-making support facilitates the interpretation of the simulation results. Useful support is the identification of trends, the slicing and dicing of data and the tracking of the cause of specific outcomes.

5 Comparison of BPS Tools

In Section 3 we described six different tools which may be applicable for BPS, and which have been developed from various viewpoints: process modelling, process execution and simulation. In Section 4 we developed a framework with a set of evaluation criteria to find strengths and weaknesses of these tools. In this section, we report our findings. We will score the BPS tools for each of the evaluation criteria ranging from good (++) and neutral (+/–) to bad (– –).

5.1 Modelling capabilities

In this section we evaluate how well and how precise a business process can be modelled in the tools. We provide a short overview per tool and at the end of the section we summarize the findings in Table 1.

Protos
The control flow of a business process and the resources can very easily be specified in Protos, as may be expected from a process modelling tool. Also the data perspective and instructions for the execution of tasks can be specified. The tool allows freehand specifications, however it also allows well-defined Petri Net structures, thus opening possibilities for further verification (e.g., in Woflan) and analysis (based on the ExSpect tool). The application of sub models allows for a transparent process model and handling of resources which can very well be communicated with process owners. Points for improvement are the possibility to assign different roles to one task and to specify part time work and overtime. This could be specified, e.g., in histograms (which can already be handled by the simulation engine but is not (yet) allowed in the Protos interface).

ARIS

The control flow part is being modelled in EPCs. This is an informal modelling language, and the simulation relies on the given semantics when the EPC language has been implemented in the ARIS Toolset. It appears that these semantics are not completely clear, which may result in unforseen behavior when using (X)OR connectors. The models can be conveniently arranged, has functional use of colors for different model elements and supports hierarchy. Due to the informal language, several workflow patterns cannot be modelled conveniently. Model verification is not supported by the tool.

FLOW*er*

With FLOW*er* it is, on the one hand, possible to handle exceptions and on the other hand, to force a sequential order handling. Due to this flexibility FLOW*er* supports most of the workflow patterns. FLOW*er* is data driven, giving it a strong data perspective and also the resource perspective is taken into account. Both the process and role graph can be modelled in several layers of detail.

FileNet

Most BPM tools, including FileNet, use a simple graphical representation of process models without formal semantics and verification of correctness. With this, users can create and discuss process models without difficulties. More advanced workflow patterns and also the resource and the data perspective need to be hard coded in FileNet.

Arena

Arena models can be created very easily, though to specify exactly those things you would like to model is more difficult. When browsing through a model, the level of detail is very convenient, due to the use of sub models and the fact that many details are hidden in the icon properties. When creating models, good knowledge about all necessary building blocks and their exact specification is required. Frequently used control flow patterns are supported, but some more advanced patterns require a bit more indirect modelling [12].

CPN Tools

The tool is based on Petri Net modelling techniques, and both benefits and suffers from this property: it has formal semantics, allows for most control flow patterns [5] and can be verified, but the price to be paid is that the models may be quite detailed and technical.

This level of detail is required to model resource handling and corresponding timing aspects, which is crucial in most business process models. Also, some constructs can only be modelled indirectly, thus resulting in model parts that can hardly be understood by business process owners. As a result, models cannot be built easily. Though very powerful, the Petri Net formalism appears to be more difficult to understand than informal modelling languages [28].

In Table 1 our score for the modelling capability criteria for each of the tools is presented.

Table 1. Modelling capabilities

Feature	Protos	ARIS	FLOW*er*	FileNet	Arena	CPN Tools
Ease of model building	++	+	+	+	+	− −
Formal semantics/verif.	+	−	− −	− −	+/−	++
Workflow patterns	+	−	+	+/−	+	+
Resources and data	+	++	++	+/−	+	+/−
Level of detail	++	++	++	+	++	− −

5.2 Simulation capabilities

In this section, we evaluate in which way a simulation can be carried out and which parameter settings can be made. We provide a short overview per tool and at the end of the section we summarize the findings in Table 2.

Protos
The simulation engine in Protos seems to be working fine. A more detailed look, however, reveals some weaknesses of the simulation. Apparently, these weaknesses seem to be introduced by the interface between Protos and ExSpect as the simulation engine of ExSpect itself does not suffer from this. The suggestion of the Protos/ExSpect simulation tool is that all data specified in the process, task and resource properties are taken into account in the simulation. It appeared that this is not the case for the number of resources and the data required for a task. As a result, decisions in the process cannot be made based on data (but instead a probability is calculated based on the weight of outgoing arcs or follow-up tasks). In addition, problems may occur when using subprocess; in some cases an OR-split can be changed into an AND-split (though this seems to be a bug instead of a design issue).

All important (standard) performance dimensions are predefined, but it is not possible to add any other dimension. The same holds true for the possible distributions. The most well-known distributions are available but these cannot be extended. In the future, distributions based on histograms may be provided to be more flexible in this aspect. Facilities for animation and scenarios are not available.

ARIS
Before running a simulation, several simulation parameters need to be set: average processing times and distributions, number of cases being generated, case arrival distribution and probabilities of outgoing arcs from XOR-split connectors. It is possible to use animation during the simulation and an animation icon

can be selected. ARIS is based on an informal process modelling language. Since the simulation models can be executed, a semantic is chosen for constructs which leave room for interpretation, i.e. (X)OR splits and joins. An example of this is the choice for a waiting time for incoming branches: if the waiting time has been exceeded, it is assumed that the data that has arrived already will be processed and that no other data will reach the connector for this particular case. It is unclear what exactly happens beneath the surface.

FLOW*er*
Most BPM tools, including FLOW*er*, only provide the possibility to test or play with the workflow by launching some cases and execute them manually. In this sense the workflow engine is used as a runtime simulation engine. This, however, does not provide explicit simulation functionality.

FileNet
After simulation with FileNet the flow of cases can be replayed in an animation. Both time and costs aspects are taken into account but without fluctuations because only constant performance measures are used in the simulation. It is possible to create scenarios of a simulation model, but it is not possible to change the process structure in the process simulator itself.

Arena
In Arena a model can be simulated by pressing the go-button in the toolbar. The model then enters the simulation mode and cannot be edited anymore. The simulation can be done step-by-step and in normal and fast-forward modes. All performance dimensions and frequently used distributions can be added on those places necessary in the model. Animations are obtained by icons flowing through the model or 3D animations (in a post-processing tool). Alternative models can be defined and evaluated in the Process Analyzer.

CPN Tools
CPN Tools has been developed for simulation purposes, and this shows in the simulation capabilities. When a model (part) has been created, it can be simulated directly, making use of a step-by-step simulation, or a chosen number of steps. All performance dimensions can be measured in the monitoring part of the tool. A number of standard monitors are pre-programmed, but most monitors need to be programmed manually. Animation facilities are not available in the standard tool, but an additional tool (BRITNeY) aims at building and deploying visualizations of Colored Petri Net Models, see e.g. [37]. Scenarios can be implemented quite easily by creating model versions with adapted model parameters.

In Table 2 our score for the simulation capability criteria for each of the tools is presented.

Table 2. Simulation capabilities

Feature	Protos	ARIS	FLOW*er*	FileNet	Arena	CPN Tools
Performance dimensions	− −	++	− −	+	++	++
Distributions	+	+	− −	− −	++	++
Animation	− −	+	− −	+	++	+
Scenarios	− −	−	− −	+/−	+	+

5.3 Output analysis capabilities

In this section, we evaluate how well the simulations statistically can be carried out, how well they match the situation in real life and how the user is supported in the evaluation of the simulation results. We provide a short overview per tool and at the end of the section we summarize the findings in Table 3.

Protos
The simulation results are made available in a very basic spreadsheet, but all important performance dimensions are listed and supplemented with means and 90% and 99% confidence intervals. However, depending on the data specified in the process model, the simulation results may be incorrect (see 5.1).

ARIS
The output format is (a set of) Excel spreadsheets, with raw detailed and/or cumulative data. Statistics need to be calculated manually and support for what-if analysis and scenarios is not directly available in the tool. ARIS Toolset however has a good interface with other ARIS tools which can provide these, e.g., ARIS Process Performance Manager or ARIS Business Optimizer.

FLOW*er*
Most BPM tools, including FLOW*er*, do not provided simulation functionality and output analysis functionality.

FileNet
The first impression of the performance reports provided by FileNet is a good one. Nice graphics are shown for different performance indicators and more detailed views are easy to realize. However, a closer look shows that it is unclear what is presented and how the performance indicator should be interpreted. It is hard to come to conclusions and there are only averages presented. It is impossible to view the results of one scenario or to compare scenarios, because the results for all scenarios for a certain simulation model are aggregated.

Arena
Arena provides standard statistics for all performance indicators specified. For each statistic, the minimum and maximum value is given, as well as mean and half length of the 95% confidence interval. When a simulation has run to com-

pletion, you can see the results in a standard report, it can be analyzed later in the output analyzer (in the advanced process panel) or it can be written to an Excel file (by inserting the read-write module). Conclusion making support is provided in the process analyzer.

CPN Tools
Strong point of the tool is the statistically correct output of the simulation. All aspects specified in the process model are taken into account, thus resulting in good simulation results. The standard output format gives 90, 95 and 99% confidence intervals. In addition other confidence intervals can be calculated making use of the raw simulation data. Weak point of the tool is the lack of support when drawing conclusions on the simulations. The output is provided on a html-page and any further processing should be done manually, e.g. when comparing different scenarios.

In Table 3 our score for the output analysis capability criteria for each of the tools is presented.

Table 3. Output analysis capabilities

Feature	Protos	ARIS	FLOW*er*	FileNet	Arena	CPN Tools
Statistics	–	– –	– –	–	++	++
Format	–	+/–	– –	+/–	+	+/–
What-if analysis	– –	+	– –	– –	–	–
Conclusion-making support	– –	+	– –	–	+	– –

6 Conclusion

In this paper we considered a number of software tools on their suitability for BPS. The tools have been evaluated on their modelling capabilities, simulation capabilities and possibilities for output analysis. The tools were selected for different reasons. Protos and ARIS were selected because of their strong background in process modelling. The modelling power, transparency for business users and the ability to model data and resource perspectives met our expectations. Filenet and Flower were selected because of their usage in business process management, i.e. their strong support of workflow processes. Filenet and Flower appeared to be strong in this respect. Finally, CPN Tools and Arena were selected because of their excellent track record in simulation. Both tools performed well on this aspect.

The above mentioned tools, however, were not only evaluated on their respective "known" strong points, but of course also on all other aspects relevant when modelling and simulating business processes. Both business process management tools fel short on their simulation capabilities; Flower did not support simulation at all (like most business process management tools) and Filenet did support simulation though without stochastic functions and statistical analysis. The process modelling tool Protos provides a simulation module based on the ExSpect simulation engine. However, the interface between the two modules omits important details with respect to data and resources, thus making the outcome of a simulation unreliable. Flower, Filenet and Protos are considered to be unsuitable for solid BPS studies.

The three remaining tools, ARIS, Arena and CPN Tools, all three qualify for BPS studies. These tools have different principles that determine the suitability of the tool for a particular simulation study. ARIS is based on the informal process modelling language of EPCs and has difficulty to model workflow patterns. However, its strong point is the suitability for communication with process owners, which frequently is an important condition in such simulation studies. Arena is a strong simulation tool that proved to be appropriate for BPS. The modelling with this tool is based on predefined building blocks, which can be adapted and extended if necessary. In this tool, it is important to have a profound knowledge about the building blocks that are available and about the exact mode of operation. Finally, CPN Tools is based on the formal modelling techniques of Petri Nets. This opens many possibilities for the formal verification of the simulation model. The price to be paid however, is high. Like modelling in Arena, a profound knowledge is required on modelling Petri Nets, but CPN Tools differs from Arena in that respect that the resulting models are hard to understand by general process owners who should be able to understand and validate the models.

7 Future work

In our research, e.g. on the quantification of business process redesign heuristics, we benefit a lot from the formal verification techniques. Based on this and the results of the evaluation, we choose CPN Tools as a basis for further development. Further BPS research points towards elaboration on CPN Tools. We foresee two possible directions: (1) making the process of modelling business processes easier and (2) making simulation output more transparent for business process owners.

As future work in the direction of process modelling, we consider the development of a library of building blocks dedicated for business process modelling. This library may cover, for instance, resource handling, some timing aspects and statistical output analysis. As a starting point we will consider the strong points of ARIS and Arena as described above and the work previously done in the Petri net community, e.g., the development of the ExSpect libraries [1].

In ExSpect, for example, building blocks have been defined dedicated for e.g., logistic analysis. A difference between CPN Tools and ExSpect, however, is the fact that the logic of an ExSpect transition is hidden in the transition whereas the logic of a CPN Tools transition is derived from the logic on input and output arcs. Furthermore, ExSpect knows a strong separation of the definition of a transition and its actual installation. As a result, it is more straightforward to define a library of ExSpect transitions than a library of CPN Tools transitions. A possible solution can be found in the creation of subpages in CPN Tools. Each subpage represents a particular building block, that can be applied in a new CP net. For instance we consider a resource building block as an important component [22]. The number of input and output arcs, as well as the transition logic of the corresponding substitution transition in the CP net is defined. Apart from the selection and definition of such building blocks, we should also consider how to create a library of building blocks/subpages and how to call a building block from a library.

Future work in the direction of simulation output includes visualization of the simulation: tokens are moving over the arcs during the processing of a task instead of being consumed and produced with a delay. Furthermore, output statistics are captured both statically and dynamically, e.g. in performance dashboards, e.g. as in Arena or ExSpect. Finally, we will consider the addition of what-if analysis support to compare several different scenario's. Recent developments in the BRITNeY Suite [36, 37] will also be considered here.

Acknowledgement

This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst. Modelling and Analysis of Complex Logistic Systems. In *Proceedings of the IFIP WG 5.7 Working Conference on Integration in Production Management Systems*, pages 203–218, Eindhoven, the Netherlands, 1992.
2. W.M.P. van der Aalst. Three Good Reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, Nov 1996.
3. W.M.P. van der Aalst and P.J.S. Berens. Beyond Workflow Management: Product-Driven Case Handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51. ACM Press, New York, 2001.
4. W.M.P. van der Aalst and K.M. van Hee. Business Process Redesign: A Petri-net-based approach. *Computers in Industry*, 29(1-2):15–26, 1996.
5. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
6. W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2003)*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, Berlin, 2003.

7. J. Becker, M. Kugeler, and M. Rosemann. *Process Management - A guide for the design of business processes.* Springer-Verlag: Berlin, 2003.
8. P. Bradley, J. Browne, S. Jackson, and H. Jagdev. Business Process Reengineering (BPR)– A study of the software tools currently available. *Computers in Industry*, 25(3):309–330, 1995.
9. M.A. Centeno and M.F. Reyes. So you have your model: What to do next. A tutorial on simulation output analysis. In D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 23–29. IEEE Computer Society Press, Los Alamitos, 1998.
10. T.H. Davenport. *Process innovation : reengineering work through information technology.* Harvard Business School Press, Boston, 1993.
11. V. Hlupic and S. Robinson. Business process modeling and analysis using discrete-event simulation. In D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 1363–1369. IEEE Computer Society Press, Los Alamitos, 1998.
12. M.H. Jansen-Vullers, R. IJpelaar, and M. Loosschilder. Workflow patterns modelled in arena. Technical Report BETA Working Paper Series, WP 176, Eindhoven University of Technology, The Netherlands, 2006.
13. M.H. Jansen-Vullers and H.A. Reijers. Business Process Redesign at a Mental Healthcare Institute: A Coloured Petri Net Approach. In K. Jensen, editor, *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (PB-576)*, pages 21–38, Department of Computer Science, University of Aarhus, Oct. 2005.
14. W.D. Kelton. Analysis of Output Data. In J.D. Tew, S. Manivannan, D.A. Sadowski, and A.F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 62–68. Society for Computer Simulation International, San Diego, 1994.
15. W.D. Kelton, R.P. Sadowski, and D.T. Sturrock. *Simulation with Arena.* McGrawHill, 2004.
16. W.J. Kettinger, J.T.C. Teng, and S. Guha. Business Process Change: A Study of Methodologies, Techniques, and Tools. *MIS Quarterly*, 21(1):55–80, 1997.
17. J.P.C. Kleijnen and W.J.H. van Groenendaal. *Simulation: a statistical perspective.* Wiley, Chichester, 1992.
18. M. Laguna and J. Marklund. *Business Process Modeling, Simulation, and Design.* Pearson Prentice Hall, New Jersey, 2005.
19. A.M. Law and W.D. Kelton, editors. *Simulation modeling and analysis.* McGraw-Hill, New York, 2000.
20. D. Makajic-Nikolic, B. Panic, and M. Vujosevic. Bullwhip Effect and Supply Chain Modelling and Analysis using CPN Tools. In K. Jensen, editor, *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (PB-570)*, pages 219–234, Department of Computer Science, University of Aarhus, Oct. 2004.
21. J. Mendling, M. Moser, G. Neuman, H.M.W. Verbeek, B.F. van Dongen, and W.M.P. van der Aalst. A quantitative analysis of faulty EPCs in the SAP Reference Model. In *Proceedings of the fourth International Conference on Business Process Management (BPM 2006)*, page (to appear), Vienna, Sept. 2006.
22. M. Netjes, W.M.P. van der Aalst, and H.A. Reijers. Analysis of Resource-Constrained Processes with Colored Petri Nets. In K. Jensen, editor, *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (PB-576)*, pages 251–265, Department of Computer Science, University of Aarhus, Oct. 2005.

23. M. Netjes, H.A. Reijers, and W.M.P. van der Aalst. Supporting the BPM life-cycle with FileNet. In T. Latour and M. Petit, editors, *Proceedings of the CAiSE'06 Workshops and Doctoral Consortium*, pages 497–508, Luxembourg, June 2006.
24. R.J. Paul, G.M. Giaglis, and V. Hlupic. Simulation of Business Processes. *The American Behavioral Scientist*, 42(10):1551–1576, 1999.
25. H. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*, volume 2617 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.
26. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow data patterns. Technical Report QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
27. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow resource patterns. Technical Report BETA Working Paper Series, WP 127, Eindhoven University of Technology, The Netherlands, 2004.
28. K. Sarshar and P. Loos. Comparing the control-flow of epc and petri net from the end-user perspective. In *Business Process Management*, pages 434–439, 2005.
29. A.W. Scheer. *ARIS: Business Process Frameworks*. Springer-Verlag, Berlin, 1998.
30. A.W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 1998.
31. K. Spurr, P. Layzell, L. Jennison, and N. Richards, editors. *Software assistance for business re-engineering*. Wiley, Chichester, 1993.
32. K. Tumay. Business Process Simulation. In J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, editors, *Proceedings of the 1996 Winter Simulation Conference*, pages 93–98. ACM Press, 1996.
33. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing workflow processes processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
34. H.M.W. Verbeek, M. van Hattem, H.A. Reijers, and W. de Munk. Protos 7.0: Simulation Made Accessible. In G. Ciardo and P. Darondeau, editors, *International Conference on Application and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 465–474. Springer-Verlag, Berlin, 2005.
35. G.J. de Vreede, A. Verbraeck, and D.T.T. van Eijck. Integrating the Conceptualization and Simulation of Business Processes: A Modelling Method and an Arena Template. *SIMULATION*, 79(1):43–55, 2003.
36. M. Westergaard. BRITNeY Suite: Experimental Test-bed for New Features for CPN Tools. http://wiki.daimi.au.dk/britney/britney.wiki, last access 04/10/2006.
37. M. Westergaard and K. Bisgaard Lassen. Building and Deploying Visualizations of Coloured Petri Net Models Using BRITNeY Animation and CPN Tools. In K. Jensen, editor, *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (PB-576)*, pages 119–135, Department of Computer Science, University of Aarhus, Oct. 2005.

The BRITNeY Suite: A Platform for Experiments

M. Westergaard

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark,
Email: `mw@daimi.au.dk`

Abstract. This paper describes a platform, the BRITNeY Suite, for experimenting with Coloured Petri nets. The BRITNeY Suite provides access to data-structures and a simulator for Coloured Petri nets via a powerful scripting language and plug-in-mechanism, thereby making it easy to perform customized simulations and visualizations of Coloured Petri net models. Additionally it is possible to make elaborate extensions building on top of well-designed interfaces to Coloured Petri nets created using CPN Tools.

1 Introduction

The Coloured Petri nets formalism [15] (CP-nets or CPNs) has been successfully used to model a lot of different systems, including network protocols [9, 10, 17, 18, 21], work-flows [2, 16], etc. The formalism is supported by CPN Tools [6, 22], a tool for editing and simulating CP-nets. As the CP-nets formalism has a lot of power, both computationally (in general, the question of whether a transition is enabled is not computable) and from a modeling standing point, a lot of research is made to improve analysis of the formalism (in particular experiments with different state space reduction techniques) and to use the formalism in different settings.

While CPN Tools is a great editor for CP-nets, it is not, in general, well-suited for experiments with the formalism, e.g. to test new language constructs or to integrate CPN models easily with other formalism or programs, for several reasons. Firstly, CPN Tools is closed source, which makes it difficult for people other than the developers to make modifications and extensions to the tool. Secondly, CPN Tools is written in the Beta programming language [20], which is not widely known, which makes it difficult to get started for an average programmer even if he had access to the CPN Tools source code as well as impossible to obtain off-the-shelf components for common tasks, e.g. chart-drawing, and programmers therefore have to build such components themselves. Finally, CPN Tools has no plug-in mechanism, which, in combination with the previous problems, makes it virtually impossible to experiment with CP-nets created using CPN Tools without great effort.

The BRITNeY Suite [23, 25] alleviates all of these problems by providing a Java-based, open source (GNU Public License, GPL), pluggable platform for experimenting with CP-nets created in CPN Tools. Additionally the BRITNeY Suite comes equipped with a scripting language, which makes it easy to perform simple experiments. Earlier versions of the BRITNeY Suite have already been used in various projects, some of these are described in [16, 18, 19].

The contribution of this paper is not so much theoretical as it is a description of a platform. The reader is expected to have knowledge of CP-nets as implemented by CPN Tools as well as object-oriented programming, preferably in Java. The reader is assumed to be familiar with CPN Tools and the BRITNeY Suite (or have access to the BRITNeY Suite and try out the examples while reading).

In the next section we shall take a look at the architecture of the BRITNeY Suite and how it has evolved from a simple animation tool to a full bodied platform for experimenting with CP-nets. In Sect. 3 we shall take a look at how to create a simple script for running a customized simulation of a model downloaded from the Internet. In Sect. 4 we will take a look at how to extend the BRITNeY Suite with completely new functionality using the plug-in-mechanism, and, finally, in Sect. 5 we conclude.

2 A Brief History of the Architecture of the BRITNeY Suite

In order to fully understand the architecture of the BRITNeY Suite and how it can be used to perform experiments with CP-nets, it is beneficial to take a brief look at how the BRITNeY Suite evolved from a simple animation library to a full-blown platform for interacting with CP-nets. The walk through is historical, but also provides a top-down look at the current structure of the BRITNeY Suite, from the most abstract level to a more concrete level.

In Fig. 1 we see an abstract view of the internals of CPN Tools. We see that CPN Tools is in fact composed of two components, the CPN Tools GUI, to the left, and the CPN simulator, to the right. The GUI is responsible for editing CP-nets and sending them to the simulator, where they are checked for correctness and code is generated to simulate the model. The GUI communicates with the simulator using a proprietary binary protocol, which may change with new versions of CPN Tools. The uni-directional arrow indicates that the CPN Tools GUI must initiate any action (but data can be transferred in the other direction as responses to requests, e.g. the GUI may check a CP-net, and the simulator sends back data about how the check went). This architecture may seem a bit strange at first, but it has several advantages: Firstly, it is possible to write the editor in a language suitable for writing graphical user interfaces, and we can write the simulator in a language well-suited for implementing compilers. Secondly, we are able to run the editor on a single desktop PC while running the simulator on a server with a lot of memory and a powerful processor.

The BRITNeY Suite started as a simple animation library for use with CPN Tools. The library supported drawing and moving simple geometric figures on the screen. Originally the library was a simple application which exposed functions,

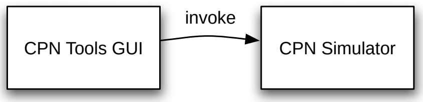


Fig. 1. The architecture of CPN Tools.

which could be called using a simple Remote Procedure Call (RPC) mechanism [5, Chap 5.3] based on the COMMS/CPN library [7]. A bit of code had to be loaded into the simulator in order for the simulator to be able to execute the remote procedures of The BRITNeY Suite. This architecture can be seen in Fig. 2, where we have the CPN Tools GUI and simulator, as before, on the left. These two comprise CPN Tools. Loaded into the simulator, we find some RPC stubs, which are able to call the procedures in the animation tool on the right. Here the CPN Tools GUI must initiate actions in the simulator (simulate a CP-net), and the simulator can then initiate actions in the BRITNeY Suite (animate objects). The animation library was written as a separate application in order to be able to use the powerful standard library of Java.

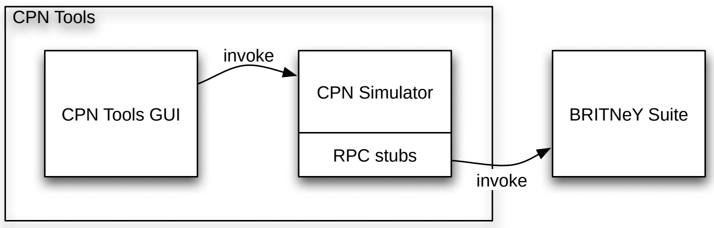


Fig. 2. The architecture of the first version of the BRITNeY Suite.

The next thing to happen was the introduction of so-called *animation objects*, objects representing an animation. By adding a new animation object, it is possible to extend the functionality of the animation tool, to add, e.g., functionality for drawing Gantt-charts. In order for CPN Tools to be able to communicate with the newly added animation objects, the user needed to generate stubs manually and load them into the simulator. This architecture can be seen in Fig. 3. The only change from Fig. 2 is that the tool is no longer locked to a single kind of animation, but contains interfaces, which can be used to extend the functionality.

After adding animation objects, the next step was to add *animation plug-ins* which are, like animation objects, responsible for creating an animation. The main difference between animation objects and animation plug-ins is that animation plug-ins are detached from the tool, can be developed outside the development tree of the BRITNeY Suite itself, and can be automatically downloaded from the Internet and loaded on runtime. Now the exposed features of the BRITNeY Suite are truly dynamic, and the old method of manually generating

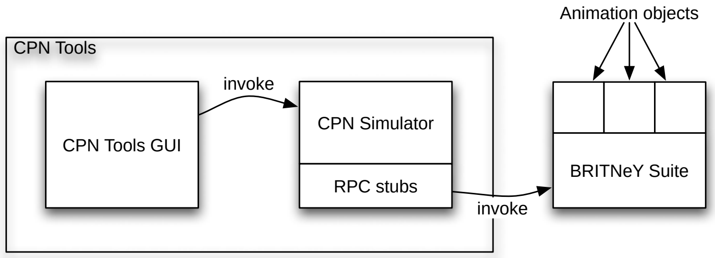


Fig. 3. The architecture of the BRITNeY Suite with animation objects.

and loading RPC stubs becomes infeasible in practice. Therefore an automatic stub-generator was introduced. The responsibility of this component is to generate and load the stubs into the CPN simulator. In order to facilitate this, the BRITNeY Suite needs to communicate with the simulator (and not just the other way around). Unfortunately, only one process can communicate with the simulator at any time. Therefore the BRITNeY Suite acts as a proxy for the communication from CPN Tools. The architecture can be seen in Fig. 4. Here the CPN Tools GUI is on the left, the BRITNeY Suite in the middle and the simulator on the right. The CPN Tools GUI no longer communicates directly with the simulator, but communicates with the *Simulator proxy* component of the BRITNeY Suite. Above the BRITNeY Suite main component, completely detached, lie the animation plug-ins. The *Stub generator* component uses Java reflection [12] to automatically generate stub code for each registered animation plug-in. This code is then loaded into the simulator whenever the CPN Tools requests the creation of a new simulator.

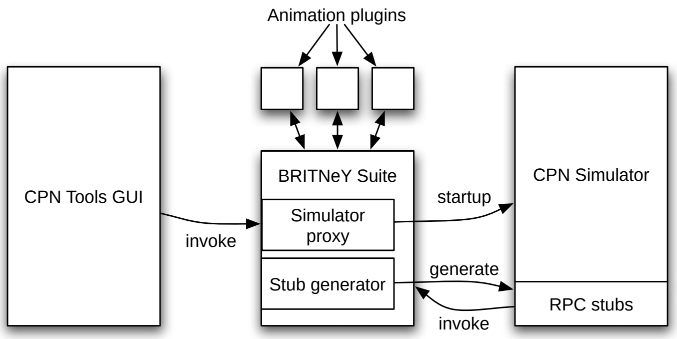


Fig. 4. The architecture of the BRITNeY Suite with animation plug-ins.

Having a very flexible plug-in architecture in place, a lot of refactoring took place. Firstly, the program was split up into a number of different plug-ins to make it easier to maintain and to provide versions that only support the required functionality. Secondly, the home-made RPC protocol was thrown away in favor of a standardized open RPC protocol, XML-RPC [26]. These two changes made the tool much more open, as extensions can be written as simple plug-ins, and other tools can interface with the animation plug-ins using a simple standardized protocol. In Fig. 5 this can be seen. Here the BRITNeY Suite, in the middle, is split up into a *Simulator* plug-in, a *Simulator proxy* plug-in and an *Animation* plug-in with the Stub generator[1]. The plug-ins may depend on and communicate with other plug-ins, as indicated by the unlabeled bidirectional arcs (a plug-in depends on plug-ins below it connected by arcs). Animation plug-ins are normal plug-ins communicating with the plug-in Animation. They have been renamed to *Extension plug-ins*, as they can be used for more than just animation, e.g. a plug-in has been created to facilitate data-storage.

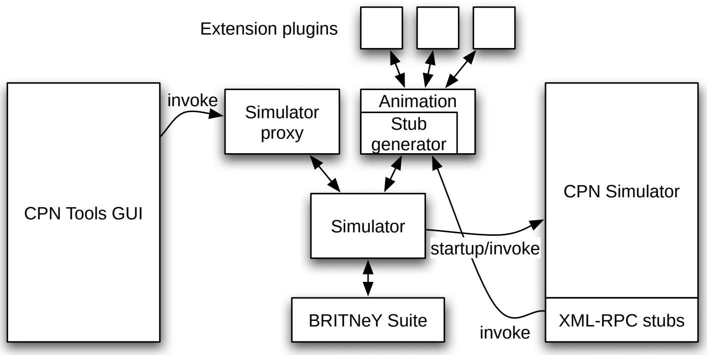


Fig. 5. The architecture of the BRITNeY Suite with general plug-ins.

The next thing to happen is that the BRITNeY Suite learns more about CP-nets. Originally, the tool had no need to know anything about CP-nets, as it was just a dumb RPC-server. In order to define extensions at a higher level, a simple CP-net model was introduced into the BRITNeY Suite. The model can be created by either snooping on the channel of the Simulator proxy (in order to syntax check and simulate a CPN model, all details of the model are transferred from CPN Tools, via the BRITNeY Suite, to the CPN Simulator) or by loading a net description from a PNML [11] file. Nets created with CPN Tools can also be loaded by translating them into PNML first (this is done automatically and transparently to the user). The net model can be used for various things. For example the model provides a high-level abstraction of Fusion places and transitions of the model, allowing extension plug-in writers to use a simple interface to the state and actions of the model. This can be used to create simple views of the model using Java.

[1] This is a simplification of the plug-in architecture of the BRITNeY Suite; in fact the BRITNeY Suite consists of nearly 20 plug-ins.

To sum up, the BRITNeY Suite provides a platform for experimenting with CP-nets created using CPN Tools for the following reasons:

– it is distributed under an open source license (GPL),
– it is written in a well-known language (Java), enabling the reuse of off-the-shelf components,
– it has a pluggable architecture,
– it has abstractions of the CPN simulator on several levels (from direct access to the simulator's interface up to an executable CPN object model).

3 Scripting Engine

Even though the BRITNeY Suite comes equipped with a powerful plug-in system, the full power of this is often not needed. In order to allow simple tasks to be performed, the BRITNeY Suite employs a simple scripting engine, which, while simple, is powerful, as most of the internals of the BRITNeY Suite are exposed through this engine.

The scripting engine used is BeanShell [1], which allows programmers to use a Java-like syntax to create macros (in fact Java syntax is allowed, but so is a more relaxed version). A complete tutorial on using BeanShell is out of scope of this paper, but the provided example should be a good starting point for writing your own macros.

The example assumes we have created a CPN model with a nice animation using the approach described in [24], which shows how to create CPN models, using animations created for the BRITNeY Suite, which can be executed without the help of CPN Tools. Basically, you have to do all initialization of the animation using an init-transition, and use the Save simulator as-tool from the Simulator-menu, to export a simulator-image, which we have uploaded to our web-server, `http://www.example.org/`, as `simulator.x86-win32`. We now want is to write a script which allows us to run a simulation a number of times. To make the simulation more personal to the user, we will ask him for his name and use this during the simulation. The created such that we need to execute 100 steps of the simulation with 100 ms delay between steps. We want the user to be able to configure the number of runs using the built in option pane in the BRITNeY Suite. To make execution easy, we will add a new menu item to start the simulation. In order to do this, we will use some exported objects and some utility classes provided by the BRITNeY Suite. Exported objects allow the programmer to manipulate the internal data-structures of the BRITNeY Suite and utility classes allow the programmer to add entries to the menu for evaluating scripts.

The rest of this section is structured as follows: First we will take a brief look at some of the exported objects and their use (thereby adding the functionality described above) and then we will look at some of the utility classes and how to use them (thereby adding a menu item for our new feature).

3.1 Exported Objects

The BRITNeY Suite, at the time of writing, exports 6 objects for use in the scripting language. The objects are Options, MessageDisplay, ToolModel, SimulatorService, Tool, and IndexModel. In addition to this, all commands of the tool (i.e. actions that can be selected from the menus) are exported as well.

In this example, we shall use the first four exported objects (the two other are not used that often) and one of the exported commands. The Options exported object can be used to add new pages to the options facility of the tool, MessageDisplay can be used to display messages and progress-bars in the status-line. ToolModel can be used to get access to low-level parts of the tool, in particular a reference to the main window, which can be used to create modal dialog windows. The SimulatorService can be used to instantiate or load CP-net simulators. The IndexModel exported object can be used to remove items from the menu of the applications and the Tool object gives access to the internal objects of the BRITNeY Suite, but most of these are already exported into the BeanShell engine, and Tool is therefore rarely used.

The code in Listings 1 and 2 implements the new feature. Listing 2 contains the actual implementation and Listing 1 contains standard Java code defining a new class, Runs, with a single field (count) and two accessor methods, getCount and setCount, coded using the guide lines in [13, Chap. 7: Properties] (basically a property, foo must have getter and setter methods called `getFoo` and `setFoo`).

Listing 1 BeanShell code to implement the Runs class.

```
public class Runs {
  int count = 3;

  public int getCount() {
    return count;
  }

  public void setCount(int count) {
    this.count = count;
  }
}
```

Line 1 in Listing 2 creates an instance of our newly created class. Note that in the BeanShell language, we do not need to specify the type of the variable. In line 2 we use a method of the Options object to add a new managed object for options. The first parameter is a descriptive name, the second is a reference to the object we want to be managed. The last parameter is a so-called stop-class, which can be used to restrict what variables are shown in the options pane. Normally Object.class is fine[2].

[2] Check the documentation for `java.beans.Introspector#getBeanInfo(java.lang.Class,java.lang.Class)` for more information.

Listing 2 BeanShell code to implement multiple runs of an animation.

```
1  runs = new Runs();
2  Options.addManagedObject("Runs", runs, Object.class);
3
4  void runSimulation() {
5    MessageDisplay.showProgress("Loading", 0, 2);
6    mainWindow = ToolModel.getMainWindow();
7    MessageDisplay.showProgress("Loading", 1, 2);
8    name = javax.swing.JOptionPane.showInputDialog(mainWindow,
9      "What is your name");
10   MessageDisplay.showProgress("Loading", 2, 2);
11   MessageDisplay.showMessage("Hello " + name, 0);
12
13   s = SimulatorService.getNewSimulator(
14     new java.net.URL("http://www.example.org/simulator.x86-win32"));
15   hs = s.getHighLevelSimulator();
16
17   oldsteps = Play.steps;
18   olddelay = Play.delay;
19   Play.steps = 100;
20   Play.delay = 100;
21
22   hs.evaluate("name := \"" + name + "\"");
23
24   for (int i = 0; i < runs.getCount(); i++) {
25     hs.initialState();
26
27     event = new java.awt.event.ActionEvent(hs, 0, "");
28     Play.execute(event);
29   }
30
31   Play.steps = oldsteps;
32   Play.delay = olddelay;
33 }
```

The rest of the code is the actual implementation of the new functionality, and consists of a single method. We notice that in BeanShell methods can exist outside of a class. Lines 5, 7, and 10 provide a short progress-bar using MessageDisplay. The showProgress method takes 3 parameters: a descriptive text, how much has been completed, and how much do we need to complete. Lines 6, 8, and 9 take care of showing a dialog box querying the user for his name. In line 6 we obtain a reference to the main window of the application, which is used to display a modal dialog to the user using JOptionPane in lines 8 and 9. In line 11 we show a friendly greeting to the user in the status bar using the showMessage method of MessageDisplay. The method takes two arguments, the text to display, and a message type. The message type is no longer used and should just be set to 0.

In lines 13–15 we obtain a reference to the simulator that we have uploaded to our web-server using SimulatorService. First we call getNewSimulator, which takes as parameter a URL pointing to the location of the simulator, and then we use the obtained simulator to obtain a HighLevelSimulator, which encapsulates a lot of functionality.

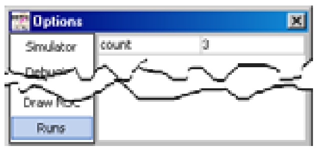
Lines 17–20 will be explained later. In line 22 we store the name of the user, which we have saved in the name variable in the simulator. We simply evaluate ML-code corresponding to assigning the value of the BeanShell variable to a ML reference. This code assumes that we have added a declaration `val name = ref "dummy";` in our CPN model.

In line 24 we start the actual loop. This is a standard Java for loop, which makes use of the getCount accessor method of the runs variable we declared in Listing 1. If the user changes the setting in the options pane, this will automatically be reflected, and the loop will run more than the default 3 iterations.

In line 25 we reset the simulator to the initial state. This is performed by calling a method on the HighLevelSimulator we obtained earlier.

Lines 27 and 28 take care of executing the simulation using the Play command. First, in line 27, we create an event to send to the command. The ActionEvent class takes as first parameter the source of the event. The Play command works on HighLevelSimulators, so we give this as parameter. The two last parameters are not used and should just be set as in the example. Finally, in line 28, we execute the Play command. In order to run the simulation with the desired settings (100 steps, at least 100 ms delay between transitions), we prepare our use of the Play command in lines 17–20. We need to set the options of this command, but as these are persistent, we first save the old values (which we restore later in lines 31–32), and then set the new values to execute 100 steps with a minimum delay between transitions of 100 ms.

After evaluating the code from Listings 1 and 2 in the Script Console, we can change the value of the runs parameter in the Options dialog (see Fig. 6) and start the simulation by evaluating `runSimulation()` in the Script Console. This procedure may not be very user-friendly, so let us immediately turn to creating menu items for our new feature using some utility classes.



count 3

Fig. 6. The Options dialog for our newly created option, Runs. The middle of the dialog has been cut out.

3.2 Important Utility Classes

Using the BeanShell scripting engine, users have unlimited access to all classes available in Java and the BRITNeY Suite. Here we shall only treat two of them (actually we have already used a couple of other classes provided by both the BRITNeY suite and Java earlier in this section, namely the Simulator, HighLevelSimulator, JOptionPane, ActionEvent, and URL classes). The two classes we shall need are the ScriptingCommand and ScriptingTools classes. These classes allow us, from the BeanShell, to create new commands, which can be added as items in new menus.

In the BRITNeY Suite menus are represented using tool boxes. One implementation of a tool box is provided by the ScriptingTools class. Tool boxes contains commands, which can be executed. The ScriptingCommand provides one such implementation, which is able to execute BeanShell code. We shall use this to add a “Demo” menu to our application. This menu shall have one entry, namely “Start simulation”, which will call the method we defined earlier in this section.

In Listing 3 we see how to create a ScriptingCommand in lines 1–4. The first parameter is a short descriptive name, which is shown in the menu. The second parameter is a longer description, which is shown as tool-tip when the command is shown in the tool-bar. The third and final parameter is the code to execute. Any BeanShell code can appear here, but quotes (”) must be escaped (as \”), so it will often be easier to write a method for the feature and then just call the method from the ScriptingCommand, just like we have done in this example.

Listing 3 BeanShell code to add a new menu with an item to execute the method defined in Listing 2.

```
1 sc = new dk.klafbang.tincpn.scripting.ScriptingCommand(
2   "Start simulation",
3   "Runs the number of simulations specified in the options",
4   "runSimulation()");
5 tb = new dk.klafbang.tincpn.scripting.ScriptingTools("Demo");
6 tb.add(sc);
```

In Fig.7 the resulting menu and menu-item is shown. When the item is selected, the code from Listing 2 is called, starting by asking the user for his name and then running (in this case) 3 simulations of the model.

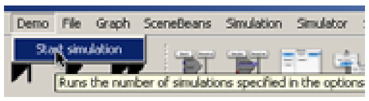


Fig. 7. A menu generated using the script code in Listing 3.

3.3 Other Possibilities

In this section we have seen a simple example of how to run a customized simulation of a CPN model with an associated animation. The example uses a lot of the features in the BRITNeY Suite. There is, however, an important feature, we have not touched at all, namely the ability to inspect and control the simulation of a CPN model. This can be used for a lot of things, such as altering the state of the model or controlling how/in what order transitions are executed.

As a simple example, let us see how we can implement prioritized transitions with the BRITNeY Suite. This can of course be done by altering the model, but for real models, this is not very easy and unnecessarily clutters the model. We will look the net in Fig. 8. Here we want to transport the single token of type UNIT from the Start place to the End place, via the P1 and P2 places. Transport can be done either by the transitions A# or B# where $\# \in \{1, 2, 3\}$. We furthermore put a token on the A (resp. B) place whenever a A# (resp. B#) transition fires. We want A# transitions to have priority over B# transitions (i.e. when we are done, we want two tokens on A and one token on B).

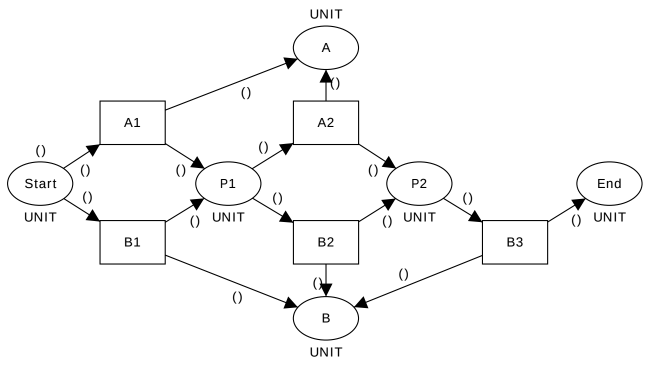


Fig. 8. A simple CP-net. We want all A transitions to have priority over B transitions.

When we simulate the net in Fig. 8 in CPN Tools, we will often get more than one token on B, indicating that A# transitions do not have priority over B# transitions. We can fix this problem with a bit of scripting in the BRITNeY Suite. Basically, we have to write our own scheduler, which prefers A# transitions over B# transitions. Although this sounds like a complicated task, it is quite simple as can be seen by the code in Listing 4. The code first loads the net (ll. 1–2), extracts an abstract description of the net (l. 4). In addition to the abstract description of the net (the netmodel), the triple returned in line 1 also contains an abstract description of how the net is displayed (the second component, a viewmodel, which which can be used to change how the net is presented to the user—we will not use this feature in this example to keep things simple), and a CPN checker process (the third component, a netchecker), from which we can obtain a CPN simulator object.

After loading the model, two lists are constructed, one for high-priority transitions, and one for low-priority transitions (ll. 6–19). The lists are constructed by traversing all instances (l. 10) of all transitions (l. 9) on all pages of the net (l. 8), and if the transition name starts with an A (l. 11) add it to the list of high-priority transitions (l. 12) and otherwise add it to the list of low-priority transitions (l. 14). After that we obtain the simulator for the net (ll. 19–20), reset the simulator (l. 22), initializes the step counter and the maximum number of steps to take (l. 23) and starts the simulation (ll. 24–29). The simulation updates the progress bar (l. 25), delays (l. 26), and calculates whether it was possible to execute any transitions (l. 27). This calculation is performed by executing an A# transition if one is enabled and otherwise a B# transition (the right-hand side of the or expression is not evaluated if the left-hand side evaluates to true). Then the step counter is incremented (l. 28). When the simulation is done, the progress-bar is reset (l. 30). Even though the script is simple, it will do the task, and it even keeps track of progress and reacts to changes to the global options of the BRITNeY Suite.

We have now seen how we can use inspection and control over the execution of CPN models to support priority of transitions by gathering all high-priority transitions in one list and all other transitions in another, and then execute transitions from the first list before transitions from the other list. In a similar way, we can also simulate inhibitor arcs [3] by creating arcs with the inscription empty where we intend inhibitor arcs. This arc will never prevent the transition from being enabled, so if we interpret the arc as an inhibitor arc, it will be a restriction. When we search for a transition to execute, we will simply omit all transitions with "inhibitor"-arcs if any place in the other end is marked with tokens. The final application of this, we will look at, is transition fusion [4] (or synchronous channels) with no data-transfer. Here a transition, e.g., A!, is enabled iff another transition, A?, is enabled. If we ignore all !-transitions where no corresponding ?-transition is enabled (and vice versa), and always execute an ?-transition immediately after a !-transition, we have implemented transition fusion with no data-transfer. We can implement transition fusion with data-transfer by also inspecting the enabled bindings, but this is more complicated.

We can use the ability to alter the state to communicate with external processes in an asynchronous way, for example we can implement fusion places between two independent nets by running the simulations in parallel and synchronizing the tokens on all shared places between transitions. We can also combine the place inspection features with the transition selection features to implement bounded places (by inspecting the effect of executing a binding element on all bounded places and disallowing the execution if it violates the bound) and FIFO (first-in-first-out) places (by keeping track of when tokens arrive, and only allow the execution of a binding element if it consumes tokens in the correct order).

These examples are just appetizers of what can be accomplished relatively easily, using few lines of scripting code. The fact that well-known concepts can be realized with such ease, even though the interface was not designed for that purpose initially, makes it believable that completely new concepts can be real-

Listing 4 Code implementing a scheduler, which prefers A# transitions over B# transitions.

```
triple = LoadNet.loadNet(new java.io.File("c:/contention.cpn"));
LoadNet.registerIndexNode(triple);

netmodel = triple.getFirst();

As = new ArrayList();
Bs = new ArrayList();
for (page : netmodel.getPages()) {
  for (transition : page.getTransitions()) {
    for (instance : transition.instances()) {
      if (transition.getName().startsWith("A"))
        As.add(instance);
      else
        Bs.add(instance);
    }
  }
}

netchecker = triple.getThird();
simulator = netchecker.getSimulator();

simulator.initialState();
i = 0; max = Play.steps;
do {
  MessageDisplay.showProgress("Executing", i, max);
  Thread.sleep(Play.delay);
  executed = simulator.fireAny(As) || simulator.fireAny(Bs);
  i++;
} while (executed && i < max);
MessageDisplay.showProgress("Executing", max, max);
```

ized relatively easily as well. As already mentioned, these features can of course be realized directly by changing the model, but the idea behind introducing more elaborate constructs is to simplify the model. Using the BRITNeY Suite, we are able to experiment with new constructs before they are integrated into CPN Tools.

4 Extension Plug-ins

In this section, we will look at an example of how to create a simple extension to the BRITNeY Suite. This consists of three steps: First we write one or more Java classes implementing the new feature, then we write a plug-in-descriptor describing how the plug-in should be loaded, and finally we package our plug-in and deploy it.

Throughout this section we will develop a simple plug-in that allows us to play sounds. We will add a new menu, Sound, to the GUI of the BRITNeY Suite add a menu item which will play a sound through the computer's speakers, and register a new extension plug-in, which can be used to allow CPN models to play sound during the execution. This example also illustrates the reason for renaming animation plug-ins to extension plug-ins, as a sound player is not a visualization, but just a new feature, we can add to the toolbox available from CPN Tools.

4.1 Writing a Java Implementation

Writing Java classes is out of scope of this paper, but it may be instructional to know of a few good hooks into the BRITNeY Suite. For more detailed information, refer to the on-line source code documentation, which can be found at `http://www.daimi.au.dk/∼mw/local/tincpn/doc/`. In the following we will refer to packages and classes within this documentation.

We will start by creating a class for playing sounds. This is just regular Java code, and has nothing to do with the BRITNeY Suite. The code relies on the Java Applet code to play sounds (even though a better way has found its way into Java in later versions, but this is much more verbose). The class is able to load sound clips and store them for later playback (the loadSound method) as well as playing back previously loaded sound clips (the playSound method). The code can be seen in Listing 5.

The most important hook is probably the ToolBox plug-in, which contains classes that make it possible to add new menus, menu items and corresponding actions. The plug-in is implemented in the dk.klafbang.tincpn.tools package, and contains among other items the classes Command and AbstractToolBox, which can be used as a basis for adding new commands that can be executed when an item is selected from a menu and for adding new menus respectably.

The implementation of the command for playing sounds can be seen in Listing 6. We start by importing the classes we will need (ll. 1–4). We then create a new class inheriting from the Command class (l. 6). The constructor (ll. 9–12) calls the constructor with a name to show in the menu and a tool tip. The super class constructor can optionally take a third parameter, the location of an icon for the command. We also create an instance of the SoundPlayer class we created above (ll. 7 and 11). In the execute method (ll. 14–20), the actual action takes place. We start by calling the execute method of the super class (l. 15). We then use the built-in class, JFileChooser, to show a file dialog, which allows the user to select a file to play. If the user selects a file and presses Ok, the sound is played using the SoundPlayer instance (l. 18–23) by first converting the the file selected by the user into a string representation of the URL pointing to the file (l. 19), loading the sound clip (l. 20), and finally playing back the clip (l. 21). As all of the methods may fail, we have encapsulated the sound playing code in an exception handler (ll. 18,22–23).

We would also like to be able to show the newly created command in a new menu. We do this by sub-classing the AbstractToolBox as seen in Listing 7. The

Listing 5 A simple class for playing back sound in Java.

```java
import java.applet.*;
import java.net.URL;
import java.util.*;

public class SoundPlayer {
  Map<Integer, AudioClip> clips;
  int counter = 0;

  public SoundPlayer(final String name) {
    clips = new HashMap<Integer, AudioClip>();
  }

  public synchronized int loadSound(final String url) throws Exception {
    final URL location = new URL(url);
    final AudioClip audioClip = Applet.newAudioClip(location);
    clips.put(counter, audioClip);
    return counter++;
  }

  public void playSound(final int key) throws Exception {
    clips.get(key).play();
  }
}
```

code is very simple. The constructor calls the super class constructor with the name of the new menu (l. 5) and adds a new SoundPlayerCommand (l. 6). We can add as many commands as we want.

Other hooks that may be interesting include the net model (in the package dk.klafbang.tincpn.nets.model, in the plug-in NetModel) and the simulator interface (the classes Simulator, HighLevelSimulator, and SimulatorService in the package dk.klafbang.tincpn.simulator in the plug-in Simulator). We have already seen how to use these from the scripting language in the previous section, but we can of course also use them from plug-ins.

4.2 Writing a Plug-in-descriptor

In order to be able to use our newly created plug-in, we need to tell the BRITNeY Suite how to use it. To do this, we will need to create a plug-in-descriptor. The descriptor describes where the plug-in code can be found, what plug-ins the plug-in relies on, and, most importantly, how the plug-in should be integrated.

Plug-ins in the BRITNeY Suite are created using the Java Plug-in Framework [14]. This means a plug-in-descriptor has to be written using an XML-file. The complete specification of the plug-in-descriptor format can be found at `jpf.sourceforge.net/dtd.html`, but it should be quite easy to infer the general format from this example.

Listing 6 A command for playing sounds.

```
import java.awt.event.ActionEvent;
import javax.swing.JFileChooser;
import static javax.swing.JFileChooser.APPROVE_OPTION;
import dk.klafbang.tincpn.tools.Command;

public class SoundPlayerCommand extends Command {
  SoundPlayer player;

  public SoundPlayerCommand() {
    super("Play sound", "Plays a sound");
    player = new SoundPlayer();
  }

  public void execute(final ActionEvent event) {
    super.execute(event);
    JFileChooser chooser = new JFileChooser();
    if (chooser.showOpenDialog(null) == APPROVE_OPTION) {
      try {
        final String url = chooser.getSelectedFile().toURL().toString();
        final int key = player.loadSound(url);
        player.playSound(key);
      } catch (final Exception e) {
      }
    }
  }
}
```

Listing 7 A toolbox for playing sounds.

```
import dk.klafbang.tincpn.tools.AbstractToolBox;

public class SoundTools extends AbstractToolBox {
  public SoundTools() {
    super("Sound");
    add(new SoundPlayerCommand());
  }
}
```

The entire plug-in-descriptor for our sound plug-in can be seen in Listing 8. The first four lines states that this is a plug-in-descriptor.

Line 4 identifies this plug-in to the surroundings. The id must be unique and the version should reflect the version of the plug-in.

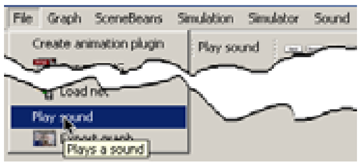
Next, in lines 5–8 follow the dependencies section. We state that we depend on the ToolBox plug-in (to create Commands and AbstractToolBoxes) and the AnimationTools plug-in (to register our extension plug-in).

This is followed by the runtime section (ll. 10–17), which describes where to find the code. In this case the runtime consists of two locations, a code and a resources location. Each library declaration consists of an id (a unique descriptor of the library) a path where the code can be found. In this case this is just ".", i.e. the code can be found in the same location as the plug-in descriptor itself. The path can also point to another directory, a jar-file or even a web-location. You can have more than one library declaration for each plug-in. This allows you to use third party jar-files in your plug-in.

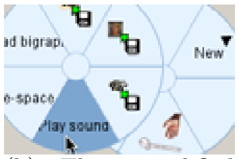
Listing 8 The plug-in-descriptor for the Sound plug-in.

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.3"
  "http://jpf.sourceforge.net/plugin_0_3.dtd">
<plugin id="Sound" version="1.0.0">
  <requires>
    <import plugin-id="ToolBox"/>
    <import plugin-id="AnimationTools"/>
  </requires>

  <runtime>
    <library id="Sound.code" path="." type="code">
      <export prefix="*"/>
    </library>
    <library id="Sound.res" path="." type="resources">
      <export prefix="*"/>
    </library>
  </runtime>

  <extension plugin-id="ToolBox" point-id="ToolBox"
      id="SoundTools">
    <parameter id="class" value="SoundTools"/>
  </extension>
  <extension plugin-id="ToolBox" point-id="NewCommand"
      id="SoundPlayerNewCommand">
    <parameter id="class" value="SoundPlayerCommand"/>
  </extension>
  <extension plugin-id="ToolBox" point-id="LoadCommand"
      id="SoundPlayerLoadCommand">
    <parameter id="class" value="SoundPlayerCommand"/>
  </extension>
  <extension plugin-id="ToolBox" point-id="AdditionalCommand"
      id="SoundPlayerCommand">
    <parameter id="toolbox" value="Files"/>
    <parameter id="class" value="SoundPlayerCommand"/>
  </extension>
  <extension plugin-id="AnimationTools" point-id="Animation"
      id="SoundTools">
    <parameter id="class" value="SoundPlayer"/>
  </extension>
</plugin>
```

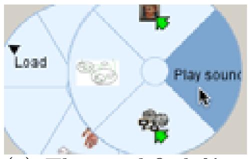
After this follows the section where your plug-in is tied to the BRITNeY Suite. This is done using so-called *extension points*. The BRITNeY Suite comes pre-configured with a number of extension points. This plug-in extends all five currently available extension points to show how it is done. Each extension point is specified using a plugin-id, the id of the plug-in specifying the extension point, a point-id, the id of the point to extend, and an unique id for the extension. The first extension (ll. 19–22) adds a new tool box (menu) to the tool. It takes a single parameter, namely the class implementing the tool box. In this case this is the SoundTools class. The next two extensions (ll. 23–30) are related and add commands to the marking menu on the desktop. The first adds a "new"-command, i.e. a command for creating new things (e.g. creating a new net or starting a new simulator etc.). The second extension adds a "load"-command, i.e. a command for loading things (e.g. loading a net or loading a simulator). The fourth extension (ll. 31–35) is for smaller plug-ins, which do not require their own top-level menu, but which wants to add a command to an existing menu, here the Files menu. The final extension (ll. 36–39) adds a new extension plug-in. We just add our utility-class as the extension plug-in, and nothing extra needs to be done. Now, if we start the BRITNeY Suite and CPN Tools, we will see an extra menu for playing sounds in the BRITNeY Suite (Fig. 9(a) at the top-right corner), we will see an extra item in the Files menu (Fig. 9(a) to the left) and extra entries in the Load (Fig. 9(b)) and New (Fig. 9(c)) marking menus. Furthermore we can use our extension plug-in in CPN Tools, e.g. as in Listing 9.



(a) The modified menu bar. The middle of the figure has been cut out.



(b) The modified Load marking menu.



(c) The modified New marking menu.

Fig. 9. A menu and two marking menus generated by the Sound plug-in.

This example has shown how to add completely new functionality to the BRITNeY Suite and tie it into the tool in various places. Often we will not tie it into the tool in as many places as we have done here, but e.g. only register a class as an animation extension (thereby creating an extension plug-in), or only

Listing 9 Code to instantiate and use our newly created extension plug-in

```
structure sound = SoundPlayer(val name = "Ignored");
val _ = sound.playSound "c:/sound.au"
```

add an entry to the menu, e.g. to create new editing facilities. It is also possible to add new extension points to the tool, but this is out of the scope of this paper. As already mentioned, most of the BRITNeY Suite is actually implemented as plug-ins in a very small core application; for example the visualization facility, the previous core functionality of the BRITNeY Suite, is just a plug-in, and can be removed if desired.

5 Conclusion

In this paper we have seen how the architecture of the BRITNeY Suite supports experimentation with CP-nets for people without access to the source of CPN Tools. The BRITNeY Suite is distributed under an *open source license*, it is *written in a well-known programming language*, namely Java. The BRITNeY Suite has a *pluggable architecture*, which provides *abstractions of the CPN simulator* as well as a powerful *scripting engine*.

We have seen how to create simple extensions using the scripting engine and how to provide completely new functionality by creating a new extension plug-in.

Future work includes making the CPN abstraction more Java-like, and e.g. provide an *Observable* [8][pp. 293-304] abstraction of places, so it is possible to be notified whenever the marking on a place or a fusion group changes. Other than that, current and future research include experimentation with variations of CP-nets using the abstractions provided by the BRITNeY Suite.

References

1. The BeanShell Scripting Language. Java Specification Requests, JSR: 274, `http://jcp.org/en/jsr/detail?id=274`.
2. C. Bossen and J.B. Jørgensen. Context-descriptive prototypes and their application to medicine administration. In *DIS '04: Proc. of the 2004 conference on Designing interactive systems*, pages 297–306, Boston, MA, USA, 2004. ACM Press.
3. G. Chiola, S. Donatelli, and G. Franceschinis. Priorities, Inhibitor Arcs and Concurrency in P/T nets. In *Proc. of ICATPN 1991*, pages 182–205, 1991.
4. S. Christensen and N.D. Hansen. Coloured Petri Nets Extended with Channels for Synchronous Communication. In *Proc. of ICATPN 1994*, volume 815 of *LNCS*, pages 159–178. Springer, 1994.
5. G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design*. Addison-Weslay, 3rd edition, 2001.
6. CPN Tools. `www.daimi.au.dk/CPNTools`.
7. G. Gallasch and L.M. Kristensen. A Communication Infrastructure for External Communication with Design/CPN. In *Proc. of 3rd Workshop on Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-554 of *DAIMI*, pages 79–93. Department of Computer Science, University of Aarhus, 2001.

8. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
9. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In *Proc. of ICATPN'02*, volume 2360 of *LNCS*, pages 182–202. Springer-Verlag, 2002.
10. B. Han and J. Billington. Formalising the TCP Symmetrical Connection Management Service. In *Proc. of Design, Analysis, and Simulation of Distributed Syste ms*, pages 178–184. SCS, 2003.
11. Combined WD Circulation, CD Registration and CD Ballot, WD 15909-2, Software and Systems Engineering, High-level Petri Nets – Part 2: Transfer Format. version 0.9, `wwwcs.uni-paderborn.de/cs/kindler/publications/copies/ISO-IEC15909-2-WD0.9.0.Ballot.pdf`, 2005.
12. The Java^{TM} Tutorial: The Reflection API. `java.sun.com/docs/books/tutorial/reflect/`.
13. JavaBeans 1.01 specification. `java.sun.com/products/javabeans/docs/spec.html`.
14. Java Plug-in Framework. `jpf.sourceforge.net/`.
15. K. Jensen. *Coloured Petri Nets—Basic Concepts, Analysis Methods and Practical Use. Volume 1: Basic Concepts.* Springer-Verlag, 1992.
16. J.B. Jørgensen and K.B. Lassen. Aligning Work Processes and the Adviser Portal Bank System. In *REBNITA05*, 2005.
17. L.M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad-hoc Networks. In *Integration of Software Specification Techniques for Application in Engineering*, volume 3147 of *LNCS*, pages 248–269. Springer-Verlag, 2004.
18. L.M. Kristensen, M. Westergaard, and P.C. Nørgaard. Model-based Prototyping of an Interoperability Protocol for Mobile Ad-hoc Networks. In *Proc. of Fifth International Conference on Integrated Formal Methods*, volume 3771 of *LNCS*, pages 266–286. Springer-Verlag, 2005.
19. R.J. Machado, K.B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Execution of UML Models with CPN Tools for Workflow Requirements Validation. In *Proc. of Sixth Workshop and Tutorial om Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-576 of *DAIMI*, pages 231–250, 2005.
20. O.L. Madsen, B. Møller-Pedersen, and K. Nygaard. *Object Oriented Programming in the BETA Programming Language*. Addison Wesley, 1993.
21. C. Ouyang and J. Billington. On Verifying the Internet Open Trading Protocol. In *Proc. of 4th International Conference on Electronic Commerce and Web Technologies*, volume 2738 of *LNCS*, pages 292–302. Springer-Verlag, 2003.
22. A.V. Ratzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In *Proc. of ICATPN 2003*, volume 2679 of *LNCS*, pages 450–462. Springer-Verlag, 2003.
23. M. Westergaard. BRITNeY Suite website. `wiki.daimi.au.dk/britney/`.
24. M. Westergaard and K.B. Lassen. Building and Deploying Visualizations of Coloured Petri Net Models Using BRITNeY animation and CPN Tools. In *Proc. of Sixth Workshop and Tutorial om Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-576 of *DAIMI*, pages 119–136, 2005.
25. M. Westergaard and K.B. Lassen. The BRITNeY Suite Animation Tool. In *Proc. of ICATPN 2006*, volume 4024 of *LNCS*, pages 431–440. Springer-Verlag, 2006.
26. D. Winer. XML-RPC Specification. `www.xmlrpc.org/spec`.

Modelling Defence Logistics Networks [⋆]

Guy Edward Gallasch[1], Nimrod Lilith[1], Jonathan Billington[1], Lin Zhang[2], Axel Bender[2], and Benjamin Francis[2]

[1] Computer Systems Engineering Centre
School of Electrical and Information Engineering
University of South Australia
Mawson Lakes Campus, SA 5095, AUSTRALIA
Email: {`guy.gallasch | nimrod.lilith | jonathan.billington`}`@unisa.edu.au`
[2] Logistics Mission
Land Operations Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh, SA 5111, AUSTRALIA
Email: {`lin.zhang | axel.bender | benjamin.francis`}`@dsto.defence.gov.au`

Abstract. Military logistics concerns the activities required to support operational forces. It encompasses the storage and distribution of materiel, management of personnel and the provision of facilities and services. A desire to improve the efficiency and effectiveness of the Australian Defence Force logistics process has led to the investigation of rigorous military logistics models suitable for analysis and experimentation. Logistics networks can be viewed as distributed discrete event systems, and hence can be formalised with discrete event techniques which support concurrency. This paper presents a Coloured Petri Net (CPN) model of a military logistics system and discusses some of our experience in developing an initial model. Interesting modelling problems encountered, and their solutions and impact on CPN support tools, are discussed.

Keywords: Defence Logistics, Coloured Petri Nets, Modelling experience, CPN support tools.

1 Introduction

Logistics is concerned with the procurement, maintenance, distribution, and replacement of personnel and materiel [23]. More colloquially, logistics is the process of having the right quantity of the right item in the right place at the right time [10]. Modern logistics concepts have their roots in military affairs, but their extension and application to the private sector since World War II has had a profound impact on globalisation and the world economy due to improving practices in integrated and efficient management of commercial supply chains. Business logistics is very much based on efficiency and thus profitability. The military perspective, on the other hand, is broader. Military logistics comprises both peace-time logistics, which is driven by similar considerations as business logistics, as well as support to on-shore and off-shore operations. In operations military logistics is primarily focussed on effectiveness, and needs to be adaptable to sudden changes in demand, resilient to disruptions, and often anticipatory of peak consumption. Thus military logistics systems are designed around the principles of effectiveness, robustness, flexibility and agility while striving to maximise efficiency where possible.

Military logistics is often a sizable business activity as well as an integral part of a nation's overall military capability. For instance, the Logistics operations of the US Department of Defense (DoD) is estimated to cost \$90B(USD) per year, involving over a million people [18]. There is therefore a continuing need to improve efficiency and effectiveness. Similar to its major allies, Australian Defence Force (ADF) Logistics is undergoing a process of transformation. Major drivers of this transformation include: the increasingly uncertain and complex future operational environment, demanding more robust and agile logistics support; recent technological advances (especially in the areas of sensor, communications and information technologies) promising improved situational awareness, more efficient planning and flexible execution of logistics activities; and ongoing pressures on the availability

[⋆] Authors from the University of South Australia were supported by DSTO Contract No. 4500498737

of resources, especially human resources due to the ageing population and the difficulty of retaining skilled personnel.

1.1 Problem Statement and Motivation

The ADF aspires to a networked and distribution based logistics system in the year 2025 with the characteristics of agility, robustness, precision, interoperability and deployability. Contemporary business and military logistics initiatives will be investigated and, where appropriate, applied to transform Defence Logistics. An important approach to the analysis of advanced concepts and military capability development is experimentation [17]. A powerful paradigm for experimentation is *Model-Exercise-Model* (M-E-M), which involves three steps: modelling for constructive simulation, including initial behavioural analysis and performance evaluation; wargames and exercises to collect data to test the validity of the concepts and the extent and nature of capability improvement in more realistic settings; and finally, the refinement of the models through calibration with the wargame and exercise results in order for the models to be used for validation, sensitivity analysis and performance evaluation.

A logistics system can be viewed as a distributed system, with the characteristics of resource sharing and distribution, geographical separation of interacting components, and concurrent operation. It involves both physical networks (such as road, rail and sea) for transporting goods, materials and personnel, and information networks that may be used to communicate requests for supplies, system status and commands to carry out various actions related to logistics support. This makes a logistics system a more general distributed system than say that of computer networks. It has been our experience that many useful insights have been gained from applying formal modelling techniques to computer networks (e.g. [3]). In particular, because distributed systems involve concurrency, it should be useful to investigate the use of models of concurrency in the logistics domain. Thus the aim of this paper is to investigate the application of Coloured Petri Nets (CPNs) [11, 12] to the problem of creating an ADF logistics network model suitable for concept and capability studies in accordance with the M-E-M paradigm.

Logistics systems are by nature very complex and are thus a very good example for testing the limits of our tools and techniques, firstly just for their formal specification but with an eye to their analysis. We thus require a technique that is quite expressive because the data types that we need to model are quite complex. This leads us to use CPNs as implemented in Design/CPN [7] (and CPN Tools [5]).

1.2 Related Work

Van der Aalst developed a timed coloured Petri net framework for the modelling of business logistics systems [1]. This approach, however, considers relatively simple examples, using simple constructs and therefore does not model to the depth required for application to military logistics. Petri nets have also been used to model aspects of logistics systems such as supply chains [19–21], manufacturing [16, 22], transport systems [6, 15], and other logistics functions [4, 8]. CPNs have been successfully applied to operations planning in a military environment [13, 14, 24, 25], however, to the best of our knowledge, CPNs have not been previously employed to model logistics systems in the military domain.

1.3 Overview and Contribution

This paper describes an initial attempt to model a military logistics network, to a significant level of detail. We start by identifying the key components of such a network and illustrate its operation with a simple example (Section 2). We provide some insight into the architectural design of the CPN model and record our key modelling assumptions before the model is described (Section 3). The paper then relates some of our experience in using Design/CPN and CPN Tools to build and check the model (Section 4). Concluding remarks and the identification of future work can be found in Section 5.

The major contribution of this paper is the CPN formalisation of the characteristics and dynamics of a Defence Logistics Network. This represents a significant undertaking due to the size and complexity of the domain, but it also contributes to the knowledge and experience of applying CPNs to the modelling of large complex systems. The CPN model provides a formal and unambiguous specification of the characteristics and dynamic behaviour of a logistics network, which has great intrinsic value as a vehicle for facilitating a common understanding of the behaviour of such systems and for the education of others. It also provides a basis for the formal analysis of such systems. Finally we discuss some limitations of CPNs and their tools in this context.

1.4 Project Management and Effort

This paper reports the outcomes of a 6 month project initiated and managed by the Logistics Mission of the Land Operations Division within the Australian Defence Science and Technology Organisation (DSTO). It began with a 4-day course on CPNs given by the Computer Systems Engineering Centre (CSEC) to DSTO and experts of the ADF logistics domain. This was followed by a 1-day workshop to begin the process of modelling Defence logistics. A joint project team was formed to develop the model. It comprised three DSTO staff members from the Logistics Mission, three experts from the ADF logistics domain and three CSEC staff members with CPN expertise to develop the models. Team meetings were held regularly (mostly once a week) to discuss progress and make decisions on the important areas to be modelled. These meetings were relatively formal and included: minutes to record the proceedings and decisions; power point presentations on the model by CSEC; and presentations by DSTO to provide further requirements and details of the system to be modelled. This was supplemented by adhoc meetings and emails between the CSEC and DSTO teams to clarify various issues and obtain further input to the model. The development of the CPN model and its documentation [9] by UniSA personnel has taken approximately 8 person-months.

2 A Defence Logistics Network

This section introduces the notion of a Defence Logistics Network [2]. We have identified five key physical components of such a logistics network:

Nodes are physical locations for the organisation and redistribution of assets and/or the transition point from one form of transport to another. A node can be as simple as a cache or dump, or as complex as a major military base.

Modes (Transporters) refer to the physical means of transportation of materiel and personnel and are usually general descriptions of the *mode* of transport, e.g. Sea, Air, Road, Rail. When referring to specific characteristics of the physical means of transportation we will use the term *transporters*.

Links are transport paths between nodes. The eligibility of transporters to travel over links is directly influenced by specific link characteristics.

Cargo is the general term for the assets that are to be distributed to nodes via the logistics network. When in a node, cargo can be referred to as *stock*.

Modules refer to a number of standard containers into which cargo is packed for storage or for movement from one node to another.

An *information network* exists, whereby knowledge is disseminated to nodes and transporters. A *control* framework also exists, implementing mechanisms that encompass aspects of the information and physical networks, and which dictates activities and actions that are to be performed. However, our focus is on the physical network and we abstract from both the information network and control mechanisms to a large extent (see Section 3.2).

To give an intuitive feel, consider the illustrative example shown in Fig 1. This figure presents a simple logistics network consisting of four nodes, drawn as circles, and three links, drawn as arcs connecting nodes. Each of the nodes is considered to reside in a major Australian town, hence the

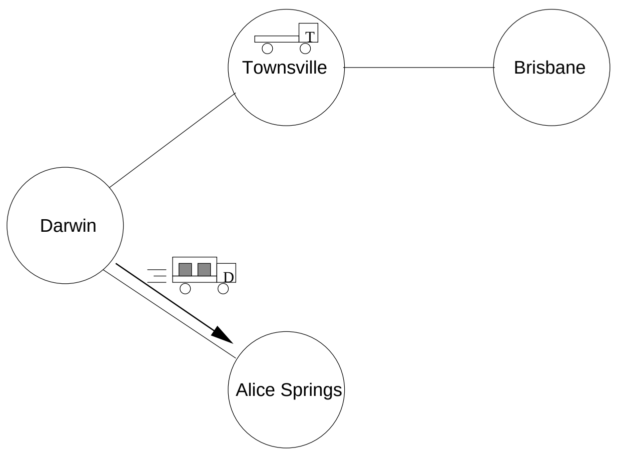


Fig. 1. An example of the topology of a logistics network.

names Darwin, Townsville, Alice Springs and Brisbane are given to the nodes. The node Darwin produces cargo for distribution to and consumption by the rest of the network. Two transporters are shown: one laden with cargo (packed in modules) travelling from Darwin to Alice Springs, and the other stationary in Townsville. Cargo is being sent to Alice Springs from Darwin in response to a demand generated by Alice Springs. Cargo flows in a *downstream* direction, depicted as left to right in Fig. 1. This system is known as a *pull* logistics system, where nodes pull resources towards themselves from supplying nodes. This example is revisited after each of the five key components are described in more detail below.

2.1 Nodes

Aside from their current stock levels, nodes have a number of characteristics that relate to the distribution and movement of assets. These can be broken up into three categories: *Accessibility*, including depth and wharf size for sea ports, landing facilities and runway length for air ports, and the maximum parking area capacity and carriageways for vehicles; *Capacity*, including maximum storage area capacity, berths/anchorage, and refuelling capabilities; and *Throughput*, resulting from characteristics of the discharge equipment, storage areas, stacking equipment, parking equipment, and transshipping equipment.

2.2 Transporters

Besides the general ‘mode of transport’ descriptions like Sea, Air, Road or Rail, transporters may have more specific characteristics such as: maximum payload, both volume and weight; maximum weight, both loaded and unloaded; maximum speed and range; mobility category (for land-based vehicles); and landing/docking/loading/unloading requirements and equipment (for air- and sea-based vehicles).

2.3 Links

Links have characteristics that are similar in nature to those of transporters. For example, a road link may have characteristics such as Surface, Mobility, Capacity (weight and number of transporters), Width, Maximum speed (daytime and nighttime), and other restrictions. The eligibility of a particular transporter for a particular link is determined by comparing the transporter and link characteristics, e.g. a segment of unsealed road will be unsuitable for some wheeled transport modes. Another example is that an inland waterway will dictate a waterbourne transporter.

2.4 Cargo

The ADF use 9 predefined *classes* of cargo. Examples include food, medical supplies, spare parts, ammunition, and fuels, oils and lubricants. Cargo characteristics that are relevant to distribution

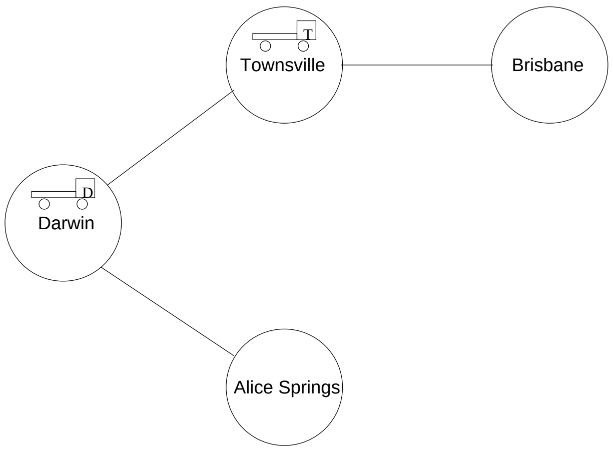


Fig. 2. Example logistics network.

include cargo dimensions, weight, density, any special handling needs, and any hazardous properties of the cargo. Cargo is usually containerised (as discussed below). There are numerous restrictions on the classes of cargo that can be packed together. For example, fuel cannot be packed with ammunition or food. When packing cargo there is a tendency for cargo containers to bulk out before they weigh out, i.e. it is normally the space restriction, not the weight restriction, that controls how much cargo can be packed into a given container.

2.5 Modules

There are a number of standard containers into which cargo is packed. For the purposes of this task, we only consider three broad types of module: boxes (smallest); pallets; and ISO shipping containers (largest). Any smaller module can be nested either on or in a larger sized module. For example, boxes can be stacked onto pallets or in ISO containers, and pallets can be stored inside ISO containers. The capacity of larger modules is specified in terms of smaller modules (see Section 3.2).

The box is defined as the smallest unit of cargo that can exist in the system. The weight of a module is determined from the weight of the empty module plus the weight of any cargo and smaller modules carried by the module. The weight of cargo is determined by the number of boxes-worth of cargo multiplied by the weight (specified as kilograms per box).

2.6 Illustrative Example

We consider a 4 node network as shown in Fig. 2. Each node is considered to exist at a major Australian town. The downstream direction is again from left to right. The ‘top-level’ node, Darwin, is a production node and produces cargo for consumption in the lower-level nodes. Cargo is moved by transporters, each of which has a home node to which they always return. In this example (and the following model in Section 3), transporters are restricted to travelling on links from their home node to a node one level downstream, and then returning. For example, to move cargo from Darwin to Brisbane two transporters are required: one from Darwin to Townsville and another from Townsville to Brisbane.

Cargo is always carried in modules. Cargo in modules and empty modules cannot be carried at the same time by a transporter. Transporters that return to their home node from a destination node only carry empty modules held by the destination node at the time of cargo delivery. Thus cargo in modules travels in the downstream direction and empty modules travel upstream.

Modules become empty through the process of cargo consumption. As cargo is consumed the number of modules required to contain the remaining cargo decreases. Any modules held in the consuming node over this amount are considered empty. The consumption of cargo reduces the holdings of cargo in nodes. When the level of cargo falls below a *reorder* threshold, the node orders more cargo of the class(es) at or below the reorder threshold of that node. This cargo is then supplied by a node one level upstream of the ordering node. Although not shown in this example, if more than one upstream node is eligible to supply cargo for a given order then any one could be chosen non-deterministically. As cargo transportation is driven by orders of downstream nodes, the logistics model is 'pull-driven'.

It is assumed that each downstream node initially holds no cargo and no modules, and that the Darwin and Townsville nodes are home to one transporter each, labelled by the letters 'D' and 'T' respectively in Fig. 2.

Ordering of Cargo In this example, only two classes of cargo have been considered. Each of the nodes holds no cargo initially, except for the producing node, Darwin, which is considered to hold an infinite amount of cargo. This means that every non-producing node is eligible to order. The amount of cargo ordered by a node is derived from the cargo thresholds of that node, indicating the maximum, minimum and re-order cargo levels of a node. The maximum is the amount of cargo that a node will reorder to. The minimum threshold is the level of cargo a node will not go below when setting aside cargo for a downstream order. The maximum, minimum, and reorder thresholds for both cargo classes of each node for this example are Alice Springs:(5,5),(0,0),(2,2), Townsville:(6,6),(1,1),(2,2), and Brisbane:(4,2),(0,0),(2,1). The minimum stock level for Alice Springs and Brisbane are 0 for both cargo classes as they are *leaf nodes*, i.e. they have no nodes further downstream that may require cargo. As each non-producing node initially holds no cargo, it orders up to its maximum level for all classes.

Assigning Cargo An upstream node may attempt to supply an order if it has sufficient cargo or it is a cargo-producing node. For example, to satisfy the order of Alice Springs, the Darwin node, being a producing node and the only immediate upstream node of Alice Springs, sets aside 5 boxes of both Class 1 and Class 2 cargo. However, Townsville cannot satisfy the order of Brisbane as it has no cargo.

Packing, Loading, and Sending Cargo The cargo destined for Alice Springs is then packed into modules and loaded onto an idle transporter. In our illustrative example, we assume that the two classes of supply, Class 1 and Class 2, cannot be mixed at the pallet level, but can be mixed at the ISO container level. We also instigate the rule that modules are to be nested if more than two sub-modules need to be moved, i.e. two or more boxes need to be placed on a pallet, and two or more pallets need to be put into an ISO container. Hence, 10 boxes, 2 pallets and 1 ISO container are used to transport 5 boxes of Class 1 and 5 boxes of Class 2 cargo to the destination of Alice Springs. Large orders can be split over multiple transporters (not shown in this example), but each transporter may deliver to only one node before returning.

Cargo Holdings and Order Status A transporter departs the Darwin node bound for Alice Springs, as depicted in Fig. 3. The cargo holdings and order status for all of the nodes are shown in Fig. 3. The numbers in the two boxes below the node's label show the current cargo holdings of the node. As Darwin is a producing node it is considered to have an infinite amount of Class 1 and Class 2 cargo. All nodes downstream of Darwin have no Class 1 nor Class 2 cargo as they are yet to be supplied. The order status of each non-producing node is shown below and to the right of the node. The amount of cargo *outstanding* (ordered but not yet addressed by an upstream node) is labelled O, and the amount of cargo *pending* (ordered and addressed by an upstream node, but not yet delivered) is labelled P. Townsville's and Brisbane's order status both reflect an outstanding amount of cargo equivalent to their respective stock level maximums and no cargo pending for either node, as no cargo has been set aside for them by an upstream node. Conversely, the order status of Alice Springs shows 5 boxes of both Class 1 and Class 2 cargo pending, as this is the amount of cargo set aside, loaded, and sent by Darwin. The module holdings of each node are not shown to simplify the diagrams.

A transporter that is en route to a destination may subsequently arrive at its destination, or it may return to its origin node or be redirected to another eligible node. In this case the transporter is redirected whilst en route to Alice Springs to the Townsville node. This is shown in Fig. 4.

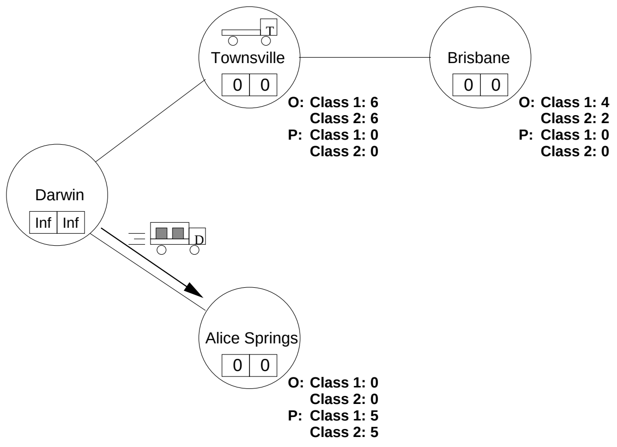


Fig. 3. A transporter leaving a node.

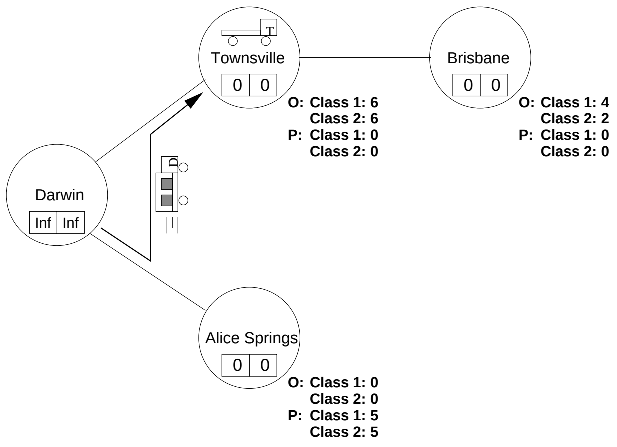


Fig. 4. A transporter redirected en route.

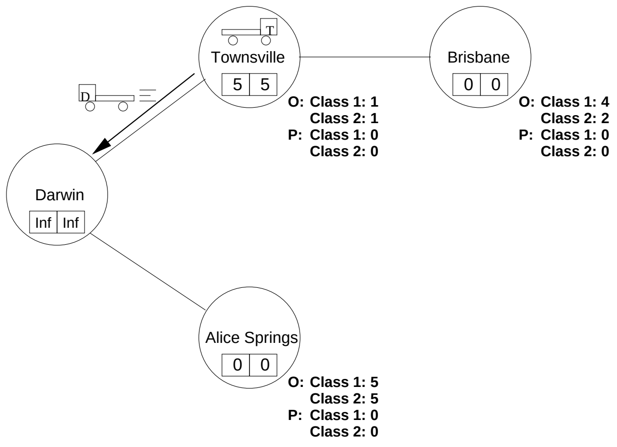


Fig. 5. A transporter returning after being received.

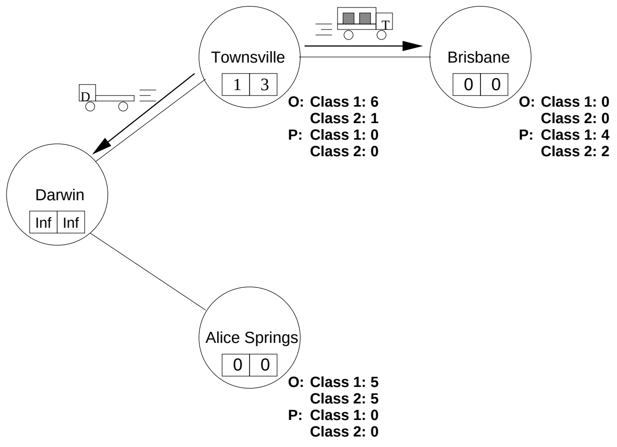


Fig. 6. A transporter returning and a node further downstream being supplied.

Receiving a Transporter at a Node and Order Reconciliation An incoming transporter may be received by a node, or it may be redirected or undergo transshipment. The transporter from Darwin is received by the Townsville node. The cargo and modules it was carrying are unloaded and the transporter returns to its home node of Darwin, as depicted in Fig. 5. Townsville now holds 5 boxes worth of each cargo class and the corresponding 10 boxes, 2 pallets and one ISO container that were used to ship the cargo. The order status for both Townsville and Alice Springs is now updated. Townsville now has 1 box of Class 1 and Class 2 cargo outstanding, whilst Alice Springs has 5 boxes of both Class 1 and Class 2 cargo outstanding and no cargo pending, as Townsville received the cargo Alice Springs was expecting.

Supply of Cargo to Nodes further Downstream The Brisbane node may be delivered some cargo as the node immediately upstream, Townsville, now holds cargo. The Townsville node sets aside as much cargo as it can, without going below its minimum cargo levels, in an attempt to satisfy the outstanding cargo amount of the Brisbane node. In this case it may set aside enough cargo to fully satisfy Brisbane's order. This reduces Townsville's Class 1 cargo level below its reorder threshold of 2 boxes, and thus it orders up to its maximum of that class, 6 boxes. The order status of Brisbane now shows no cargo as outstanding and four boxes of Class 1 and 2 boxes of Class 2 as pending. It then loads a transporter of its own with cargo destined for Brisbane, which subsequently departs. The transporters returning to Darwin and en route to Brisbane are depicted in Fig. 6.

3 Model Description

Although the example in Section 2.6 may seem simple, it will be seen that the model of the components themselves is comprehensive and the generic nature of the model allows large logistics networks to be instantiated. The modelling tool Design/CPN [7] was chosen as the computer tool with which this modelling work would be undertaken. The primary rationale was that Design/CPN provided integrated state space visualisation facilities and simulation-based performance analysis (although the latter is now also supported by CPN Tools [5]), and that automated support exists for the translation of a Design/CPN model to CPN Tools. This section introduces the architectural design philosophy used, the modelling scope and assumptions, and finally introduces the CPN model itself.

3.1 Architectural Design

During the development of our logistics network CPN we attempted to incorporate the following desirable features of a high-level architectural design. *Extensibility*: the design should allow for easy addition of detail and extensions to the model. *Modularity*: well-defined interfaces between components allow components to be developed independently and aids extensibility. *Precision*: the architecture must accurately reflect the real-life system, to an appropriate level of abstraction. *Future-proofing*: the architecture should guard against possible future changes to the greatest extent possible.

There were two main trade-offs that we considered during the development of our model. The first was *Compactness versus Readability and Understandability*, which incorporates trade-offs in visualisation of control flow and data flow, model size, extensibility, ease of debugging and ease of maintenance. The second was *Generality versus Specificity*. A very specific model of a particular scenario may not be so easily adapted to other scenarios. However, a very general model may capture too much and be too difficult to analyse. This also relates to the flexibility of the model.

Nodes and links can be considered as relatively static *objects*, which could be considered as *processes*. Transporters, on the other hand, traverse links and move from one node to another. We therefore modelled nodes and links as (substitution) transitions, and transporters as tokens that traverse the CPN model. Interfaces between nodes and links contain information, and are modelled as shared places.

The logistics network CPN model underwent many changes during its development [9]. Here we just discuss the representation of network topology. The topology was encoded in tokens, resulting in *generic node* and *generic link* pages as 'execution engines'. This architectural design allowed the modelling of any topology without requiring structural changes to the model, as node and link additions are accomplished by the addition of tokens. This facilitates modelling dynamic topology changes 'on-the-fly', for example the unavailability of a given inter-nodal link for a temporary period could be modelled through token manipulation. A further advantage was the creation of a compact model, at the cost of losing the ability to visualise the network topology (i.e. net elements such as substitution transitions corresponding directly to nodes or links). This meant that the number of model pages remained the same as nodes and links were added, compared with a combinatorial explosion of pages if the topology was modelled using net structure.

3.2 Modelling Scope and Assumptions

The model complexity was reduced to only that required to represent the behaviour of interest. We have abstracted from both the information network and the control mechanisms and embedded as little control in the model as possible, with a view to creating a model of the physical network as a test-bed for evaluation of particular control strategies. We have assumed that the information network is perfect, in that the knowledge that nodes have about each other is correct and updated instantaneously for all nodes.

For the initial modelling work documented in this paper, we have only considered a *pull* (demand driven) procedure for the distribution of assets, whereby downstream nodes request assets from upstream nodes. We do not consider a *push* procedure or *anticipatory* logistics, where upstream nodes direct assets to be sent to downstream nodes.

Nodes Storage area capacity and stock level characteristics were identified as the most relevant and important for our initial model of a node, as they are necessary for measuring effectiveness and performance. As previously discussed, we consider each node to have a maximum stock level, a minimum stock level, and a reorder threshold for each class of cargo modelled. In addition, we have modelled the following 6 activities identified as the core activities of a node:

- **Send:** a node sends a transporter carrying cargo to another node.
- **Receive:** a node receives a transporter (empty or carrying cargo) from another node.
- **Transship:** cargo is moved from one transporter to another.
- **Redirect:** transporters are directed away from this node towards an another node.
- **Consumption:** supplies are consumed within a node.
- **Reordering:** supplies are requested when the stock levels of one or more classes of supply fall below a predetermined level (the reorder threshold).

Transporters The maximum payload (both volume and weight), the maximum weight of specific transporters (both loaded and unloaded), and the maximum speed and range of a transporter were considered to be important for our initial model. The maximum volume of payload is specified in terms of modules (see below). We define a *home node* for each transporter, as the node where it resides when it is idle. We do not consider landing/docking/loading/unloading characteristics of transporters, in line with our modelling assumptions and scope for the modelling of nodes. We also do not consider the breakdown (failure) of transporters in this initial model.

Links We consider links to be either unidirectional or bidirectional, and if bidirectional, half-duplex (allowing one direction of travel at a time) or full duplex (allowing travel in both directions simultaneously). We consider the link characteristics to be the broad description (e.g. Sea, Air, Road, Rail), length, and maximum traversal speed only. The following link behaviours are included in our model:

- **Successful traversal of the link**;
- **Redirect en-route**, where a transporter changes its destination and is directed towards another destination, provided the eligibility requirements of the transporter and link are met; and
- **Return**, where a transporter returns to its origin without traversing the entire link.

We make the assumption that the flow of supplies are acyclic. This allows for an unambiguous definition of an upstream and downstream direction over each link, as introduced in Section 2. We also make the assumption that the topology is static.

In addition, the domain of each transporter is restricted to be its home node and all nodes immediately downstream of its home node (i.e. the nodes directly *supplied* by the home node). This reflects the assumption that we are only considering a pull (demand) driven logistics system, in which it is only possible for orders to be satisfied by the nodes immediately (one hop) upstream from the ordering node. Hence, the movement of transporters is restricted to only one node downstream.

Cargo Cargo and supplies have been considered in an abstract way. In this example we consider only two distinct classes of cargo, rather than 9. We did not abstract to a single class of cargo, as we wished to keep in mind the implications for extending the model to cover more than two classes of cargo. Modelled characteristics of cargo include the cargo class and the cargo weight (density).

Modules We assume that each pallet can carry a maximum of 12 boxes, and that each ISO shipping container can hold a maximum of 20 pallets or $20 * 12 = 240$ boxes (without pallets). It was not important to keep track of empty boxes, but we do keep track of empty pallets and ISO shipping containers. We assume that all cargo is boxed, for simplicity.

As mentioned in Section 2.4, there are numerous rules and restrictions on the classes of cargo that can be packed and stored together. Given that we only model two classes of cargo, we abstract from

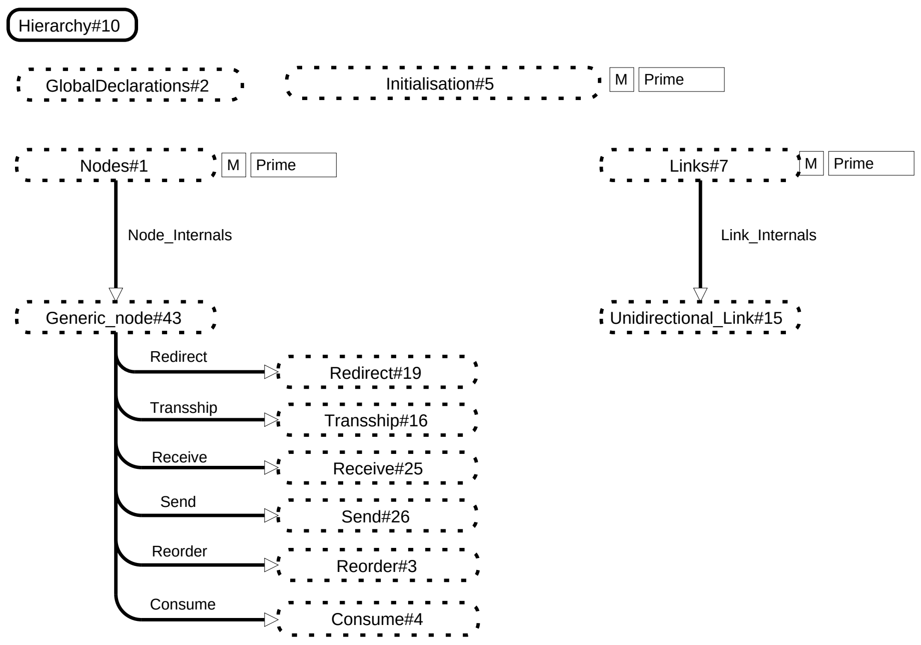


Fig. 7. Model Hierarchy.

these rules and restrictions by considering a policy of *cargo segregation* at the level of the pallet. As in the example, when packing cargo into modules, we have followed the principle that if two or more modules of the same size are included in a load then they must be loaded on/in a larger module (but only without violating cargo segregation).

3.3 Model Description

The hierarchy page, shown in Fig. 7, gives an overview of the architecture of the model. The model contains 11 pages (excluding the hierarchy and global declarations pages), 69 places (including port and socket places and separate instances of fusion places), 8 substitution transitions, and 13 concrete transitions. This only tells part of the story, as substantial complexity is modelled in the inscriptions due to the high degree of folding in this model.

The model is divided into two sections, one with the Nodes page at the head modelling logistics nodes, and the other with the Links page at the head modelling inter-nodal links. The hierarchical structure of pages modelling nodes has three layers. At the third layer, six pages represent the six actions identified as core actions of a node: Redirect, Transship, Receive, Send, Reorder and Consume. The hierarchical structure of pages modelling links is much simpler, with only one page at each of the two hierarchical levels.

Separate from the node and link pages are two other pages: Global Declarations, which contains all the colour set definitions, variable declarations, and functions; and the Initialisation page which is used to populate the net with its initial marking. This design allows for the population of places in the net with a pseudo-initial marking via functions rather than through an explicit hard-coded initial marking. Modification of this pseudo-initial marking is possible by editing the function declarations of the Global Declarations page rather than editing the net pages directly. Using functions in this way for the initial marking provides a mechanism where, in the future, the model can be populated with a particular scenario from an external source, e.g. a file or a Transmission Control Protocol (TCP) connection. Due to space limitations, we will describe the Nodes page and some of its subpages, but not the Links or Unidirectional_Link pages. Details of these pages can be found in [9].

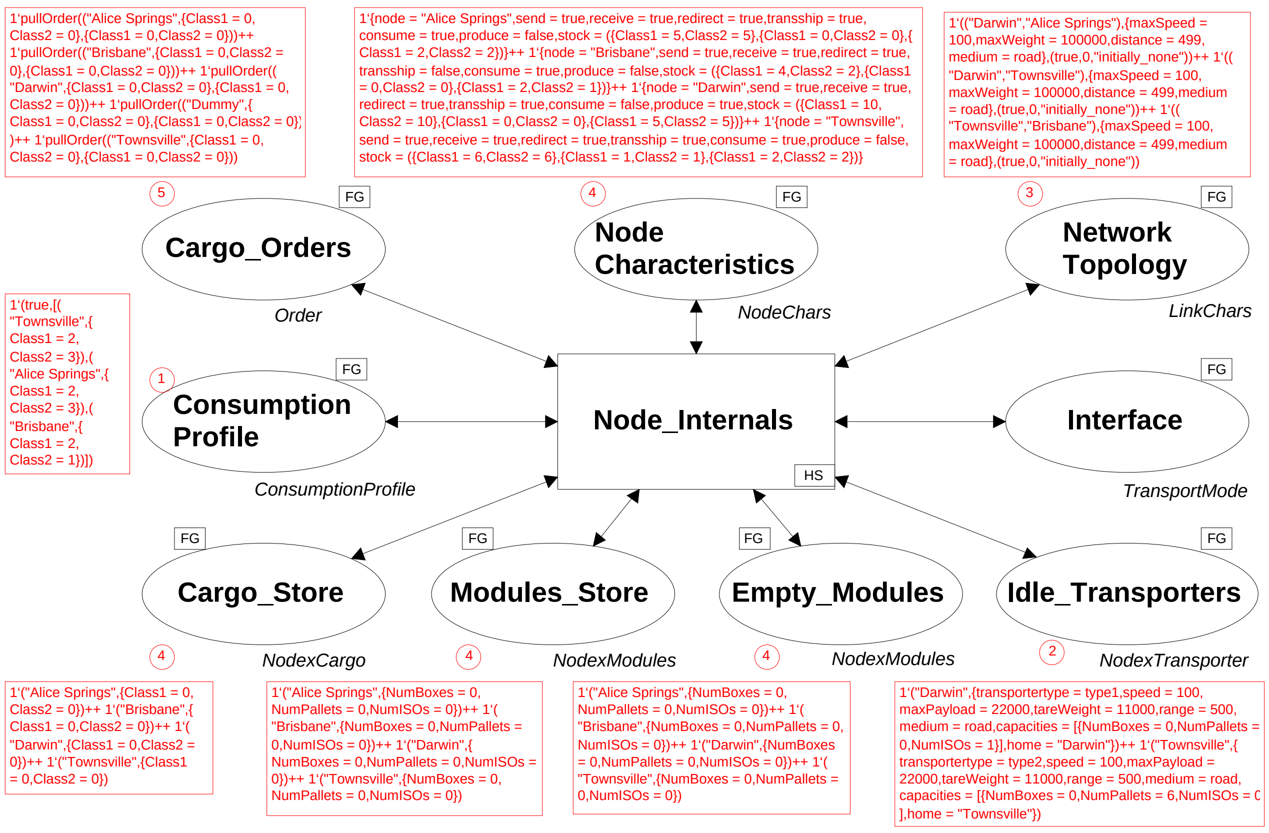

1'pullOrder(("Alice Springs",{Class1 = 0, Class2 = 0},{Class1 = 0,Class2 = 0}))++ 1'pullOrder(("Brisbane",{Class1 = 0,Class2 = 0},{Class1 = 0,Class2 = 0}))++ 1'pullOrder(("Darwin",{Class1 = 0,Class2 = 0},{Class1 = 0, Class2 = 0}))++ 1'pullOrder(("Dummy",{ Class1 = 0,Class2 = 0},{Class1 = 0,Class2 = 0}))++ 1'pullOrder(("Townsville",{Class1 = 0, Class2 = 0},{Class1 = 0,Class2 = 0}))
1'{node = "Alice Springs",send = true,receive = true,redirect = true,tranship = true, consume = true,produce = false,stock = ({Class1 = 5,Class2 = 5},{Class1 = 0,Class2 = 0},{ Class1 = 2,Class2 = 2})}++ 1'{node = "Brisbane",send = true,receive = true,redirect = true, tranship = false,consume = true,produce = false,stock = ({Class1 = 4,Class2 = 2},{Class1 = 0,Class2 = 0},{Class1 = 2,Class2 = 1})}++ 1'{node = "Darwin",send = true,receive = true, redirect = true,tranship = true,consume = false,produce = true,stock = ({Class1 = 10, Class2 = 10},{Class1 = 0,Class2 = 0},{Class1 = 5,Class2 = 5})}++ 1'{node = "Townsville", send = true,receive = true,redirect = true,tranship = true,consume = true,produce = false, stock = ({Class1 = 6,Class2 = 6},{Class1 = 1,Class2 = 1},{Class1 = 2,Class2 = 2})}
1'(("Darwin","Alice Springs"),{maxSpeed = 100,maxWeight = 100000,distance = 499, medium = road},(true,0,"initially_none"))++ 1'(("Darwin","Townsville"),{maxSpeed = 100, maxWeight = 100000,distance = 499,medium = road},(true,0,"initially_none"))++ 1'(("Townsville","Brisbane"),{maxSpeed = 100, maxWeight = 100000,distance = 499,medium = road},(true,0,"initially_none"))
5
FG
Cargo_Orders
Order
4
FG
Node Characteristics
NodeChars
3
FG
Network Topology
LinkChars
1'(true,[("Townsville",{ Class1 = 2, Class2 = 3}),("Alice Springs",{ Class1 = 2, Class2 = 3}),("Brisbane",{ Class1 = 2, Class2 = 1})])
1
FG
Consumption Profile
ConsumptionProfile
Node_Internals
HS
FG
Interface
TransportMode
FG
Cargo_Store
4
NodexCargo
FG
Modules_Store
4
NodexModules
FG
Empty_Modules
4
NodexModules
FG
Idle_Transporters
2
NodexTransporter
1'("Alice Springs",{Class1 = 0, Class2 = 0})++ 1'("Brisbane",{ Class1 = 0,Class2 = 0})++ 1'("Darwin",{Class1 = 0,Class2 = 0})++ 1'("Townsville",{Class1 = 0,Class2 = 0})
1'("Alice Springs",{NumBoxes = 0, NumPallets = 0,NumISOs = 0})++ 1'("Brisbane",{NumBoxes = 0,NumPallets = 0,NumISOs = 0})++ 1'("Darwin",{ NumBoxes = 0,NumPallets = 0,NumISOs = 0})++ 1'("Townsville",{NumBoxes = 0, NumPallets = 0,NumISOs = 0})
1'("Alice Springs",{NumBoxes = 0, NumPallets = 0,NumISOs = 0})++ 1'("Brisbane",{NumBoxes = 0,NumPallets = 0, NumISOs = 0})++ 1'("Darwin",{NumBoxes = 0,NumPallets = 0,NumISOs = 0})++ 1'("Townsville",{NumBoxes = 0,NumPallets = 0,NumISOs = 0})
1'("Darwin",{transportertype = type1,speed = 100, maxPayload = 22000,tareWeight = 11000,range = 500, medium = road,capacities = [{NumBoxes = 0,NumPallets = 0,NumISOs = 1}],home = "Darwin"})++ 1'("Townsville",{ transportertype = type2,speed = 100,maxPayload = 22000,tareWeight = 11000,range = 500,medium = road, capacities = [{NumBoxes = 0,NumPallets = 6,NumISOs = 0],home = "Townsville"})

Fig. 8. The Nodes Page.

Node Hierarchy The top-level page of the node hierarchy, the Nodes page from Fig. 7, is shown in Fig. 8. The Node_Internals substitution transition interacts with a set of places described below. The marking shown is the initial marking corresponding to the illustrative example from Section 2.6.

Node Characteristics Each individual logistics node is represented by a token in the Node Characteristics place. The associated colour set, NodeChars, is defined as a record on lines 13 to 21 in Listing 1.

The 'node' field (line 14) is typed by the colour set, NodeID (lines 1 and 2), which defines a subset of strings constituting legal names for the physical nodes. The last string of the subset, initially_none, is used as the initial transporter direction on an inter-nodal link.

The six boolean fields (lines 15 to 20) denote whether a specific node can perform the corresponding functions of sending, receiving, redirection, transshipping, consumption or production of cargo. These fields are used as an enabling mechanism implemented via transition guards. This structure allows for the creation of nodes which may only perform a limited number of functions, such as transshipment nodes that do not hold cargo of their own, as well as providing the ability to alter node functionality 'on-the-fly' simply by changing the boolean values in the corresponding token.

Finally, the 'stock' field (line 21), typed by the StockLevels colour set (lines 8 to 11), specifies the maximum, minimum, and reorder levels for cargo held at the node. These three levels are typed by the Cargo colour set (lines 4 to 6), which is a record consisting of two integers representing the number of boxes of each of two classes of cargo.

Network Topology The Network Topology place contains information regarding the inter-nodal links of the model. Each token in this place represents the information for one link joining a pair of nodes, and is typed by the colour set LinkChars, shown on lines 18 to 21 of Listing 2.

The LinkID colour set (lines 1 to 3) is a product of two node IDs, defining a pair of nodes joined by a link. The first node of the product is interpreted by the model as being one level upstream of the

Listing 1. Colour set Declarations for the Node Characteristics place.

```
1  color NodeID = subset STRING with ["Darwin","Townsville","Brisbane","Alice Springs",
2    "initially_none"];
3
4  color Cargo = record
5    Class1 : INT *
6    Class2 : INT;
7
8  color StockLevels = product
9    Cargo * (* Max Stock Levels *)
10   Cargo * (* Min Stock Levels *)
11   Cargo;  (* Reorder Threshold *)
12
13 color NodeChars = record
14   node      : NodeID * (* The node with these characteristics *)
15   send      : BOOL * (* Node can send *)
16   receive   : BOOL * (* Node can receive *)
17   redirect  : BOOL * (* Node can redirect *)
18   transship : BOOL * (* Node can transship *)
19   consume   : BOOL * (* Node can consume cargo *)
20   produce   : BOOL * (* Node can produce cargo *)
21   stock     : StockLevels; (* Max, Min and Reorder Threshold *)
```

Listing 2. Colour set Declarations for the Network Topology place.

```
1  color LinkID = product
2    NodeID * (* source/destination *)
3    NodeID;  (* destination/source *)
4
5  color LinkType = with air | rail | sea | road;
6
7  color LinkInfo = record
8    maxSpeed  : INT *      (* Max speed of this link *)
9    maxWeight : INT *      (* Max weight of this link *)
10   distance  : INT *      (* length (distance) of this link *)
11   medium    : LinkType;  (* The medium of this link *)
12
13 color LinkDirectionSynch = product
14   BOOL *  (* full duplex (true) or half duplex (false) *)
15   INT *   (* number in transit *)
16   NodeID; (* last destination/direction of flow *)
17
18 color LinkChars = product
19   LinkID *   (* identifer for this link *)
20   LinkInfo * (* the characteristics of this link *)
21   LinkDirectionSynch; (* Direction synchronisation information *)
```

second node. The LinkInfo colour set (lines 7 to 11) specifies the characteristics of the link, where the transport medium is described by the enumerated colour set, LinkType, on line 5. These characteristics define the eligibility criteria for transporters to traverse this link. The LinkDirectionSynch colour set (lines 13 to 16) provides a means of coordinating the directions of link traversal and enforcing traversal in one direction at a time for half-duplex links.

Interface The node and link pages of the model interact via the Interface place, typed by the TransportMode colour set shown on lines 25 to 26 in Listing 3. A TransportMode token specifies a transporter (laden or unladen) entering or leaving a given node. The Transporter field specifies an individual transporter. The Transporter colour set (lines 10 to 18) is a record specifying a TransporterType (line 6), which may be thought of as a transporter descriptor, as well as a number of characteristics specific to

Listing 3. Colour set Declarations for the Interface place.

```
1  color Modules = record
2    NumBoxes    : INT *
3    NumPallets  : INT *
4    NumISOs     : INT;
5
6  color TransporterType = with type1 | type2 | type3;
7
8  color Capacity = list Modules;
9
10 color Transporter = record
11   transportertype : TransporterType *
12   speed : INT *
13   maxPayload : INT *
14   tareWeight : INT *
15   range : INT *
16   medium : LinkType *
17   capacities : Capacity *
18   home : NodeID;
19
20 color Load = product Modules * Cargo * NodeID;
21   (* Includes Original Destination *)
22
23 color Direction = with outgoing | in_transit | incoming | returning;
24
25 color TransportMode = product Transporter * Load * NodeID * NodeID * Direction;
26                            (* Transporter, Load, Source, Destination, Direction *)
```

Listing 4. Colour set Declarations for Cargo_Orders.

```
1  color PullOrder = product
2    NodeID * (* The node which generated the order/requested the supplies *)
3    Cargo *  (* The supplies requested that are not yet on their way
4                (outstanding but not pending) *)
5    Cargo;   (* The supplies that have been sent but have not yet arrived (pending) *)
6
7  color Order = union pullOrder:PullOrder; (* A union, for future-proofing against
8                                               other types of orders/supply schemes *)
```

that individual transporter, including speed (for use in future performance analysis), maximum weight of payload, unloaded (tare) weight, range, and the medium via which the transporter can travel. The capacities field, typed by colour set Capacity (line 8), is a list of records of type Modules (lines 1 to 4), and details the possible maximum legal module configurations (there may be more than one) which may be loaded onto the transporter, thus giving the volumetric limitations of the transporter. For example, a truck may be able to carry either at most one ISO container or 10 pallets outside an ISO container. Finally, the home field specifies the home location of the transporter.

The second element in the TransportMode colour set is the Load (lines 20 and 21). It specifies the number of boxes of each cargo class, the modules used to pack this cargo, and the original destination of the particular load. This last field is necessary to keep track of the original destination of the cargo, for order reconciliation when a transporter is redirected.

The two NodeID terms of the TransportMode colour set define the *current* origin and destination nodes of the transporter. The Direction term (defined on line 23) specifies whether the transporter is outgoing, in_transit, incoming, or returning. This is used by the model to determine the eligible actions of the transporter when arriving at or departing a node, or when in transit.

Cargo_Orders The Cargo_Orders contains tokens representing the current order information for each node. It is typed by colour set Order, (line 7 of Listing 4) which corresponds to a PullOrder (lines 1

Listing 5. Colour set Declarations for Consumption Profile and Cargo_Store.

```
1 color NodexCargo = product NodeID * Cargo;
2 color NodexCargoList = list NodexCargo;
3 color ConsumptionProfile = product BOOL * NodexCargoList;
```

Listing 6. Colour set Declarations for Modules_Store, Empty_Modules and Idle_Transporters.

```
1 color NodexModules = product NodeID * Modules;
2 color NodexTransporter = product NodeID * Transporter;
```

to 5) in our current model. PullOrder comprises: the identifier of the node placing the order (NodeID); the amount of cargo outstanding; and the amount of cargo pending (both of type Cargo).

Consumption Profile The declarations for the Consumption Profile place, which holds a list detailing the cargo consumption information for the nodes, is shown in Listing 5. This *consumption profile* (line 2) is modelled as a list of pairs (line 1): the node (NodeID) at which consumption occurs and the amount of cargo to be consumed (Cargo) in a particular consumption event. Thus cargo consumption proceeds by stepping through the elements in this list. This list is paired with a boolean in the ConsumptionProfile colour set (line 3) which specifies whether the consumption profile *cycles*, i.e. the consumption events in the consumption profile repeat from the start when the end of the profile is reached.

Cargo_Store, Modules_Store, Empty_Modules and Idle_Transporters Cargo_Store contains tokens representing a stock level (line 1 of Listing 5). Similarly, Modules_Store and Empty_Modules describe the numbers of modules currently being used to store cargo and empty modules, respectively (line 1 of Listing 6). Idle_Transporters contains transporters that are available for use (line 2 of Listing 6). In the colour sets of all four places, the NodeID specifies where the cargo, modules and transporter are currently situated.

The Generic_node Page and Structure The Generic_node page, shown in Fig. 9, shows the interactions of the logistics functions Send, Receive, Transship, Redirect, Consume and Reorder. Each of these logistics functions is represented on this page by a substitution transition. All of the substitution transitions are connected by bidirectional arcs to the Node Characteristics place, as all actions require knowledge of the node characteristics.

The substitution transitions that model actions in which a transporter enters or leaves a node, Redirect, Receive, Transship, and Send, are all connected to the Interface place. A transporter entering a node may be received, redirected, or transshipped. A transporter that is received or transshipped will return to its home node if its current location is not its home. Also, the send action will result in a transporter leaving the node. Therefore, the arcs connecting the Interface place to the Receive, Redirect and Transship actions are bidirectional, whereas there is only an arc from Send to Interface, and not vice versa. These four actions that deal with transporters entering or leaving a node are connected by bidirectional arcs to the Network Topology place, to obtain the required information describing the network topology.

The Receive action may result in a transporter becoming idle (if the receiving node is its home node), hence the arc from Receive to Idle_Transporters. The Transship action will require the use of idle transporters and may result in a transporter becoming idle (if the transshipping node is the home node of the incoming transporter), hence the bidirectional arc between Transship and Idle_Transporters. The Send action requires idle transporters, hence the arc from Idle_Transporters to Send.

The Cargo_Store and Modules_Store places are connected to both the Send and Receive actions, as the former results in a reduction in the cargo and modules held at a node, while the latter results in an increase of the amount of modules or the amount of modules and cargo in a node. The Transship

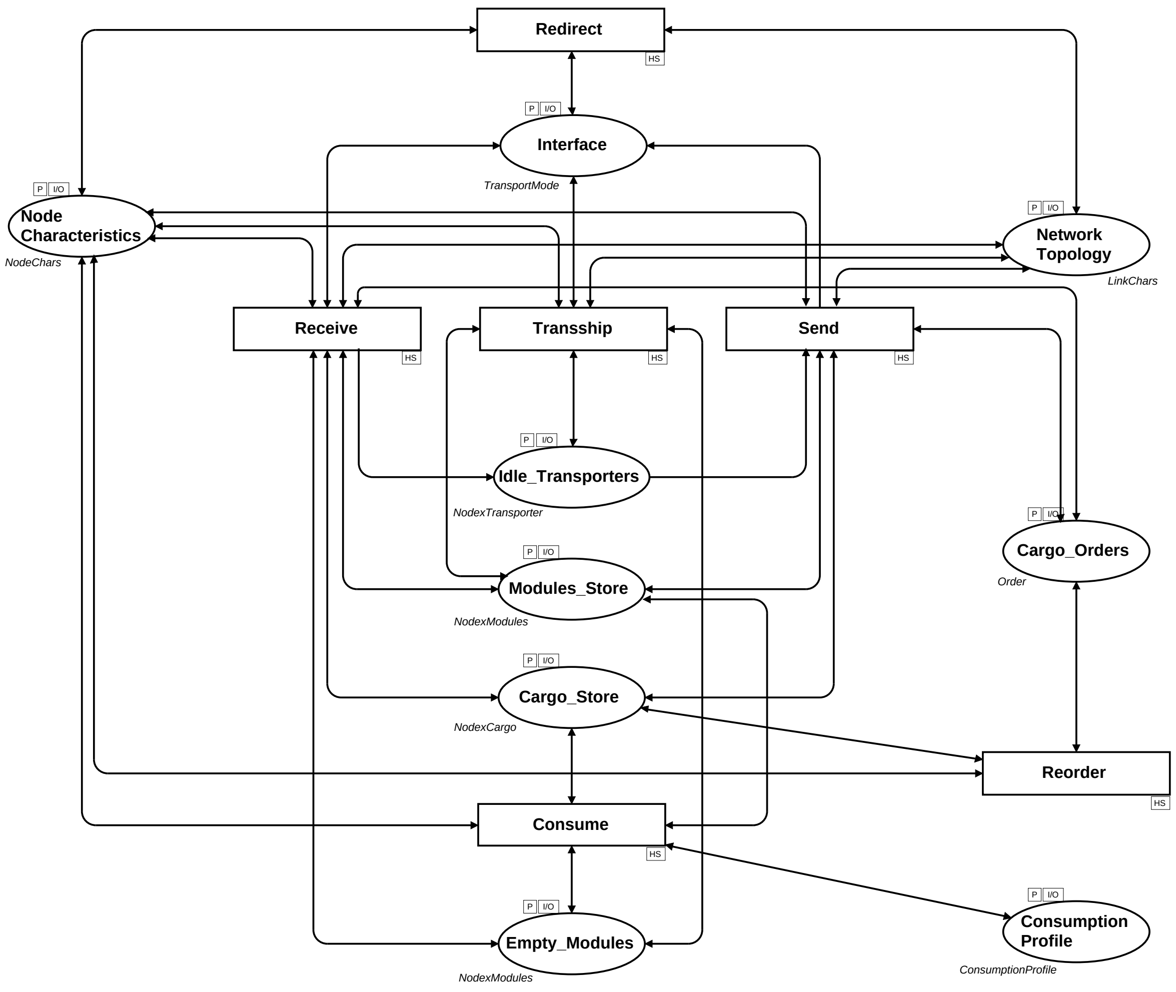


Fig. 9. Generic_node page.

action, however, is only connected to the Modules_Store place, as cargo for transshipping is modelled internally on the Tranship page. Receive and Tranship are connected to the Empty_Modules place, as empty modules may be loaded onto transporters returning to their home node from receive or transship actions.

Cargo consumption is modelled by Consume, resulting in additional empty modules being produced and a reduction in the amount of cargo and the number of modules packed with cargo in the node. The existing amount of cargo and number of modules must be known, hence there are bidirectional arcs between Consume and the Cargo_Store, Modules_Store and Empty_Modules places, to read these records and then update them.

The Reorder substitution transition needs to know the current cargo holdings of the node, hence the bidirectional arc between Reorder and Cargo_Store. A node's order token is updated, hence the bidirectional arc between Reorder and Cargo_Orders. Due to the order-driven nature of the modelled system, the Send and Receive actions also have bidirectional arcs to Cargo_Orders.

The Send action attempts to supply the required cargo to satisfy the order of a downstream node, decreasing the outstanding cargo and increasing the pending cargo of the downstream node. The Receive action reconciles orders and cargo, reducing the cargo pending (if the receiving node has cargo pending of the corresponding class(es)) and possibly increasing the cargo outstanding of the original destination (if the receiving node is not the original destination).

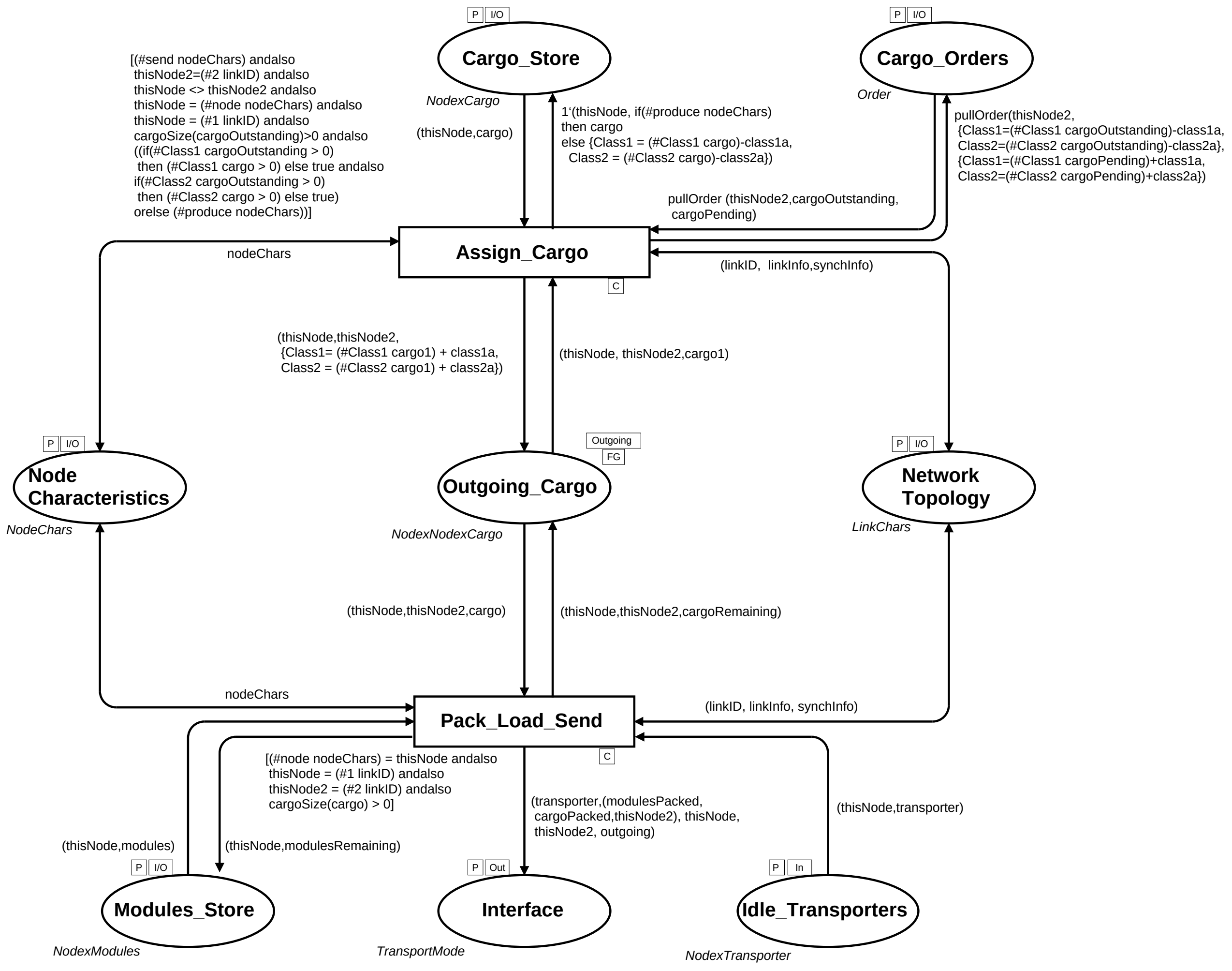

P I/O
Cargo_Store
NodexCargo
P I/O
Cargo_Orders
Order
[(#send nodeChars) andalso
thisNode2=(#2 linkID) andalso
thisNode <> thisNode2 andalso
thisNode = (#node nodeChars) andalso
thisNode = (#1 linkID) andalso
cargoSize(cargoOutstanding)>0 andalso
((if(#Class1 cargoOutstanding > 0)
then (#Class1 cargo > 0) else true andalso
if(#Class2 cargoOutstanding > 0)
then (#Class2 cargo > 0) else true)
orelse (#produce nodeChars))]
(thisNode,cargo)
1`(thisNode, if(#produce nodeChars)
then cargo
else {Class1 = (#Class1 cargo)-class1a,
Class2 = (#Class2 cargo)-class2a})
pullOrder(thisNode2,
{Class1=(#Class1 cargoOutstanding)-class1a,
Class2=(#Class2 cargoOutstanding)-class2a},
{Class1=(#Class1 cargoPending)+class1a,
Class2=(#Class2 cargoPending)+class2a})
pullOrder (thisNode2,cargoOutstanding,
cargoPending)
nodeChars
Assign_Cargo
C
(linkID, linkInfo,synchInfo)
(thisNode,thisNode2,
{Class1= (#Class1 cargo1) + class1a,
Class2 = (#Class2 cargo1) + class2a})
(thisNode, thisNode2,cargo1)
P I/O
Node
Characteristics
NodeChars
Outgoing
FG
Outgoing_Cargo
NodexNodexCargo
P I/O
Network
Topology
LinkChars
(thisNode,thisNode2,cargo)
(thisNode,thisNode2,cargoRemaining)
nodeChars
Pack_Load_Send
C
(linkID, linkInfo, synchInfo)
[(#node nodeChars) = thisNode andalso
thisNode = (#1 linkID) andalso
thisNode2 = (#2 linkID) andalso
cargoSize(cargo) > 0]
(transporter,(modulesPacked,
cargoPacked,thisNode2), thisNode,
thisNode2, outgoing)
(thisNode,transporter)
(thisNode,modules)
(thisNode,modulesRemaining)
P I/O
Modules_Store
NodexModules
P Out
Interface
TransportMode
P In
Idle_Transporters
NodexTransporter

Fig. 10. Send page.

Due to space limitations we shall only describe one page at the third level. This is the Send page, corresponding to the substitution transition of the same name on the Generic_node. Details of the other sub-pages can be found in [9].

Send Page The Send page, shown in Fig. 10, models the logistics function of sending cargo packed in modules on a transporter bound for a downstream node to fulfill an order from that downstream node. The send function is divided into two phases represented by the Assign_Cargo and Pack_Load_Send transitions.

The first (Assign_Cargo) sets aside cargo in response to an order. A code segment (not shown in Fig. 10) is used to calculate the amount of cargo to put aside for delivery to an immediate downstream node, based on the current stock levels and the outstanding cargo amount in the order from the corresponding downstream node. The assigned cargo is placed in Outgoing_Cargo and labelled with its intended destination.

The second (Pack_Load_Send) packs the previously set-aside cargo bound for one particular destination into modules and loads them onto a transporter. This also makes use of a code segment (again not shown) to calculate the maximum amount of cargo (and necessary modules for packing) that can be loaded onto a nondeterministically chosen idle transporter residing at the sending node. The code segment must take into account the weight and volumetric capacity of the transporter, the weight restrictions of the outgoing link, the policy of cargo segregtion (to keep classes of cargo segregated at the pallet level) and the packing procedure (two or more of any module must reside on or in a larger module, if it exists). The cargo and modules to be packed (and the cargo and modules left over) are calculated by incrementally adding boxes of cargo to the load of the transporter until one of the constraints is reached, or it runs out of cargo or modules. The motivation for using a code segment to calculate this amount of cargo is discussed in Section 4.4.

4 Discussion

During the course of the modelling work, we discovered a number of limitations of the modelling tools, Design/CPN and CPN Tools, as well as encountered difficulties with the use of CPNs as the modelling formalism for such a system. These difficulties, and their solutions, are discussed below.

4.1 Handling of Large Models

Early in the model's evolution, the topology of the logistics network was represented explicitly in the net structure, resulting in a large number of pages (around 30 to 35). We experienced difficulties in using the Design/CPN editor. The tool would sometimes not allow us to add, edit or remove net elements (places and transitions). Sometimes it would not save nets correctly and would crash when loading them. This problem disappeared after reducing the number of pages in the model by representing the topology in tokens.

4.2 Extensible Data Structures

Significant effort went into designing data structures for the model. For example, for cargo, we considered a list, a product and a record. Lists have the advantage of extensibility, in that the number of elements in the list (the list length) is not hard-coded into the data structure itself. Thus, in the future, the number of classes of cargo can be extended without changes to the corresponding colour set definition. However, lists suffer from a significant disadvantage, namely the additional programming effort required to ensure the model maintains a coherent and meaningful list structure and ordering.

Products and record structures to not suffer from this disadvantage, as the structure is implicit in the colour set itself and no additional effort is needed to maintain this structure. Although products and records have essentially the same underlying data structure, records provide more information to the user regarding exactly what element is being accessed, through explicit field labels. For these reasons, records were chosen. The downside of using records is that arc inscriptions become very large, as each field of the record must be explicitly represented in inscriptions if a new token of this record type is to be created, *even* if values from an existing token are being reused. An example of this can be seen on the arc from the Assign_Cargo transition to the Cargo_Orders place in Fig. 10. Another disadvantage of both products and records is that they lack the extensible nature of lists, due to their statically defined structure, thus any addition to the number of cargo classes requires modification of colour sets, all related arc inscriptions, guards and code segments.

4.3 Excessive Nondeterministic Choice: Binding Element Explosion

An interesting problem that we faced was that of *binding element explosion*. One of the strengths of the CPN formalism is its ability to handle concurrency and nondeterminism. When attempting to model a logistics network with very abstract control mechanisms, our philosophy was to model nondeterministically as many choices as possible, and so not 'hard wire' aspects of control into the model, in order to capture all possibilities of physical actions. The explosion of binding elements became a problem, most notably with the sending, ordering and consumption actions, as it became evident that the Design/CPN simulator cannot handle a large number of binding elements.

As an example, consider 10 boxes of each of 9 classes of cargo (90 boxes in total). When choosing nondeterministically a number of boxes (from 0 to 10) of each class to consume, this gives 11 choices for each of the 9 classes, i.e. 11^9 possibilities (over 2.3 billion). The syntax check within Design/CPN was successful and the simulator tool could be entered. However, attempts at both automatic and manual simulation caused the tool to crash when attempting to calculate such large numbers of enabled bindings for the Order and Consume transitions. When attempts were made to bind the values of variables manually, the tool would also crash when attempting to display all choices to the user.

After converting the model to CPN Tools format and opening it in CPN Tools, it was discovered that CPN Tools also could not handle this situation. We believe that the reason is caused by the merging of the editor and simulator in CPN Tools. CPN Tools performs syntax checking on-the-fly, thus allowing the complete parts of a partially complete model to be simulated. In our case, however, the excessive nondeterminism manifested itself as an excessively long syntax check that did not terminate (it was terminated manually after 40 minutes). The syntax check never advanced to the stage where even part of the CPN could be simulated.

This prompted us to seek a reduction in the degree of nondeterminism when choosing cargo to send, order or consume. Rather than allowing any amount of cargo to be sent from one node to another, we enforced a specific 'packing policy' by implementing an algorithm that determines the amount and configuration of cargo to be packed into a transporter and sent in the Pack_Load_Send action. This is discussed in more detail in Section 4.4 below.

The excessive nondeterminism introduced by the ordering mechanism, which allowed any node to order any amount of cargo, was curtailed by the introduction of the pull order mechanism. Each node was then able to determine specific amounts of cargo to order. The nondeterminism in cargo consumption was alleviated by the introduction of *consumption profiles*, which dictate both what is to be consumed by each node and the ordering of consumption events over the consuming nodes.

4.4 The Use of Code Segments for Implementing Nondeterministic Constraints

Arc inscriptions and guards allow the specification of constraints that restrict the set of binding elements enabled by a particular marking. These constraints are generally known and set statically while creating the model, and a given binding of variables will either satisfy the constraints, or will not satisfy the constraints. The term *nondeterministic constraints* refers to the situation in which the constraints themselves are not explicitly known prior to run-time, and are determined by the values of the variables themselves in a given binding element.

Specifically, difficulties were encountered when attempting to load a transporter with cargo and modules via the Pack_Load_Send operation. Our goal was to load as much cargo as possible (up to the amount of cargo set aside for one destination) while still conforming to the weight restrictions of the outgoing link and the weight and volumetric capacity constraints of the transporter itself. Restricting the amount of cargo to satisfy the weight and capacity constraints listed above is not difficult and can be achieved using conventional guard expressions. However, *any* binding element that satisfies these constraints is enabled. To force the amount of cargo to be 'as large as possible' within these constraints was a more difficult task, and one that we could not achieve with guards and arc inscriptions alone.

This prompted the use of *code segments* to bind the cargoPacked and modulesPacked variables (and the complementary cargoRemaining and modulesRemaining variables), based on the cargo available for sending, the transporter chosen, and the outgoing link. Due care must be taken when using code segments to bind the values of variables so as not to result in illegal token colours. The method chosen to calculate the values of the variables is as described under the Send page description at the end of Section 3.3, where the cargo (and required modules) is incrementally increased until one of the constraints is reached. This guarantees that the cargo and modules sent never exceeds the amount of available cargo and modules, the weight restrictions of the outgoing link, or the weight and volumetric constraints of the transporter.

5 Conclusions

This paper presents a Coloured Petri Net model of a Defence Logistics Network capturing the principal operations of a pull-based physical network, and some of our experiences in carrying out the modelling activity. The modelling presented a significant challenge, due to the size and complexity of the domain. This paper describes the major physical components of a logistics network, the intuition behind the model dynamics by means of a simple case study, and the detailed implementation of the operations related to supplies.

The model has nodes, which represent military bases or staging points, and links between nodes as its major components. We discuss the scope of the model and some important assumptions, including the abstraction of the information network and the control mechanisms within the logistics network model. More details, including a description of the model evolution, can be found in [9]. Care was taken to build an accurate and precise model reflecting a real-world logistics network, to ensure the extensibility and modularity of the model, and to provide some measure of future-proofing of the model. An important decision was to encode the network topology within CPN tokens rather than in the net structure itself, to allow the model to be both flexible and adaptable to a range of network implementations. Without such an encoding, any change to the topology would require a change to the net structure, whereas currently this only requires a change in the pseudo-initial marking. It also makes the CPN model much more compact and manageable, at the expense of losing a one-to-one mapping of net elements to physical entities, and hence reducing the readability and visualisation of data flow.

Our modelling work has also revealed a number of limitations of Design/CPN and CPN Tools. Most of these relate to the operation of the simulator (in Design/CPN) and incremental syntax checking and the state space tool in CPN Tools. This paper discusses the problems we encountered and our approach to overcoming them, while still achieving our original objectives.

Two key future work activities are immediately forthcoming from the work presented in this paper. The first is a continuation of the modelling of the functional behaviour of the logistics network. Many simplifications and assumptions were made to obtain the model in the time available. Thus there is considerable scope for incremental relaxation of these assumptions, and for extensions and refinements to the model. This also includes the ongoing incorporation of feedback and suggestions from domain experts in the defence logistics area. Potential future modelling tasks include: the modelling of a realistic and imperfect information network; implementation of a mechanism to allow different control schemes to be imposed on the system; a more extensible implementation of cargo classes; more complete modelling of node, transporter and link characteristics; incorporation of transporter failures; more advanced routing schemes for transporters; more advanced delivery schemes, including *milk-runs* (where a transporter delivers to multiple nodes in a single excursion) and aggregation of orders from multiple downstream levels; the ability to represent more complex network topologies including cyclic networks; and a dynamically changing network topology. Importantly, incorporation of time into the model would allow some quantitative results to be obtained.

Given our solutions for alleviating binding element explosion, our model runs successfully under both Design/CPN and CPN Tools. Thus the second key future work activity is the formal analysis of such a model. This task can be divided into two parts. The first is functional analysis, which may consist of generation and analysis of the full or partial reachability graphs of the logistics CPN model. The second is performance analysis. This will require the identification and formulation of a number of measures of performance that give quantifiable results to the measures of performance that are of interest to DSTO. Such measures of performance may include module utilisation, average stock levels, or the time spent below the minimum and/or reorder thresholds for each class of supply. Multiple simulation runs of the CPN model incorporating time, could then be undertaken to gauge and compare the relative performance of different Logistics Network configurations.

Acknowledgments

The authors would like to acknowledge Greg Allen, Michael Burgess, Bruce Vandepeer and MAJ Rob Webb for their valuable input and support throughout the course of this project.

References

1. W. M. P. van der Aalst. Timed Coloured Petri Nets and their Application to Logistics. PhD Thesis, Eindhoven University of Technology, 1992.
2. *Australian Defence Doctrine Publication 4.0 - Defence Logistics.* Defence Publishing Service, Department of Defence, 8 April 2003.
3. J. Billington, M. Diaz, and G. Rozenberg, editors. *Application of Petri Nets to Communication Networks*, volume 1605 of *Lecture Notes in Computer Science.* Springer-Verlag, 1999.
4. H. J. Bullinger and J. V. Steinaecker. Petri Net Based Modelling, Planning and Control of Logistical Processes under Environmental Goals and Constraints. In *Proc. of 9th Symposium on Information Control in Manufacturing*, volume 1, pages 433–438. Elsevier Science, June 1998.
5. CPN Tools. `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`.
6. C. Degano and A. Di Febbraro. On using Petri Nets to Detect and Recover from Faulty Behaviours in Transportation Facilities. In *Proc. of the 2002 IEEE Int. Conf. on Systems, Man and Cybernetics*, volume 1, pages 31–36. IEEE, Oct 2002.
7. Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.
8. I. Eick, D. Vogelsang, and A. Behrens. Planning Smelter Logistics: A Process Modeling Approach. In *Light Metals 2001, New Orleans, LA*, pages 393–398. Minerals, Metals and Materials Society, Feb 2001.
9. G.E. Gallasch, N. Lilith, and J. Billington. A Coloured Petri Net Model of a Defence Logistics Physical Network. Technical Report CSEC-25, Computer Systems Engineering Centre Report Series, University of South Australia, August 2006.
10. N. E. Hutchinson. *An Integrated Approach to Logistics Management.* Prentice-Hall, 1987.
11. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts.* Monographs in Theoretical Computer Science. Springer-Verlag, 2nd edition, 1997.
12. 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.
13. L.M. Kristensen, B. Mitchell, L. Zhang, and J. Billington. Modelling and Initial Analysis of Operational Planning Processes using Coloured Petri Nets. In *Formal Methods in Software Engineering and Defence Systems 2002, Proceedings of the* Satellite Workshops on Software Engineering and Formal Methods *and* Formal Methods Applied to Defence Systems, volume 12 of *Conferences in Research and Practice in Information Technology Series*, pages 105–114. Australian Computer Society Inc., 2002.
14. B. Mitchell, L. M. Kristensen, and L. Zhang. Formal Specification and State Space Analysis of an Operational Planning Process. In *Proceedings of the 5th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools (CPN'04), Aarhus, Denmark.*, pages 1–18, 2004.
15. E. Ochmanska. Object-oriented PN Models with Storage for Transport and Logistic Processes. In *Proc. of the 9th European Simulation and Symposium, Passau, Germany*, pages 483–487, San Diego, CA, USA, Oct 1997. SCS.
16. M. A. Piera, M. Narciso, A. Guasch, and D. Riera. Optimization of Logistic and Manufacturing Systems through Simulation: a Coloured Petri net-based methodology. *Simulation*, 80(3):121–129, 2004.
17. The Technical Cooperation Program (TCCP). *Guide for Understanding and Implementing Defence Experimentation (Guidex).* Version 1.1, February, 2006.
18. US Department of Defense. *Logistics Transformation Strategy - Achieving Knowledge-Enabled Logistics.* 10th December, 2006.
19. J. G. A. J. van der Vorst, A. J. M. Beulens, and P. van Beek. Modelling and Simulating Multi-echelon Food Systems. *European Journal of Operational Research*, 122(2):354–366, 2000.
20. N. Viswanadham and N. R. S. Raghavan. Performance Analysis and Design of Supply Chains: A Petri Net Approach. *Journal of the Operational Research Society*, 51(10):1158–1169, 2000.
21. M. von Mevius and R. Pibernik. Process Management in Supply Chains – A New Petri-net Based Approach. In *Prof. of the 37th Annual Hawaii Int. Conf. on System Sciences*, volume 37, pages 1153–1162. IEEE Computer Society, Jan 2004.
22. Z. Wang, J. Zhang, and F. T. S Chan. A Hybrid Petri Nets Model of Networked Manufacturing Systems and its Control System Architecture. *Journal of Manufacturing Technology Management*, 16(1):36–52, 2005.
23. Websters dictionary.
24. L. Zhang, L.M. Kristensen, C. Janczura, G.E. Gallasch, and J. Billington. A Coloured Petri Net based Tool for Course of Action Development and Analysis. In *Formal Methods in Software Engineering and Defence Systems 2002, Proceedings of the* Satellite Workshops on Software Engineering and Formal Methods *and* Formal Methods Applied to Defence Systems, volume 12 of *Conferences in Research and Practice in Information Technology Series*, pages 125–134. Australian Computer Society Inc., 2002.
25. L. Zhang, L.M. Kristensen, B. Mitchell, G.E. Gallasch, P. Mechlenborg, and C. Janczura. COAST - An Operational Planning Tool for Course of Action Development and Analysis. In *Proceedings of the 9th International Command and Control Research and Technology Symposium (ICCRTS)*, Copenhagen, Denmark, 2004.

Protos2CPN: Using Colored Petri Nets for Configuring and Testing Business Processes

F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and H.M.W. Verbeek

Department of Technology Management, Eindhoven University of Technology,
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands.
f.gottschalk@tm.tue.nl

Abstract. Protos is a popular tool for business process modelling used in more than 1500 organizations. It has a built-in Petri-net-based simulation engine which shows key performance indicators for the modelled processes. Reference process models offered for Protos reduce modelling efforts by providing generic solutions which only need to be adapted to individual requirements. However, the user can neither inspect or interact with simulations running in Protos, nor does Protos provide any explicit support for the adaptation of reference models. Hence, we aim at a more open and configurable simulation solution. To realize this we provide two transformations from Protos models to colored Petri nets (CPNs), which can be executed by CPN Tools. The first transformation enables the usage of the extensive simulation and measuring features of CPN Tools for the simulation of Protos models. The second transformation creates colored Petri nets with dedicated features for process configuration. Such configurable process models can be restricted directly within the process model without changing the model's structure and provide therefore dedicated adaptation features for Protos' reference process models.

1 Introduction

Today "process thinking" has become a mainstream organizational practice [20]. Business process models provide a graphical and systematic view on organizational processes [15]. Various tools for business process modelling have been developed since the late nineties [2]. One popular tool is Protos from the company "Pallas Athena". Currently it is used by about 1500 organizations in more than 20 countries. E.g., more than half of all municipalities within the Netherlands use Protos for the specification of their in-house business processes [24].

Most providers of modelling tools, and, e.g., also all dominant enterprise system vendors provide reference models with or for their software. Reference models are supposed to reduce the modelling efforts by providing generic solutions that just need to be adapted to individual requirements [5,9,10,11,18,19]. Pallas Athena provides several sets of reference process models implemented in Protos. As an example, there is a set of about 60 reference process models for municipalities. These are ordinary Protos models depicting common processes. The municipality or organization buying such a reference model can adapt the models to its individual requirements, avoiding the huge effort of building process models from scratch. However, it is quite important to note that neither Protos nor any other popular process modelling tool provides any explicit support for the adaptation of reference models [16].

Figure 1 depicts a reference process model for the handling of objections against parking tickets[1]. If an objection is received within the corresponding deadline it is checked for its

[1] The model is motivated by one of Pallas Athena's reference models, but specially build for the purpose of this paper.

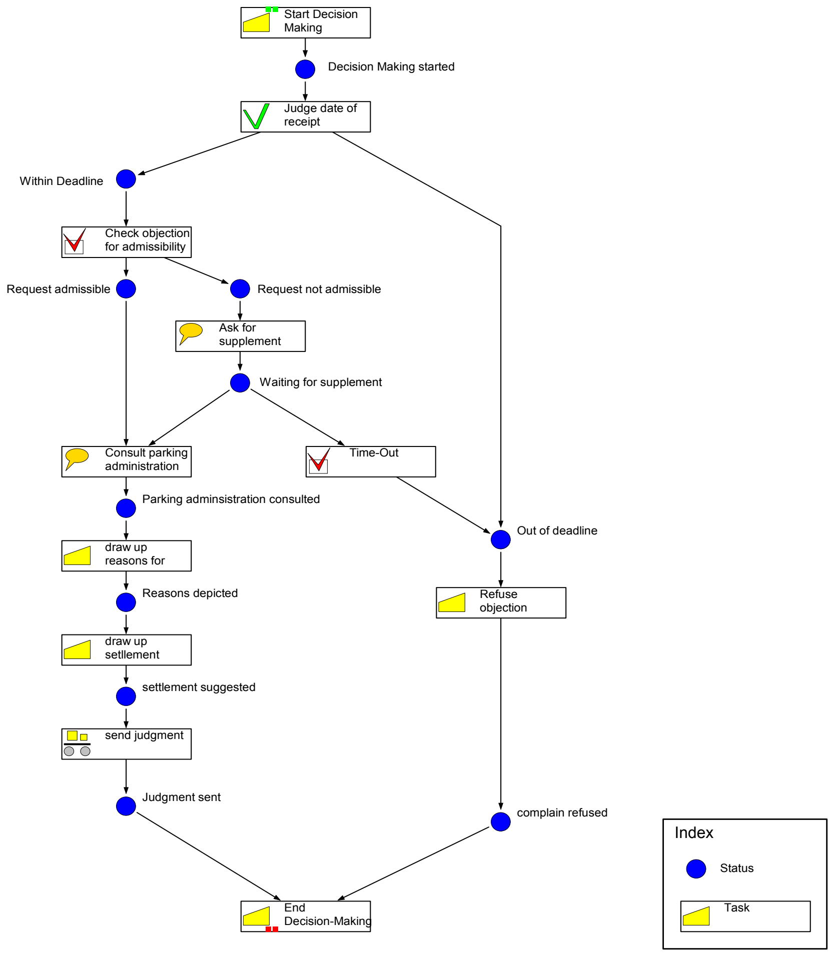


Fig. 1. An example decision-making process about objections against parking fines, modelled in Protos

admissibility. In case it is not admissible, a supplement is requested. If this is received within the new deadline or if a supplement was not needed, the parking administration is consulted, reasons for approval/refusal as well as a settlement are drawn up, and the judgement is sent to the objecting citizen. Otherwise the objection times out and is refused directly as it is in case the objection was not received within the deadline.

During this research a set of more than 500 reference and adapted process models was made available to us by Pallas Athena. Although guidelines how to model sound business processes in Protos and tools for verification exist [21,22,23], we discovered that most of these process models do not conform to the guidelines. In addition, the models lack of data required for process simulations which also means that simulation [24] was hardly used, if at all. So we can conclude that the designers were either unaware or not convinced of the value of sound models and simulation. One main reason for this is that it is unclear to designers which of the parameters that can be specified in Protos are actually used for the simulation. For example, we discovered that in Protos a field for the number of resources required for the execution of a task exists, but the simulation always uses just one resource and neglects this parameter.

Within this paper we will present two new tools helping Protos users to test and validate their process models and therefore to improve the quality. Both tools are available for download from http://www.floriangottschalk.de/protos2cpn.

First, we will depict a new way to simulate business processes modelled in Protos using CPN Tools [25]. Nowadays CPN Tools is probably the most popular modelling and analysis environment for colored Petri nets (CPNs) [14], used in more than 115 countries with more than 3000 licensees. It provides not only a nice way to visualize running processes but also extensive measurement and verification opportunities for concurrent systems. Within this research we developed a transformation from Protos models to CPNs, using the same data as the current Protos simulation. Using the simulation of CPN Tools we enable the unexperienced user to see directly in which order the process tasks are executed and what might go wrong in incorrect workflow models. In additon some basic statistics are provided to him. The advanced user will be able to add additional measurements to process models as well as he can see which of the Protos parameters are actually used during the simulation. The current Protos simulation is using a tool called ExSpect [4,24] which is based on another type of colored Petri nets. However, ExSpect does not allow for the easy creation of additional measurements. Its standard layout scheme, which is applied when loading a model, causes unacceptable delays when trying to inspect or interact with running processes. In addition, the development of ExSpect has stopped for some time already [12].

Second, we change the transformation in such a way that it creates a configurable process model from the Protos model. We developed configurable process models in our previous research as a general mechanism for process model adaptation [5,13]. When configuring a process, its unnecessary parts are eliminated, i.e. the possible process behavior is restricted [9,10,18,19]. Incorporating configuration options into the process model during the transformation creates for the first time a tool allowing to make process configuration decisions within a process model and to evaluate the configuration decision by direct interactions with the simulation model.

The paper concludes with a summary and an outlook on open issues.

2 Protos2CPN: From Protos Models to Colored Petri Nets

Basically Protos2CPN converts the data provided by Protos for the current simulation into a CPN, executable in CPN Tools. So far, CPN Tools provides no opportunities for importing other file formats than the CPN Tools XML-file format. It is therefore reasonable to use an XML export of Protos and transform this into the CPN Tools format by an XSL transformation. The current Protos simulation [24] is already using temporary stored XML files for the communication between Protos and ExSpect. So we decided to "plug-in" in between using the same XML export for the generation of the CPN Tools files, especially as our main goal is the process simulation as well. The exported XML file includes a flattened process structure (i.e, the hierarchical structure is reduced to a single level process model), the resource utilization of each task, and further statistical simulation parameters. It lacks of information how to layout the process, but we aim at providing an automatic layout functionality. Currently, the models are created with a basic layout scheme that allows for an easy re-arranging of process elements, keeping the manual effort reasonable.

In order to enable the user to look at the CPN in the same way as to the Protos model, without the need for learning a new "complicated" modelling language [17], it is our goal to transform the Protos models into CPN models that match the Protos model in look and structure as close as possible. For that reason we decided that any information not depicted in the process view of Protos (cf. Figure 1) but maintained in property dialogues and needed for simulation must be depicted on separate sub-pages. The derived CPN model provides

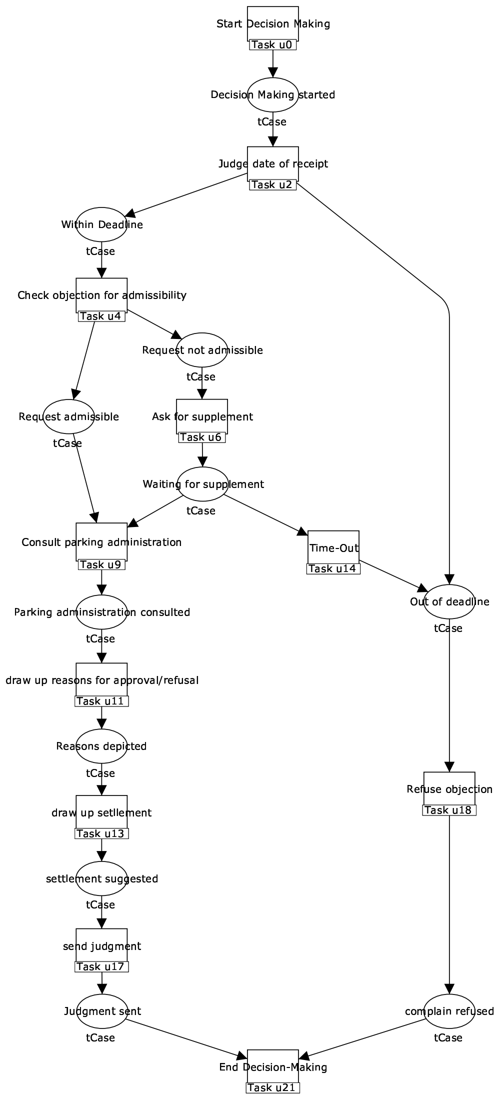


Fig. 2. The overall process view of the CPN (generated from the Protos process model depicted in Figure 1)

therefore two levels: First the overall process view, similar to the (flattened) process view of Protos, where every Protos status[2] is transformed into a CPN place[3] and each Protos task is transformed to a CPN substitution transition (cf. figures 1 and 2); and second the sub-pages of these substitution transitions representing an "execution layer" incorporating all data relevant for the simulation (cf. Figure 3).

[2] The Protos name for a place (e.g., a model object representing a channel or state) is status.

[3] Note, that in Protos it is possible to connect transitions directly to transitions or statuses directly to statuses whereas in (colored) Petri nets this is not. In this case additional auxiliary statuses or transitions are introduced into the CPN process overview model. This is already done by Protos when creating the XML export.

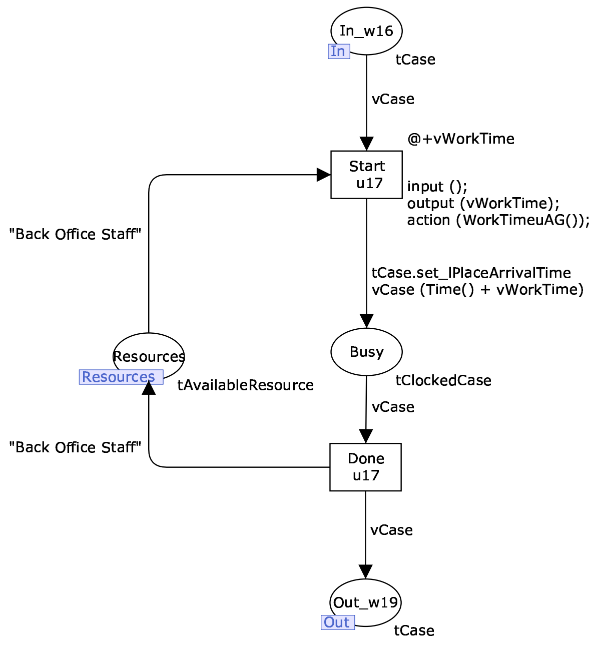


Fig. 3. Dedicated task page with information for simulation

The places in the overall model are allowed to have tokens of the type `tCase`[4] depicting details about the cases running through the process, e.g., a case id, the start time of the process, and the arrival time of the process in the particular place:

```
colset tCase = record lCaseID:INT *
                      lProcessArrivalTime:INT *
                      lRank:INT *
                      lPlaceArrivalTime:INT *
                      lCost:INT;
```

Whenever a token is residing in one of the places in the overall model it is waiting for execution by one of the subsequent transitions. All specifications of how and when a task can be performed by one of these transitions are "hidden" on the corresponding sub-page for the task, i.e. the transition depicted in the overall model is just a substitution transition for the underlying sub-page.

Each of these sub-pages consists basically of two transitions: A `Start` and a `Done` transition (cf. Figure 3). The `Start` transition symbolizes the start of the task and is enabled by token(s) on its input place(s), i.e. by tokens arriving on the preceding places in the overall process model. When the `Start` transition fires, it puts a token into a place `Busy`, symbolizing the lasting of the task. By adding a delay of `vWorkTime` to the token it is ensured that the token cannot be removed from the `Busy`-place while the task lasts. The duration of the task, i.e. the delay, is determined by the function `WorkTimeuXX()` which calculates the duration of the task according to the specification at the corresponding task in the Protos model (XX is replaced with a two letter code in the model, which is a shortcut for the name of the particular task). Firing the `Done` transition depicts the completion of the task. It removes the token from the

[4] In order to distinguish between colorsets, labels and variables in the CPNs, all colorset names start with a lowercase t (type), all labels with a lowercase l, and all variables with a lowercase v.

`Busy` place, and puts token(s) into the output place(s), i.e. in the succeeding place(s) within the overall model.

The number of input- and output places depends on the number of incoming and outgoing arcs of the particular task within the Protos model. In Protos it can be specified if several incoming arcs depict either an AND-Join or an XOR-Join. Several outgoing arcs of a task can depict either an AND-split or an XOR-split. This might be confusing for Petri net users as the substitution transitions in the overall process model are looking like a standard Petri net AND-join/-split, but can represent both XOR and AND-joins/-splits. The exact behavior is only modelled within the corresponding CPN sub-page:

– In case of an AND-join each input place is connected to the `Start` transition. The case id's of incoming tokens from different arcs are synchronized by a guard attached to the `Start` transition. (see Figure 4).
– In case of an XOR-join, a `Start` transition is introduced for each input-place/arc. So a `Start` transition can fire and result in the busy state of the task whenever a token is placed into one of the input places (see Figure 5).
– In case of an AND-split the `Done` transition puts tokens into all output places (see Figure 6).
– In case of an XOR-split, a `Done` transition is introduced for each output place/arc (Figure 7). So only one `Done` transition can take the token out of the `Busy` place and fire by removing the token from the place `Busy`, releasing the resource and putting the case token into the corresponding `Out` place. The other `Done` transitions become disabled as soon as the first one fires as no token remains in the `Busy` place..

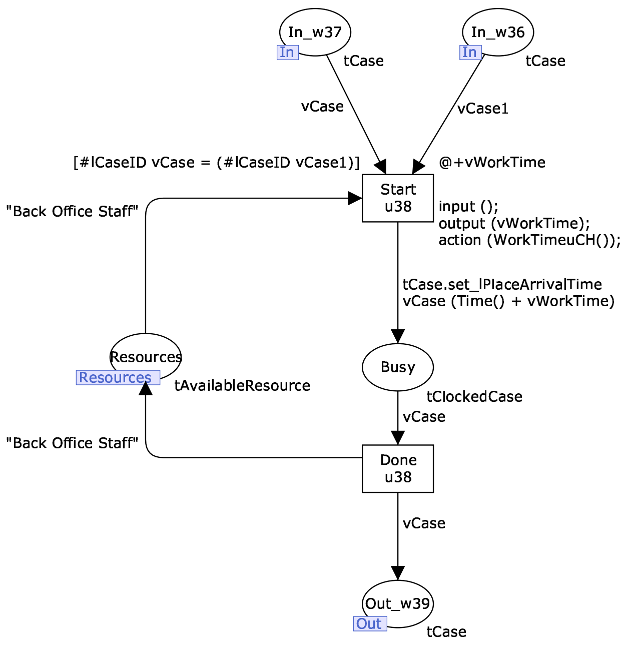


Fig. 4. The AND-Join modelled using a CPN sub-page: The transition `Start u38` needs a token with the same case id in both input places (`In_w36` and `In_w37`)

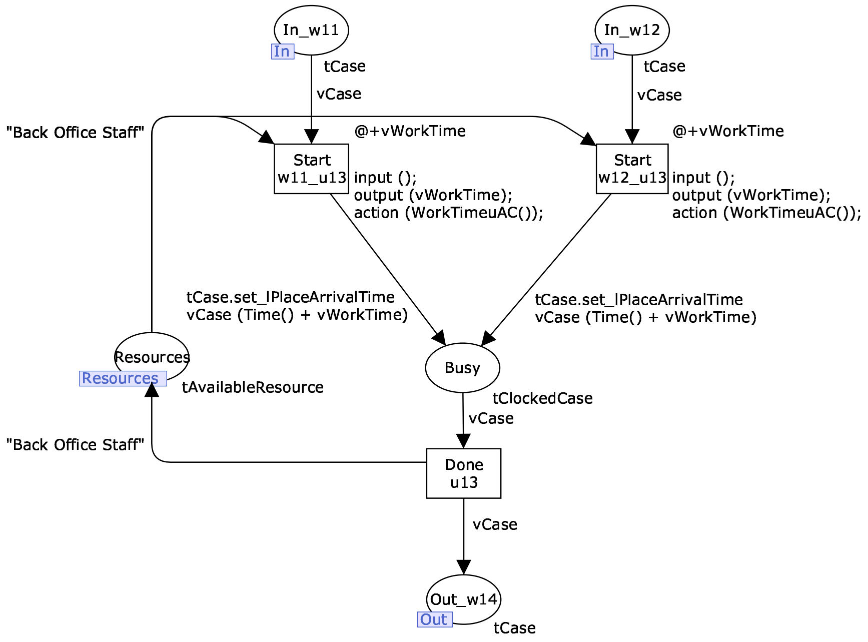


Fig. 5. XOR-Join: Each `Start` transition is enabled as soon as a token arrives in the corresponding `In` place and the resource "Back Office Staff" is available.

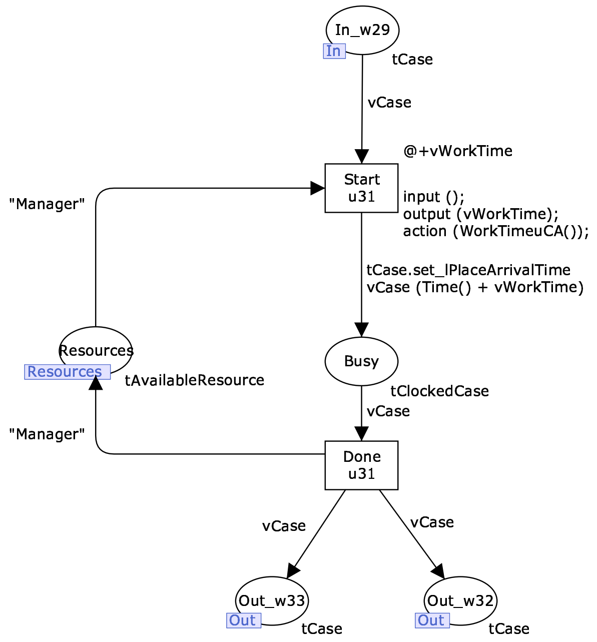


Fig. 6. AND-Split: The transition `Done u13` releases the resource "Back Office Staff" and forwards the case to both outgoing arcs by putting a case token into each of the two `Out` places.

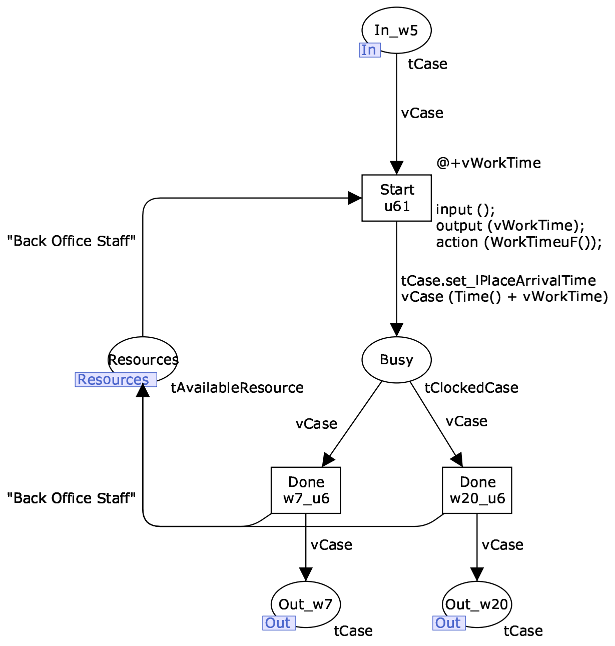


Fig. 7. XOR-Split: As soon as the case is not "busy" anymore, both `Done` transitions become enabled, but only one can fire.

In addition to the control flow behavior also resource utilization is depicted in the sub-pages. For that reason every sub-page contains a place `Resources`. All these resource places are members of a fusion set containing the available resources:

```
colset tAvailableResources = STRING;
```

Transition `Start` can only fire if the required resources are available. The resource requirements for a task are specified on the arc from the place `Resources` to the transition `Start`. When the transition `Done` is fired, depicting the completion of the task, the previously occupied resources are released, i.e. the resources removed by the `Start` transition are put back into the `Resources` place. They can afterwards be used for other tasks.

Besides the overall process model and the sub-pages for the tasks, a page `Arrival System` is created. It consists of a single transition spawning cases into the input place of the first Protos task based on some predefined arrival process, e.g. a Poisson arrival process using negative exponential inter arrival times. In this way it is ensured that the simulation is continuously feeded with new cases.

Altogether the CPN model enables a step-by-step simulation of the Protos model, allowing a detailed analysis of diverse process behavior. In addition the monitoring-features of CPN Tools enable complex data collection while simulating processes. When transforming Protos models to CPN, Protos2CPN generates automatically three types of data collector monitors:

- A data collector monitor measuring the total flow time through the process per case,
- A data collector monitor measuring the resource availability/resource utilization over time per resource type, and
- A data collector monitor measuring the waiting time of cases per task (i.e. the time cases are waiting for execution by the particular task solely due to the lack of resources).

Normally the simulation would run forever as cases are continuously spawned into the system. To stop the simulation a breakpoint monitor is defined on the arrival system. By default it stops the simulation after 500 cases have been spawned into the system. This value can of course be adapted to individual requirements. A function allowing the performance of several replications of the simulation (by default four replications) is located on an additional `SimulationStart` page.

Analyzing the three generated statistics can be used to find bottlenecks or other performance related problems. When analyzing the data, keep in mind that it might include a warm up period before reaching a steady state. Additional, more complex monitors can be added by the advanced user in CPN Tools. Users interested in details are referred to the monitoring help pages of CPN Tools [26].

3 Protos2C-CPN: Using CPN for Building Configurable Process Models

The second tool we developed is an extended variant of Protos2CPN allowing process model configuration within the generated model. For that reason we called it Protos2C-CPN where C-CPN stands for "configurable colored Petri net". When configuring a process model, e.g. the process model depicted in Figure 1, some of the tasks of the model are eliminated in such a way that they cannot be performed when the process is enacted. CPN models created by Protos2C-CPN provide dedicated features for process model configuration. These features enable the user to adapt the model to individual requirements without changing the structure of the original reference model. Afterwards the configured model can either be tested on its feasibility, i.e. its soundness, or it can be simulated in the same way as depicted in the previous section. By applying and simulating different configurations on the same process model their particular efficiency could even be compared.

For depicting how Protos2C-CPN can be used in this context and also to show the limitations of Protos2C-CPN, this section is split into three parts. First we will give some background information on the ideas behind configuration of process models. Second we will explain how these ideas are incorporated into the CPN models. And third we will conclude this section with an analysis of the limitations of the configuration approach introduced in this paper.

3.1 Configuring Process Models

Configuration is a mechanism for adapting process models that restricts the possible runtime behavior of a process [8,9]. As an example, removing an arbitrary task from the process model in figures 1 and 2 would be configuring of the process model. However, according to our definition of configuration, operations such as the adding of additional tasks or the renaming of model elements are not possible by means of configuration. Also note that not all configurations are sound/valid.

Based on concepts adopted from the inheritance of dynamic behavior [3,7], we identified two mechanisms for configuration of process models in previous research, called *blocking* and *hiding* [5,13]. Blocking refers to encapsulation. If a task in a process is blocked, it cannot be executed. The process will never reach a subsequent state and therefore all (potential) subsequent tasks will not be executed as well. If, e.g., the task `Time-Out` in Figure 1 (or Figure 2) is blocked the process will never reach the status `Out of deadline` nor execute the task `Refuse objection` after it has been in the place `Waiting for supplement`. Hiding corresponds to abstraction. If a task is hidden, its performance is not observable, i.e. it consumes neither time nor resources when it is executed. But the process flow continues afterwards and subsequent tasks will be performed. For that reason we also talk about a *silent task* or simply about *skipping the task*. If, e.g., the task `Draw up reasons for approval/refusal` in figures 1 and 2 is hidden, the task `draw up settlement` will be the next task after the parking administration was consulted. So, whenever a certain task should not be executed, but the process flow should be able to continue in this direction, the task must be hidden. If the process flow should not continue in this direction, the task must be blocked.

Configuration decisions about blocking or hiding of tasks are typically made before the execution of the process. However, sometimes not all information required to decide between activating, blocking, or hiding might or needs to be available in advance. For example, the configuration decision might be that the task `Draw up reasons for approval/refusal` can be skipped for certain specific cases whereas for others it cannot. Then the decision must be postponed to the run-time of the process and made on a case-by-case basis when the process is executed.

3.2 Configurable CPN

To cope with the two mechanisms for configuration, blocking and hiding, the models derived in Section 2 have to be extended with additional behavior. As process configuration is defined on a task level this can be done on the sub-page of each task. A task is activated if it is neither blocked nor hidden. This corresponds to the situation in ordinary, i.e. non-configurable, models. Within the CPNs generated by Protos2C-CPN these three configuration opportunities are distinguished by using the type `tConfigDecision`:

```
colset tConfigDecision = with Activated | Hidden | Blocked;
```

The decision between activating, hiding, and blocking has to be made for each task individually. For that reason a place `Configuration` is added to each sub-page as depicted in the top-right of Figure 8. When configuring the task the default decision, i.e. the initial marking of the place `Configuration`, can be changed from `Activated` to `Hidden` or `Blocked`.

Whenever a token arrives in an input place of a task, the transition `AddConfiguration` is enabled. When firing, this transition takes a configuration from the `Configuration` place (`vDecision`), combines it with the number of the particular task ("u17" in the example in Figure 8), and adds it to the list of configurations (color `tTaskConfigurations`) made for the corresponding case (function `AddConfiguration(vCase: tCase, vTask: STRING, vDecision: tConfigDecision)`. This list is a further attribute to the color set `tCase`:

```
colset tTaskConfiguration = record lTask:STRING *
                                   lConfiguration:tConfigDecision;
colset tTaskConfigurations = list tTaskConfiguration;
colset tCase = record lCaseID:INT *
                      lProcessArrivalTime:INT *
                      lRank:INT *
                      lPlaceArrivalTime:INT *
                      lCost:INT *
                      lConfiguration:tTaskConfigurations;
```

If the configuration decision is not clear during the phase of process configuration, tokens for all possible configuration decisions can be put into the `Configuration` place at the same time. The decision will then be made at runtime when the transition `Add Configuration` "selects" a configuration token. The function `notConfigured(vTask: STRING, vCase: tCase)`

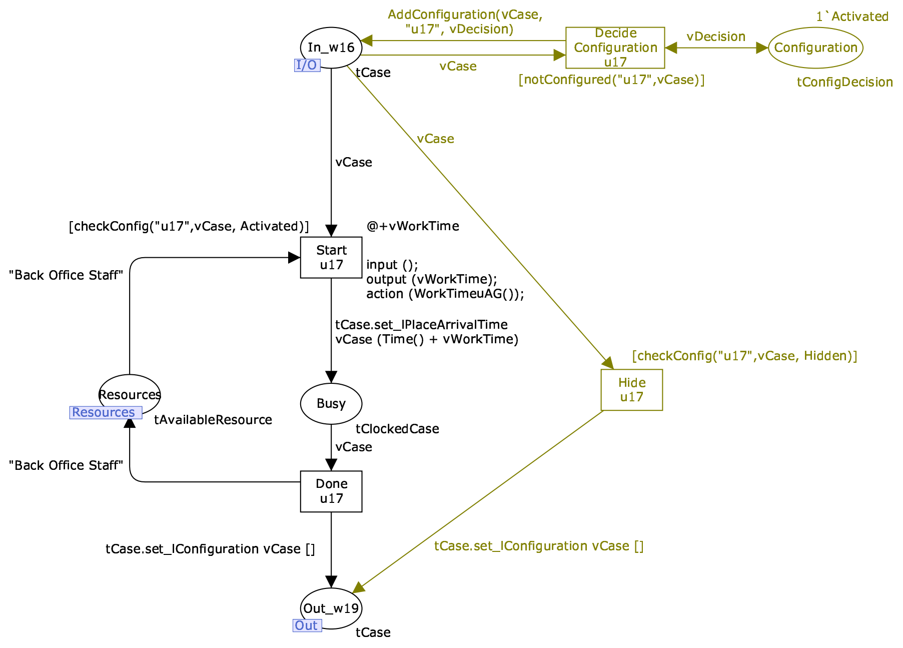


Fig. 8. A sub page of a configurable task

in the guard of the transition ensures that this decision can only be made once per task by checking the list of already added configuration decisions.

The guard of the `Start` transition ensures that the task can only be started in case it is activated, i.e. if an element of the list of configurations combines the task with the decision `Activated` (function `checkConfig (vTask: STRING, vCase: tCase, vConfigDecision: tConfigDecision)`).

In case the task is hidden, it is required to bypass the `Busy` place. This is done by an additional `Hide` transition, connecting the input place directly with the output place, without using any resources. The `Hide` transition is only enabled if the case token contains a `Hidden` decision for the particular task.

If the case is blocked for the particular task, no further behavior is allowed within this task. For that reason no transition on the sub page is able to remove a token from the input place which is blocked for the actual task. This needs to be done by another task.

If the corresponding Protos task is an XOR-join, multiple `Hide` transitions and multiple `Decide Configuration` transitions will be introduced, similar to the multiple `Start` transitions introduced in Section 2. As the `Hide` transitions combine the `Start`- and the `Done` transitions, additional `Hide` transitions must also be introduced in case of an XOR-split. Then each `Hide` transition combines one of the alternative input places with one of the alternative output places. So the maximum amount of `Hide` transitions (in case of an XOR-join and an XOR-split) is “incoming arcs of the Protos task” times “outgoing arcs of the Protos task”. As the embedding is analog to the description and the figures 5 and 7 of the non-configurable model in Section 2, we omit further figures at this point.

To enable new configurations for the same task in subsequent occurrences, the list of configuration decisions is set back to the empty list by the function `tCase.set_lConfiguration vCase []` before the case token leaves a sub-page via the `Out`-place. But note that we do not have to update the token’s arrival time.

3.3 Soundness Analysis and Limitations of Configured Process Models

After implementing configurable process models and configuring them, we are now able to test the configured process models either on their feasibility, i.e. their soundness, or on their performance. The notion of sound workflow nets [1] expresses the minimal requirements any workflow (and therefore also any executable process model) should satisfy. Simulation allows for the evaluation of various configurations in the same manner as described in the end of Section 2. By simulating different configurations the results can also be compared. In this paper we will, however, only focus on the testing on soundness as the basic prerequisite for every process model.

A workflow net is sound if it satisfies the following conditions:

1. *Option to complete*: There should always be an option to complete a case that is handled according to the process definition.
2. *Proper completion*: A workflow should never signal completion of a case while there is work in progress on this case.
3. *No dead tasks*: For every task there should at least be a chance that it is executed, i.e. for each task there should be at least one execution sequence that includes the task.

Although, theoretically only the configured models need to be sound, we require within this approach that the reference model itself is sound as well. In this way we ensure that every model element can be part of a sound process model. Otherwise these elements would have to be blocked in all configurations and would therefore be superfluous. The soundness of the reference model can be tested in Protos using Woflan [23]. After configuring the reference model we can use CPN Tools' state space analysis tool to check how far the soundness conditions are satisfied for a configured model. As the size of the state space may grow exponentially with the size of the process model, the model's complexity and its initial marking are reduced for soundness testing as follows:

– The process model is reduced to a single case as cases are handled independently and hence interactions among cases cannot invalidate soundness.
– All timing aspects are neglected as the untimed net includes every order of task execution.
– All resource requirements are neglected as soundness is purely defined on the control-flow perspective (because resource requirements do not influence soundness as long as no task requires more resources of a certain type than resources of this type are in total defined).

To test the first condition "Option to complete" our approach requires to check the list of dead markings, i.e. the possible markings of the net in which no further behavior is possible: If in such a marking a token remains in a place which is not the "final place" of the process, the condition is violated. In a dead marking, tokens cannot remain in the busy places of sub-pages as the `Done` transitions will always be enabled after a `Start` transition has fired. It is therefore possible to test if a dead marking exists that marks a place on the process overview page (Figure 2). Such a case will never be completed as the Protos model (and therefore also the CPN process overview model) completes with the execution of the last task[5]. Then the condition is violated and the configured net is not sound. However, this approach does not cover livelocks: If the process models contains loops which can be entered by tokens, but all tasks allowing to exit this loop are blocked this check will fail because then the tokens will "circle" without reaching a dead state or completing. Such a situation can only be detected by analyzing strongly connected components. Any strongly connected component that has no

[5] On the process level the last (termination) task is not connected to any subsequent status/place which could be marked by it, i.e. all tokens are "consumed" by this last task. For that reason a properly completed case leaves no tokens behind on the process overview page.

outgoing edge to any other strongly connected component, should not contain a state with a place on the process overview page marked. Otherwise the option to complete is violated and the configured net is not sound. Note that there can not be a cyclic paths between connected components, as this would result in one big connected component. As a result, the system will always be able to reach a strongly connected component that has no outgoing edge to any other strongly connected component.

It is not required to explicitly test the second condition to verify that the the reference model is sound. Configuration only restricts the possible process behavior. For that reason configured process variants cannot produce any tokens which are not produced by the complete reference model; i.e., it is impossible that new tokens which indicate the completion of the workflow are generated. Within the reference model the proper completion might, e.g., be ensured by AND-joins in the termination task. Such task behavior is kept in the configured process model even if the task itself is hidden. So the completion of a case in the configured process can only be observed if the same conditions for completion are satisfied as required by the reference model. If this is impossible, tokens will remain in the process model, which is detected by the test for the first condition.

A task of a process is dead if the `Start` transition on the task's sub-page is a dead transition. Dead tasks are indeed not desirable in a sound workflow net. However when configuring a process, i.e. restricting its possible behavior, dead tasks may be desirable. The dead tasks are the unnecessary tasks that can be removed from the configured net. When analyzing the configured net it is therefore required to check if the dead tasks are those tasks which were intended to be removed.

In the following we will use the decision-making process from Figure 2 to discuss four example configurations of this process. Making use of the results provided by the state space analysis, we will depict and analyze the purpose and sense of the configuration decisions of blocking and hiding for the selected tasks in the particular context. This analysis is far from complete as it is based on examples in which we address only selected workflow patterns. It is included here in order to highlight certain problems of process configuration.

Configuring Task `draw up reasons for approval/refusal` (Sequence)

The task `draw up reasons for approval/refusal` is located in a sequence of tasks between the task `Consult parking administration` and the task `draw up settlement`. In some municipalities it might be sufficient to draw up the settlement without explicitly drawing up reasons. In this case a single token "Hidden" is placed in the configuration place of task `draw up reasons for approval/refusal` (task number: u11, see Figure 9). If we run a state space analysis the corresponding report contains only `Task_u11'Start_u11` and `Task_u11'Done_u11` as dead transitions (besides all the other `Hide` transitions) which is exactly what we wanted to achieve: the task is never executed, but the subsequent tasks are executed.

If the task is configured as blocked (see Figure 10) the state space analysis lists further dead transitions:

```
Task_u11'Done_u11                      Task_u17'Done_u17
Task_u11'Hide_u11                      Task_u17'Hide_u17
Task_u11'Start_u11                     Task_u17'Start_u17
Task_u13'Decide_Configuration_u13      Task_u21'Decide_Configuration_w19_u21
Task_u13'Done_u13                      Task_u21'Hide_w19_u21
Task_u13'Hide_u13                      Task_u21'Start_w19_u21
Task_u13'Start_u13                     [...]
Task_u17'Decide_Configuration_u17
```

Neither the Task `draw up settlement` (task number: u13) nor the task `send judgement` (task number: u17) will ever be started. It is even never needed to decide its configuration. This means tokens will never be in the place `Reasons depicted` or `settlement suggested` which is also indicated by the upper and lower bounds of these places in the state space report which are 0. Also the task `End Decision-Making` (task number: u21) will never be executed from place `Judgment send` (place number: w19) which is indicated by the last three dead transitions.

These results of the state space analysis are not surprising as it is exactly what was intended when blocking the task. However, the configured net is not sound: In some of the dead markings a token, i.e. the case, remains in the place `Parking administration consulted` which is not the final place of the process. As depicted this is not allowed in a sound workflow net, which means that the net would remain incorrect even after removing all dead model parts. We can conclude that the blocking of a task in a sequence causes problems.

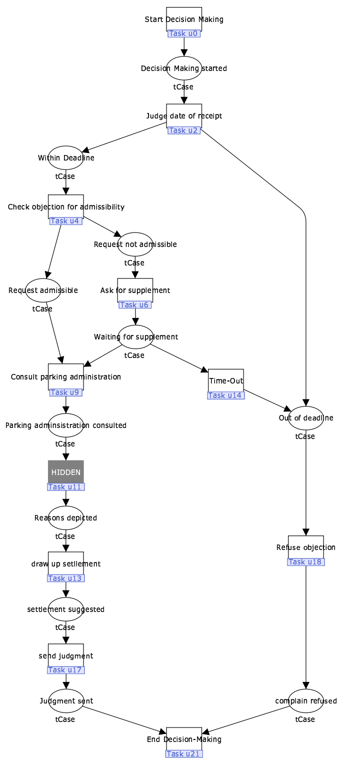


Fig. 9. **Task** `draw up reasons for approval/refusal` **hidden (grey)**

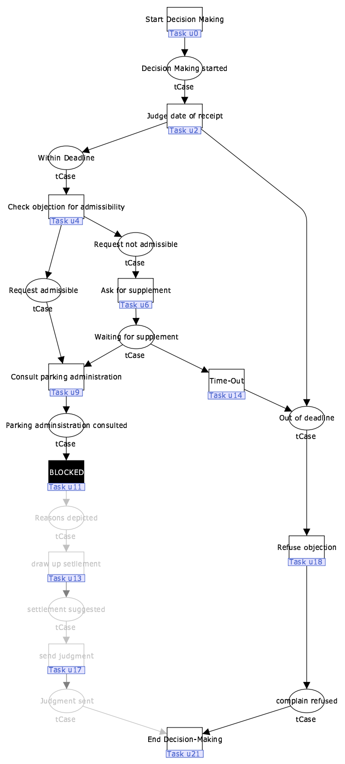


Fig. 10. **Task** `draw up reasons for approval/refusal` **blocked (black)**

Configuring Task `Time-Out` (Deferred Choice / Dummy Tasks)

The task `Time-Out` is executed when the supplement was not received within a certain period of time. This means there is a race condition between the timer triggering the `time-out` and the receival of the supplement triggering the task `Consult parking administration`. The decision which of the two tasks is executed is postponed until the execution of one of the tasks starts. Therefore this situation is also called a deferred choice. If the municipality decides that cases cannot time-out, the task `Time-Out` has to be blocked (see Figure 11). Then its `Start` and `Done` transitions are listed as dead in the corresponding state-space report. However, the state space analysis reports no dead states with tokens remaining in places other than the final place. That means in case of the construct of a deferred choice a task can be blocked without creating a deadlock.

If the task `Time-Out` is hidden (see Figure 12), it will never start nor finish but its `Hide` transition will fire, and the case reaches the `Out of deadline` place. This seems to be fine from a syntactical perspective. However when simulating the process, it becomes obvious that the behavior of the process practically conforms to the behavior of the activated `Time-Out` task. This phenomenon occurs due to the fact that the `Time-Out` task can be seen as a dummy task which is a task not corresponding to the execution of any work but introduced for changing the state of a process model, e.g. triggered by an external event. As there is no output produced, such dummy tasks are also called silent tasks or silent transitions. The hiding of such a task is questionable because in this case the effect of hiding and activating is quasi identical.

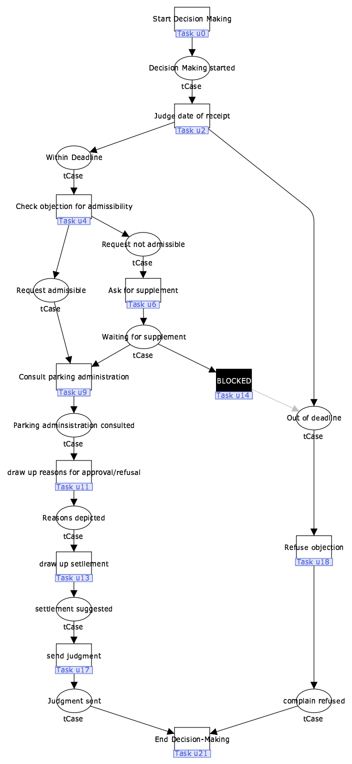


Fig. 11. **Task** `Time-Out` **blocked (black)**

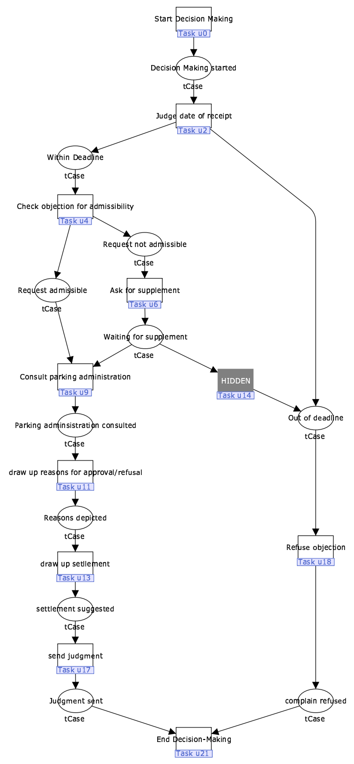


Fig. 12. **Task** `Time-Out` **hidden (grey)**

Configuring Task `Ask for supplement` (Interdependencies between Configurations)

If a municipality does not want to ask for supplements in case a request is not admissible, one could think of blocking the task `Ask for supplement` (see Figure 13). But then the municipality would end-up with cases lost in the place `Request not admissible`, never reaching the final task. So, the other option is to hide the task `Ask for supplement` which results in another issue: Non-admissible requests might time-out while waiting for an action by the municipality. Formally this is not a problem, but content-wise it could be unintended behavior. To resolve this issue and create a valid configuration, the dummy task `Time-Out` must be blocked additionally whenever the task `Ask for supplement` is hidden (see Figure 14). We can conclude that configuration decisions are not always independent of each other.

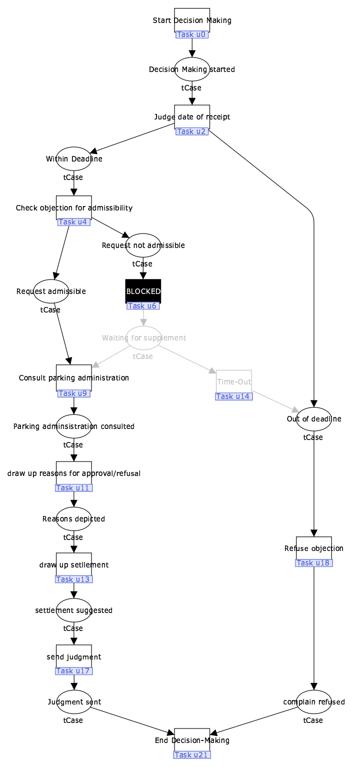


Fig. 13. Task `Ask for supplement` **blocked (black): Lost tokens may remain in the place** `Request not admissible`

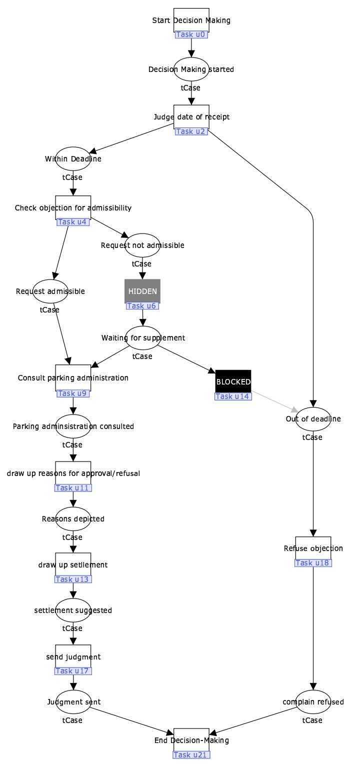


Fig. 14. To avoid asking for supplements, not only the Task `Ask for supplement` **is hidden (grey), but also task** `Time-Out` **must be blocked (black)**

Configuring Task `Judge date of receipt` (Explicit Choice)

When executing the task `Judge date of receipt` an explicit choice how the process will continue is made: When the complain was received within the deadline, it will be checked for its admissibility, whereas in case it arrives too late, it will be refused. But maybe some municipalities want to be less restrictive with the initial deadlines and consider all objections as being received within the deadline. So all objections must be checked for their admissibility. However, neither hiding nor blocking of the task `Judge date of receipt` helps here. If it is blocked the process will never go beyond this task. If it is hidden, tokens can still be placed into the `Out of deadline` place.

In the previous scenario we could achieve the desired behavior by hiding one (`Ask for supplement`) and blocking another task (`Time-Out`). But also this approach is impossible to apply as neither hiding nor blocking of task `Refuse objection` can prevent that tokens are put into the place `Out of deadline` and as soon as a token is in this place it cannot be checked for its admissibility anymore.

The only chance of enforcing the desired behavior is, to go to a lower level, i.e., to have a look at the implementation of the choice on the task page. In a standard Petri net an explicit choice can only be modelled by a deferred choice with subsequent silent transitions. In our implementation of the XOR-split, these silent transitions are the multiple `Done` transitions. Only when explicitly blocking the particular `Done` transition on this task-level (see Figure 15), it is possible to restrict the process model to the desired behavior[6]. So within a sound process model the outcome of an explicit choice cannot be restricted, i.e., configured, at the process level, but on lower implementation levels.

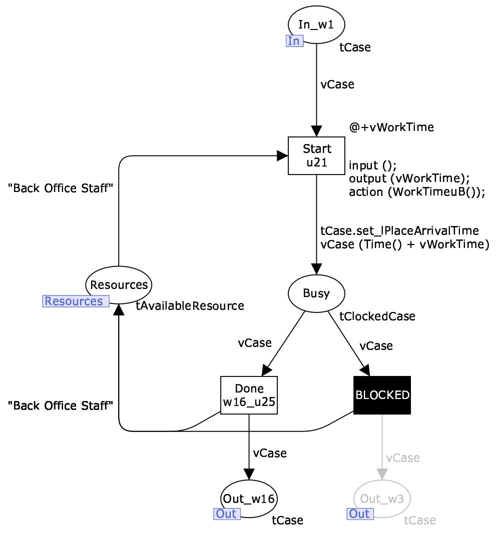


Fig. 15. `Done` **transition blocked (black) in the sub-page of the** `Judge date of receipt` **task: The task can only exit via the left path.**

This concludes our analysis of four configuration scenarios. Using the four scenarios it has been shown that the state space analysis and the simulation facilities of CPN Tools can be used to evaluate configuration decisions.

4 Conclusions

By analyzing a set of reference models designed using the business process modelling language of Protos, we discovered that these models do not conform to well defined soundness criteria which also prevents the meaningful use of Protos' simulation features. The main reasons for this are that the developers of the models either see no value in the Protos simulation, or they are not aware of its value. We also realized that it is unclear which of the parameters that can be specified in Protos are actually used in the Protos simulation. To improve the value of simulation in this context, we developed the Protos2CPN transformation which allows the simulation of Protos models in CPN Tools. The simulation of Protos models in CPN Tools

[6] The configuration of `Done` transitions is not yet implemented in the Protos2C-CPN transformation.

makes the running process visible by depicting the moving cases as tokens within the process model. It therefore allows for a detailed inspection of the running process. In addition, the monitoring features of CPN Tools enable the generation of complex statistics. The models created by our Protos2CPN transformation already include some basic measurements which can be extended by experienced users.

In a second step we developed Protos2C-CPN. As far as we know, this is the first implemented tool offering explicit support for reference model adaptation by adding standard configuration features to the tasks in the reference models. These features permit the restriction of the possible behavior of the reference model directly in the model without changing its net structure. The simulation features of CPN Tools allow for performance testing and comparison of different process configurations. By making use of CPN Tools' state space analysis feature, we were able to test configurations on their admissibility in sound process models, but also realized that certain configurations are undesirable in specific contexts.

However, it might be possible to resolve such problems on lower model levels. To do this we plan to explore configuration in the context of the workflow patterns [6]. We assume that by analyzing all workflow patterns on their configurability aspect, it might be possible to develop configuration patterns which depict how configuration can be implemented in the context of the particular workflow pattern. If such patterns are available, we could develop an improved version of Protos2C-CPN which might even be able to transform the configured model back into an ordinary process model without configuration features.

Acknowledgements

The authors would like to thank Kurt Jensen and Lisa Wells for their continuous feedback and support related to the use of CPN Tools, as well as Pallas Athena for providing the Protos reference models.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.
3. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, January 2002.
4. W.M.P. van der Aalst, P.J.N. de Crom, R.R.H.M.J. Goverde, K.M. van Hee, W.J. Hofman, H.A. Reijers, and R.A. van der Toorn. ExSpect 6.4 An Executable Specification Tool for Hierarchical Colored Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 455–464, Berlin, Germany, January 2000. Springer.
5. W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable Process Models as a Basis for Reference Modeling. In C. Bussler and A. Haller, editors, *Business Process Management Workshops: BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS. Revised Selected Papers*, volume 3812 of *Lecture Notes in Computer Science*, pages 512–518, Berlin Heidelberg, 2006. Springer Verlag.
6. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
7. T. Basten and W.M.P. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.

8. J. Becker, P. Delfmann, A. Dreiling, R. Knackstedt, and D. Kuropka. Configurative Process Modeling – Outlining an Approach to increased Business Process Model Usability. In *Proceedings of the 15th Information Resources Management Association International Conference*, New Orleans, 2004. Gabler.
9. J. Becker, P. Delfmann, and R. Knackstedt. Konstruktion von Referenzmodellierungssprachen: Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle (in German). *Wirtschaftsinformatik*, 46(4):251–264, 2004.
10. J. vom Brocke and C. Buddendick. Konstruktionstechniken für die Referenzmodellierung (in German). In J. Becker and P. Delfmann, editors, *Referenzmodellierung. Grundlagen, Techniken und domänenbezogene Anwendung, also Proceedings of the 8th Fachtagung Referenzmodellierung*, pages 19–48, Heidelberg, 2004.
11. Thomas Curran, Gerhard Keller, and Andrew Ladd. *SAP R/3 Business Blueprint: understanding the business process reference model.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1998.
12. Deloitte & Touche Bakkenist. ExSpect Home Page. http://www.exspect.com.
13. F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Configurable Process Models – A Foundational Approach. In *Referenzmodellierung 2006*, Passau, Germany, 2006. to appear in Lecture Notes in Computer Science.
14. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
15. R.J. Paul, G.M. Giaglis, and V. Hlupic. Simulation of business processes. *The American Behavioral Scientist*, 42(10):1551–1576, August 1999.
16. M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 2005. (to appear, also available via BPMCenter.org).
17. K. Sarshar and P. Loos. Comparing the Control-Flow of EPC and Petri Net from the End-User Perspective. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Managemen (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 434–439, Nancy, France, September 2005. Springer-Verlag.
18. R. Schütte. *Grundsätze ordnungsmäßiger Referenzmodellierung – Konstruktion konfigurations- und anpassungsorientierter Modelle (in German).* Gabler, Wiesbaden, 1998.
19. A. Schwegmann. *Objektorientierte Referenzmodellierung: theoretische Grundlagen und praktische Anwendung (in German).* Gabler, Wiesbaden, 1999.
20. A. Sharp and P. McDermott. *Workflow Modeling: Tools for Process Improvement and Application Development.* Artech House Publishers, Norwood, MA, 2001.
21. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan Home Page, Eindhoven University of Technology, Eindhoven, The Netherlands. http://is.tm.tue.nl/research/woflan.
22. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer-Verlag, Berlin, 2000.
23. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
24. H.M.W. Verbeek, M. van Hattem, H.A. Reijers, and W. de Munk. Protos 7.0: Simulation Made Accessible. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005: 26th International Conference (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 465–474, Miami, USA, June 2005. Springer-Verlag.
25. A. Vinter Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In W.M.P. van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer Verlag, January 2003.
26. L. Wells. Monitoring a CP-net. http://wiki.daimi.au.dk/cpntools-help/monitoring_a_cp-net.wiki.

Sweep-line Analysis of DCCP Connection Management

Somsak Vanit-Anunchai, Jonathan Billington and Guy Edward Gallasch

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

Abstract. State space explosion is a key problem in the analysis of finite state systems. The sweep-line method is a state exploration method which uses a notion of progress to allow states to be deleted from memory when they are no longer required. This reduces the peak number of states that need to be stored, while still exploring the full state space. The technique shows promise but has never achieved reductions greater than about a factor of 10 in the number of states stored in memory for industrially relevant examples. This paper discusses sweep-line analysis of the connection management procedures of a new Internet standard, the Datagram Congestion Control Protocol (DCCP). As the intuitive approaches to sweep-line analysis are not effective, we introduce new variables to track progress. This creates further state explosion. However, when used with the sweep-line, the peak number of states is reduced by over two orders of magnitude compared with the original. Importantly, this allows DCCP to be analysed for larger parameter values.

Keywords: State Space Methods, Sweep-line, DCCP, Coloured Petri Nets, State Explosion.

1 Introduction

The state space method is one of the main approaches for formally analysing and verifying the behaviour of concurrent and distributed systems. In essence, this method generates all or part of the reachable states of the system. After the state space is generated, many analysis and verification questions about the system's behaviour (such as "does the system deadlock and livelock?"), can be answered. Unlike theorem proving, state space analysis tools are easier to use because they involve less complex mathematics that is often hidden by automatic computer tools, and they can provide counter examples for debugging purposes. Despite these advantages, the state space approach suffers from the well known *state explosion problem* [23]. Even relatively small systems can generate state spaces that cannot be stored in computer memory. Thus many attempts have been made to alleviate this problem. An excellent overview and literature review about the state explosion problem is given in [23]. The attempts to alleviate state explosion fall into three broad classes. The first considers methods that represent the state space in a condensed or compact form, such as symmetry reduction [4,13]. The second class restricts state space exploration to a subset of the reachable states and includes partial order methods [21,26] such as stubborn sets [22]. The third class involves deleting or throwing away states or state information on-the-fly during state space exploration. It includes methods such as bit-state hashing [11,27], state space caching [8,10], the sweep-line method [2,15] and the pseudo-root technique [20]. To further reduce the state space, techniques from the different classes have also been combined, e.g. sweep-line and equivalence [1].

The sweep-line method exploits a notion of *progress* exhibited in the system being analysed. *Progress mappings* are defined by the user, to map the states of the system into a set of ordered progress values. Based on progress values, this method deletes old states from memory by reasoning that states with a lower progress value will not (or are unlikely to) be reached from states with a higher progress value. During exploration, a number of system properties, such as absence of deadlocks, can be verified on-the-fly.

In the area of formal analysis and verification of computer protocols that recover from loss using retransmissions, the maximum number of retransmissions of messages plays an important role in state explosion. When the maximum number of retransmissions increases, the size of the state space tends to explode rapidly. When using a small maximum number of retransmissions, errors tend to reveal themselves quickly, but this does not guarantee that the system is error free for larger values of the maximum number of retransmissions. Thus it is necessary to extend state space analysis to include the highest maximum number of retransmissions possible, up to the limit specified by the protocol's specification. Previously we have used the sweep-line method to verify termination properties of Coloured Petri Net (CPN) [12, 17] models of various protocols such as the Wireless Transaction Protocol (WTP) [9], the Internet Open Trading Protocol (IOTP) [6], and the Transmission Control Protocol [5].

The Datagram Congestion Control Protocol (DCCP), specified in Request For Comments (RFC) 4340 [14], is a new transport protocol proposed by the Internet Engineering Task Force (IETF). The protocol is designed to support various kinds of congestion control mechanisms used by different delay sensitive applications. We use CPNs to build and analyse a formal executable model of DCCP's connection management procedures according to RFC 4340 [14]. Our analysis shows that DCCP connection establishment can fail when sequence numbers wrap[1]. However, state explosion limits our analysis to a maximum of only one retransmission. We need to extend the analysis to cover two retransmissions and to determine two properties: whether the undesired deadlocks are still present; and whether any new errors have emerged as a result of the additional retransmissions.

Instead of applying the more conventional approach of trying to reduce the size of the state space, we induce state space explosion by introducing new state variables into the *specification model* in order to capture additional information during model execution. By carefully selecting new state variables, the state space of the *augmented model* has a structure which facilitates far more efficient sweep-line analysis when compared to the state space of the specification model.

The contribution of this paper is two fold. Firstly, this paper provides some insight into how an effective progress mapping for the DCCP connection management model is derived. An effective progress mapping is key to the performance of the sweep-line. Secondly, we apply sweep-line analysis to the augmented model. This method can significantly reduce the number of peak states stored compared with the sweep-line analysis of the specification model. Thus we can extend the analysis of DCCP connection establishment to scenarios which could not be reached with conventional analysis.

This paper is organised as follows. Section 2 briefly describes the DCCP connection management CPN model. Section 3 identifies sources of progress and derives a progress mapping for DCCP connection management. The sweep-line analysis results obtained for scenarios where sequence numbers wrap are discussed in Section 4. Section 5 presents conclusions and future work. We assume some familiarity with CPNs [12, 17].

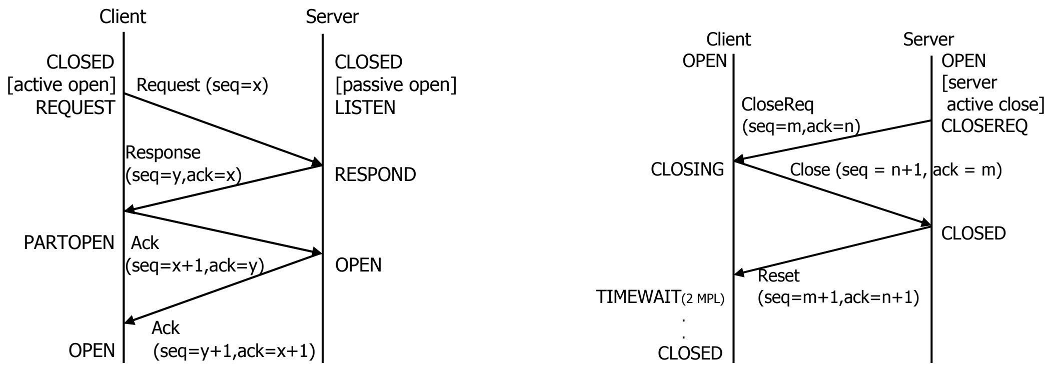
[1] Sequence number wrap occurs when the sequence number rolls over the maximum sequence number to zero.

2 CPN specification model of DCCP connection management

DCCP connection management has one connection establishment procedure and five closing procedures. It can be represented by a state diagram shown in Fig. 1 comprising nine states: CLOSED, LISTEN, REQUEST, RESPOND, PARTOPEN, OPEN, CLOSEREQ, CLOSING and TIMEWAIT. DCCP uses 8 packet types: Request, Response, Ack, DataAck, Data, CloseReq, Close, and Reset to set up, transmit data and close down the connection. Two packet types, Sync and SyncAck, are used to re-synchronize the sequence number variables. Figure 2 shows the typical procedure for connection set up (Fig. 2 (a)) and close down (Fig. 2 (b)).

```
+---------------------------+    +---------------------------+
|                           v    v                           |
|                        +----------+                         |
|          +-------------+  CLOSED  +------------+            |
|          | passive     +----------+  active    |            |
|          |  open                       open    |            |
|          |                        snd Request  |            |
|          v                                     v            |
|     +----------+                         +----------+       |
|     |  LISTEN  |                         | REQUEST  |       |
|     +----+-----+                         +----+-----+       |
|          | rcv Request           rcv Response |            |
|          | snd Response            snd Ack    |            |
|          v                                     v            |
|     +----------+                         +----------+       |
|     | RESPOND  |                         | PARTOPEN |       |
|     +----+-----+                         +----+-----+       |
|          | rcv Ack/DataAck         rcv packet  |            |
|          |                                     |            |
|          |             +----------+            |            |
|          +------------>|   OPEN   |<-----------+            |
|                        +--+-+--+--+                         |
|       server active close | |  |   active close            |
|           snd CloseReq    | |  | or rcv CloseReq           |
|                           | |  |     snd Close             |
|                           | |  |                           |
|     +----------+          | |  |            +----------+   |
|     | CLOSEREQ |<---------+ |  +----------->| CLOSING  |   |
|     +----+-----+            |               +----+-----+   |
|          | rcv Close        |          rcv Reset |         |
|          | snd Reset        |                    |         |
|<---------+                  |                    v         |
|                             |               +----+-----+   |
|                   rcv Close |               | TIMEWAIT |   |
|                   snd Reset |               +----+-----+   |
+-----------------------------+                    |         |
                                                   +---------+
                                              2MSL timer expires
```

Fig. 1. DCCP state diagram [14].



(a) (b)

Fig. 2. Typical connection establishment and release scenarios.

In brief, a connection is initiated by an application at the client issuing an “active open” command. (We assume that the application at the server has issued a “passive open” command.) After receiving the “active open” the client sends a DCCP-Request packet to the server to initialise sequence numbers, and enters the REQUEST state. The server replies with a DCCP-Response packet, indicating that it is willing to communicate with the client and acknowledging the DCCP-Request. The client sends a DCCP-Ack (or DCCP-DataAck) packet to acknowledge the DCCP-Response packet and enters the PARTOPEN state. The server acknowledges the receipt of the DCCP-Ack, enters the OPEN state and is ready for data transfer. Upon receipt of a DCCP-Ack (or DCCP-Data, DCCP-DataAck or DCCP-SyncAck) packet, the client also enters OPEN and is now also ready for data transfer.

For connection close down, the application at the server issues a “server active close” command. The server sends a DCCP-CloseReq and enters the CLOSEREQ state. The client, upon receiving the DCCP-CloseReq, enters CLOSING and generates a DCCP-Close packet in response. After the server receives the DCCP-Close packet, it responds with a DCCP-Reset packet and enters the CLOSED state. When the client receives the DCCP-Reset packet, it holds the TIMEWAIT state for 2 maximum packet lifetimes before also entering the CLOSED state.

Alternatively, either side may send a DCCP-Close packet to close the connection when receiving an “active close” command from the application. The end that sends the DCCP-Close packet will hold the TIMEWAIT state as shown in Fig. 3. Beside these three closing procedures, there are another 2 possible scenarios concerned with simultaneous closing. The first procedure is invoked when both users issue an “active close”. The second occurs when the client user issues an “active close” and the application at the server issues the “server active close” command. For a detailed description of the connection set up and close down procedures, see [14].

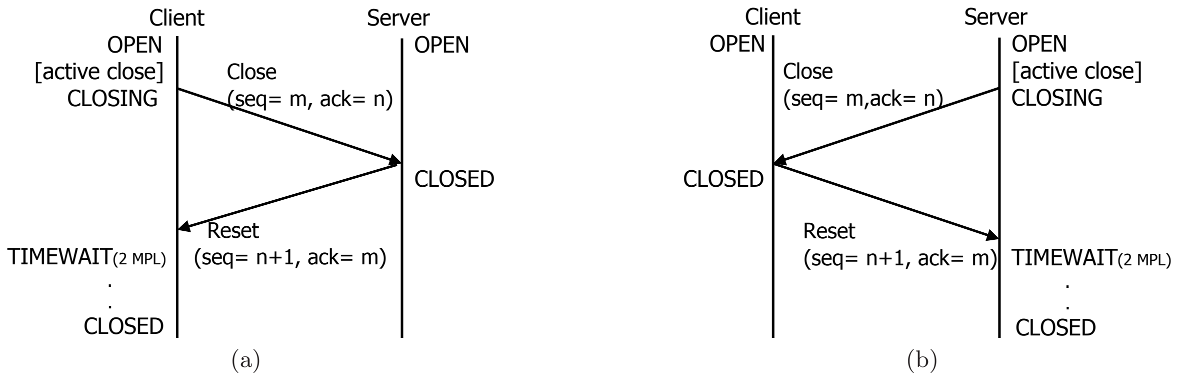


Fig. 3. Alternative close down procedures.

During the connection, DCCP entities maintain a set of variables. In addition to the state and timers, the important variables are Greatest Sequence Number Sent (GSS), Greatest Sequence Number Received (GSR), Greatest Acknowledgement Number Received (GAR), Initial Sequence Number Sent and Received (ISS and ISR), Valid Sequence Number window width (W) and Acknowledgement Number validity window width (AW). Based on the state variables, the valid sequence and acknowledgement number intervals are defined by Sequence

Number Window Low and High [SWL,SWH], and Acknowledgement Number Window Low and High [AWL,AWH].

We use Design/CPN [19] to build and maintain our DCCP connection management CPN models. In this paper we are restricted to describing details of the model at a high level only. More details can be found in [25].

2.1 Modelling assumptions

Our model represents a detailed DCCP connection management specification of RFC 4340. We consider a single connection instance while ignoring the procedures for data transfer, congestion control and other feature options. A DCCP packet is modelled by its packet type, and long (or short) sequence and acknowledgement numbers. Other fields in the DCCP header are omitted because they do not affect the operation of the connection management procedure. Malicious attacks are not considered. In this paper we only discuss the case when the communication channels can delay and reorder packets without loss.

2.2 Model structure and the top level page

The DCCP Connection management (DCCP-CM) CPN model comprises four hierarchical levels shown in Fig. 4. It has a total of 6 *places*, 53 *executable transitions*, 15 *substitution transitions* and 18 *ML functions*. The top level page named DCCP#3 is the DCCP overview page shown in Fig. 5. Two *substitution transitions* DCCP_C and DCCP_S in Fig. 5 represent the client and the server. Both are linked to the second level page named DCCP_CM. The client and server communicate via two channel places, shown in the middle of Fig. 5. The places Ch_C_S and Ch_S_C each model a unidirectional reordering channel, from the client to the server and the server to the client respectively.

2.3 Global declarations

The global declarations define data structures, colour sets and any associated variables used in the model. The declarations are written in CPN ML [3], a variant of Standard ML [18]. An example of the Declarations is given in Fig 6. Figure 6 declares *PACKETS* (line 21-24), the colour set of the channel places, as a union of four colour sets: Type1LongPkt, Type2LongPkt, Type1ShortPkt, and Type2ShortPkt. Long sequence numbers (SN48) in line 11 are represented using the *infinite integer* type and range from zero to $2^{48} - 1$. Short sequence numbers (SN24) in line 13 are represented by integers ranging from zero to $2^{24} - 1$. Each packet (lines 17-20) is defined by a product comprising the Packet Type (lines 2-5), X (Extended Sequence Number bit, called 'X' in [14], line 8) and the sequence number (or a record of sequence and acknowledgement numbers). Short sequence numbers are allowed only for DCCP-Data, DCCP-Ack and DCCP-DataAck packets. Thus line 4 defines a packet type for DCCP-Data with short sequence numbers, while line 5 defines a different packet type for DCCP-Ack and DCCP-DataAck because they also include acknowledgements. Strong typing of packets is very useful for developing and debugging the model. Strong typing allows automatic detection of violations of the typing rules via a syntax check.

The places Client_State and Server_State in Fig. 5, typed by CB (Control Block), store DCCP state information for the client and server entities respectively. We classify DCCP states into three groups according to their functional behaviour: idle, request and active

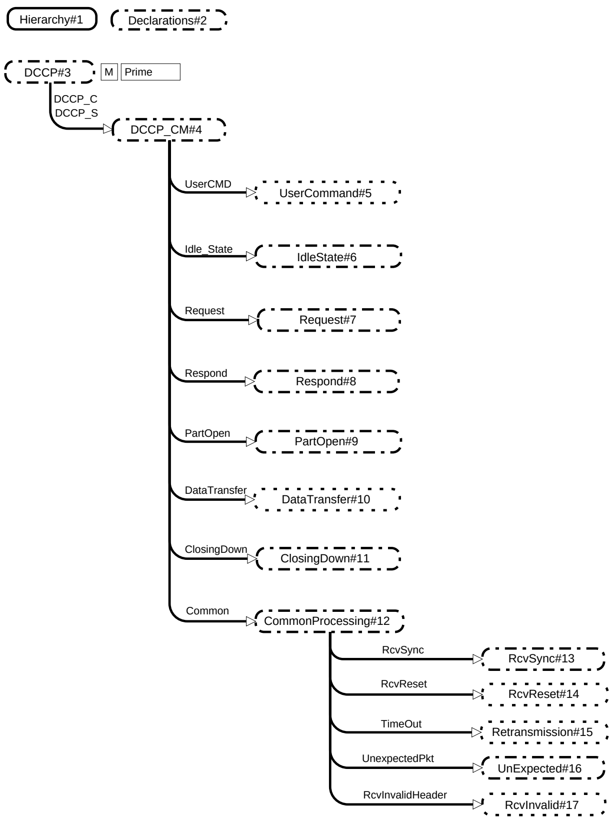


Fig. 4. The DCCP hierarchy page.

states. The differences are mainly related to how an entity responds to DCCP-Reset and DCCP-Sync packets. When in the CLOSED, LISTEN and TIMEWAIT states, the GSS, GSR, GAR, ISS and ISR state variables do not exist, while the client in the REQUEST state has only GSS and ISS instantiated. Thus we define each group with a different set of state variables. Figure 7 defines CB (line 13-15) as a union of colour sets: IDLE, REQUEST and ACTIVExRCNTxGSxISN.

IDLE (line 7 of Fig 7) defines three idle states: CLOSED, LISTEN and TIMEWAIT. The CLOSED state is split into CLOSED_I to represent the initial CLOSED state and CLOSED_F to represent a terminal CLOSED state. Differentiating the initial CLOSED state and the final CLOSED state helps increase the effectiveness of the sweep-line method when using the client and server states as a measure of progress. Because we consider only one connection instance, splitting the CLOSED state into CLOSED_I and CLOSED_F does not affect the protocol's behaviour. The colour set REQUEST (line 8) is a product comprising RCNT (Retransmission Counter, line 2), GSS and ISS. Because there is only one state in this group, the REQUEST state is already distinguished from other states by using the ML constructor, ReqState, in the union defined in line 14. ACTIVE (line 9) defines five DCCP states: RESPOND, PARTOPEN, OPEN, CLOSEREQ and CLOSING. Because the client and server respond to the CloseReq packet differently in the OPEN and CLOSING states, we differentiate these states for the client and server: C_OPEN and C_CLOSING for the client; and S_OPEN and S_CLOSING for the server. ACTIVExRCNTxGSxISN (line 10) is a product comprising ACTIVE (line 9), RCNT, GS (Greatest Sequence and Acknowledgement Numbers, line 3) and ISN (Initial

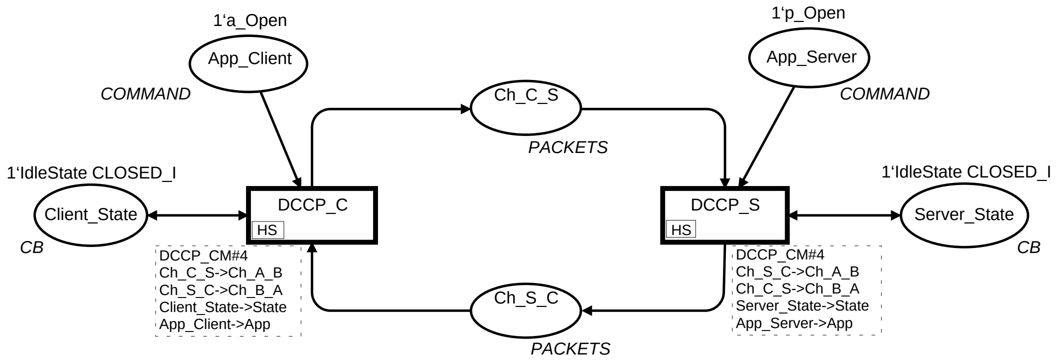


Fig. 5. The DCCP overview page.

```
 1: (* Packet Types *)
 2: color PktType1 = with Request | Data;
 3: color PktType2 = with Sync | SyncAck | Response | Ack | DataAck | CloseReq | Close | Rst;
 4: color DATA = subset PktType1 with [Data];
 5: color ACK_DATAACK = subset PktType2 with [DataAck,Ack];
 6:
 7: (* Extended Sequence Number Bit*)
 8: color X = bool;
 9:
10: (* Sequence and Acknowledgement Number (Long/Short) *)
11: color SN48 = IntInf with ZERO..MaxSeqNo48;
12: color SN48_AN48 = record SEQ:SN48*ACK:SN48;
13: color SN24 = int with 0..max_seq_no24;
14: color SN24_AN24 = record SEQ:SN24*ACK:SN24;
15:
16: (* Four Kinds of Packet *)
17: color Type1LongPkt=product PktType1*X*SN48;
18: color Type2LongPkt=product PktType2*X*SN48_AN48;
19: color Type1ShortPkt=product DATA*X*SN24;
20: color Type2ShortPkt=product ACK_DATAACK*X*SN24_AN24;
21: color PACKETS = union PKT1:Type1LongPkt
22:                     + PKT2:Type2LongPkt
23:                     + PKT1s:Type1ShortPkt
24:                     + PKT2s:Type2ShortPkt;
```

Fig. 6. Definition of DCCP PACKETS.

Sequence Numbers, line 4). Places Client_State and Server_State (Fig. 5) both have an initial marking of one CLOSED_I idle state token.

Figure 7 also defines a colour set COMMAND on line 17. The places App_Client and App_Server in Fig 5, typed by COMMAND, model DCCP user commands (i.e. commands that can be issued by the applications that use DCCP). For example, the user command 1'a_Open is the initial marking of App_Client indicating that the client's application desires to open a connection.

```
(* DCCP variables: Counter; Greatest Sequence Numbers; Initial Sequence Numbers *)
color RCNT = int;     (* Retransmission Counter *)
color GS = record GSS:SN48*GSR:SN48*GAR:SN48;
color ISN = record ISS:SN48*ISR:SN48;

(* Major States *)
color IDLE = with CLOSED_I | LISTEN | TIMEWAIT | CLOSED_F;
color REQUEST = product RCNT*SN48*SN48;
color ACTIVE = with RESPOND | PARTOPEN | S_OPEN | C_OPEN | CLOSEREQ | C_CLOSING |S_CLOSING;
color ACTIVExRCNTxGSxISN = product ACTIVE*RCNT*GS*ISN;

(* DCCP's Control Block  *)
color CB = union IdleState:IDLE
               + ReqState:REQUEST
               + ActiveState:ACTIVExRCNTxGSxISN;
(* Application Commands *)
color COMMAND = with p_Open | a_Open | a_Close | server_a_Close;
```

Fig. 7. DCCP's control block and commands.

3 Sweep-line analysis

The success of applying sweep-line relies on the notion of a *progress measure* which is defined [2] as a tuple $\mathcal{P} = (\mathcal{O}, \sqsubseteq, \varphi)$, where $\mathcal{O}$ is a set of progress values, $\sqsubseteq \subseteq \mathcal{O} \times \mathcal{O}$ is a partial ordering operator on the progress values, and $\varphi : \mathbb{M} \rightarrow \mathcal{O}$ is a progress mapping function from markings of the CPN model, $\mathbb{M}$, to progress values. Gordon et al. [9] point out that the protocol can exhibit more than one source of progress and suggests the use of a vector of progress values that we shall call a progress vector. In [5] the progress vector is used in conjunction with lexicographical ordering. We use this approach to analyze the DCCP-CM CPN model.

The sweep-line algorithm generates the successors of all unexplored states with the lowest progress value first. Once all states with this lowest progress value have been explored, they will be deleted from memory and the conceptual "sweep-line" will move on to states with the new lowest progress value. A progress mapping is said to be *monotonic* if, for every reachable state, it has a progress value equal to or less than all of its successors. When this is not the case, i.e. at least one successor of one state has a progress value that is less than its predecessor (representing *regress* rather than progress), the sweep-line must conduct additional sweeps of (part of) the state space, using the destinations of these so-called *regress edges* as roots of a new sweep. Thus, some parts in the state space may be explored more than once. However, the sweep-line still guarantees full exploration of a state space and is guaranteed to terminate. Detailed explanations of the sweep-line method can be found in [2, 15].

3.1 Notation

We define some notation used for identifying sources of progress in a CPN model. Let $M \in \mathbb{M}$ be a marking of the CPN model. $M(p)$ is the marking of place p (the multiset of tokens on place p) and $\|M(p)\|$ is the number of tokens on place p. Measures of progress, such as the greatest sequence number sent, are often encapsulated in a product token residing on a state place, e.g. Client_State. To extract information from a product token we shall use projection functions. Our progress mappings operate on markings, which are multisets of tokens rather than tokens themselves. However, when p is a state place, it only contains one token ($\forall M \in [M_0\rangle,\ ||M(p)|| = 1$, where $[M_0\rangle$ is the set of reachable markings), and hence $M(p)$ is a *singleton multiset*. Therefore in this situation, if we wish to extract information from the single token in such a place we firstly need to convert the marking of the place, which is a singleton multiset, into its basis element (i.e. the token). We define this conversion function (*conv*) as follows using standard set theory to represent a multiset (a function) by a set of pairs comprising a basis element (i.e. from its domain) and its multiplicity (from its range).

Definition 1. *Let $MS : B \to \mathbb{N}$ be a multiset over B and B_{MS_1} be the set of all singleton multisets over a basis set B: $B_{MS_1} = \{\{(b, 1)\}|\ b \in B\}$. A function that converts a singleton multiset to its basis element is given by $conv : B_{MS_1} \to B$, where $conv(\{(b, 1)\}) = b$.*

3.2 Usual sources of progress

The following describes three usual sources of progress.

1. Sequence Number Variables Firstly, consider GSS. Every time the client or server sends a packet, the state variable GSS (the Greatest Sequence number Sent) increases by one. GSS is stored within a single product token in the places Client_State and Server_State. However when an entity is in an idle state (CLOSED, LISTEN and TIMEWAIT), there is no GSS. To capture the progress of GSS from the marking of the place Client_State we define a progress mapping for the client $\varphi^c_{gss} : \mathbb{M} \to \mathbb{N}$, where the superscript "c" refers to the Client entity and

$$\varphi^c_{gss}(M) = Proj_{gss}(conv(M(Client\_State))) \tag{1}$$

and $Proj_{gss}$ takes a state variable of type CB and returns GSS for active states and ISS otherwise. During connection establishment and close down, the number of transmitted packets is small. We consider that the sequence number may wrap only once. If the GSS value is less than ISS, it means that the GSS value has wrapped. In this situation, $Proj_{gss}$ returns GSS plus 2^{48} in order to maintain increasing progress values. If there is no GSS value in the state variable, $Proj_{gss}$ returns ISS because it is the starting value of GSS for each connection. The progress mapping, φ^s_{gss}, φ^c_{gsr}, φ^s_{gsr}, φ^c_{gar} and φ^s_{gar} for other client and server sequence number variables can also be defined in a similar way.

2. Major States Both entities progress through the states defined by colour sets IDLE, REQUEST and ACTIVE. To capture this progress from the token in the place Client_State, we define a progress mapping $\varphi^c_{state} : \mathbb{M} \to \mathbb{N}$ where

$$\varphi^c_{state}(M) = State2Num(Proj_{state}(conv(M(Client\_State)))) \tag{2}$$

The projection function, $Proj_{state}$, takes a state variable of type CB and returns the DCCP state. Function $State2Num$ maps this state to an integer according to the ordering of DCCP states shown in Column 1 of Table 1. The server's progress mapping, φ^s_{state}, is defined analogously, using the ordering of DCCP states shown in Column 3 of Table 1.

Because of lost and delayed packets, an entity may retransmit. The progress exhibited by the retransmission counters (RCNTs) has already been covered by GSS because GSS is increased for every packet sent, including retransmissions. Thus the progress captured by RCNTs is not needed. However when retransmission occurs, the state is not changed, and hence, intuitively, it is useful to include both major state and GSS in progress mappings.

Table 1. An ordering and corresponding mapping for DCCP state.

Client state	$State2Num(state)$	Server state	$State2Num(state)$
CLOSED_I	1	CLOSED_I	1
REQUEST	2	LISTEN	2
PARTOPEN	3	RESPOND	3
C_OPEN	4	S_OPEN	4
C_CLOSING	6	C_CLOSEREQ	5
TIMEWAIT	7	S_CLOSING	6
CLOSED_F	8	TIMEWAIT	7
		CLOSED_F	8

3. Application Commands During connection set up and close down, the applications at the client and server will issue commands. The progress of issuing commands can be captured by the decrease in the total number of command tokens in both the App_Client and App_Server places. This can help to differentiate between the CLOSED_F state caused by timer expiry in the TIMEWAIT state and the CLOSED_F state resulting from an application close command. Thus we define $\varphi_{cmd} : \mathbb{M} \rightarrow \mathbb{N}$ where

$$\varphi_{cmd}(M) \;=\; 4 \;-\; \|M(App\_Client)\| \;-\; \|M(App\_Server)\| \tag{3}$$

In this paper we do not consider scenarios in which an application issues more than two commands, and thus the constant "4" ensures that $\varphi_{cmd}(M)$ starts small (possible zero) and increases to 4 as application commands are issued. Every application command issued corresponds to a change of state. However other DCCP state changes also occur due to internal behaviour. Hence we consider a state change due to an application command to be a more significant event. Thus we give φ_{cmd} greater weighting than φ^{c}_{state} and φ^{s}_{state}.

3.3 More subtle measures of progress

From our experience with DCCP, more than 90% of the state space has at least one entity in CLOSED, which has no sequence number variables. φ^{c}_{state} and φ^{c}_{gss} provide no differentiation when the client is CLOSED, and similarly for the server. Thus some measure of progress is needed for when an entity is in the CLOSED state. When an entity is in an idle state (CLOSED,LISTEN and TIMEWAIT), there are two ways in which progress is exhibited:

a) **Receiving DCCP-Reset Packets** When either end receives a DCCP-Reset Packet, the total number of packets in both channel places will decrease by one. This is simply because in CLOSED, LISTEN and TIMEWAIT, the entity discards the DCCP-Reset and does not send packet in response. To capture this measure of progress we define $\varphi_{ch\_num} : \mathbb{M} \rightarrow \mathbb{N}$ where

$$\varphi_{ch\_num}(M) \;=\; 1000 \;-\; \|M(Ch\_C\_S)\| \;-\; \|M(Ch\_S\_C)\| \tag{4}$$

For the scenarios considered in this paper, the total number of packets in both channels will never be more than 1000 (this can be checked on-the-fly). Thus "1000" is used in the above progress mapping. Note that this progress mapping initially decreases as the first packets are sent into the channel. However, when component progress measures are combined using lexicographical ordering (as will be done shortly) this *regress* is more than offset by *progress* captured in other 'more significant' component progress measures when the number of messages in the channel are increasing.

b) **Receiving non-DCCP-Reset Packets** When any packet but DCCP-Reset is received from one channel by an entity in an idle state, it will send a DCCP-Reset packet into the other channel in response, so that the total number of packets over both channels remains the same. Owing to no GSS, GSR or GAR in these states, the DCCP-Reset packet sent will have a sequence number set to the acknowledgement number of the received packet plus one, and an acknowledgement number set to the sequence number of the received packet. Thus the summation of all sequence numbers and acknowledgement numbers in all packets over both channel places will be increased by one. Thus we define a progress mapping $\varphi_{ch\_sum} : \mathbb{M} \rightarrow \mathbb{N}$ to capture the progress when an entity in an idle state replies with a Reset packet, where

$$\varphi_{ch\_sum}(M) = Sum\_Seq\_Ack(M(Ch\_C\_S)) + Sum\_Seq\_Ack(M(Ch\_S\_C)) \tag{5}$$

where the function Sum_Seq_Ack takes the multiset of packets in a channel and returns the summation of sequence and acknowledgement numbers of every packet. However, there are still two problems. The first is that when an entity in an idle state receives a packet with no acknowledgement number, the sequence number of the Reset packet (sent in reply) is set to zero and its acknowledgement number to the sequence number of the packet received. Thus the markings before and after this action have the *same* progress value. To overcome this problem, when computing Sum_Seq_Ack, we consider that the packets with no acknowledgement number have an acknowledgement number = -1. The second problem is that when sequence numbers wrap $\varphi_{ch\_sum}(M)$ will decrease and degrade the performance of the sweep-line. Thus we add 2^{48} to a sequence or acknowledgement number if it is less than ISS, as was the case with $Proj_{gss}$.

φ_{ch_num} has higher significance than φ_{ch_sum} because the effect of φ_{ch_sum} is only important when φ_{ch_num} is constant. Both φ_{ch_num} and φ_{ch_sum} have lower significance than other progress mappings because they are only effective when an entity is in an idle state.

In conclusion, we have identified several sources of progress of the specification model. The progress vector φ_s when considering only GSS has 7 dimensions, where

$$\varphi_s(M) = [\varphi_{cmd}(M), \varphi^c_{state}(M), \varphi^s_{state}(M), \varphi^c_{gss}(M), \varphi^s_{gss}(M), \varphi_{ch\_num}(M), \varphi_{ch\_sum}(M)] \tag{6}$$

and the subscript "s" refers to the specification model.

Our experiments show that this progress vector is monotonic for all scenarios analysed in this paper, and swapping the order of the client and server progress mappings has no effect on the performance of the sweep-line. Moreover the performance is improved slightly by including progress mappings for the GSR and GAR variables in the vector. Thus the full progress vector φ_s is given by

$$\begin{aligned}\varphi_s(M) = [&\varphi_{cmd}(M), \varphi^c_{state}(M), \varphi^s_{state}(M), \varphi^c_{gss}(M), \varphi^s_{gss}(M), \varphi^s_{gsr}(M), \varphi^c_{gsr}(M),\\ &\varphi^c_{gar}(M), \varphi^s_{gar}(M), \varphi_{ch\_num}(M), \varphi_{ch\_sum}(M)]\end{aligned} \tag{7}$$

3.4 Searching for a better progress measure

Although the sweep-line helps us to analyze protocols for scenarios that could not be reached before, most results of protocol verification (see for example [5, 6, 9, 16]) show that the peak states stored are around 20-30% of the full state space. The best reduction is shown in [5] where the number of peak states is around 10% (i.e. a reduction in states stored of a factor of 10). The sweep-line was also applied to compare the service language with the protocol language (language inclusion) of DCCP connection management in [7]. It used the same progress measures as described in Section 3.2 and 3.3. However the reduction in peak states stored [7] is about a factor of 3 to 4. Because the state space grows very rapidly with respect to the number of retransmissions, the progress measures used in [7] allowed only a few additional scenarios to be analysed (that could not be analysed by conventional state space analysis). Further reduction is thus required to verify DCCP's connection management behaviour.

We learnt from [5, 7] that although the state variables (states and sequence number sent) in the entities seem to be intuitively good progress measures, there are three fundamental problems. Firstly, more than 90% of the state space has either the server or the client in an idle state. Secondly, in these idle states there are no sequence number state variables. Thirdly, sequence numbers can wrap, leading to regress rather than progress. In other words, the efficiency of the sweep-line largely depends on the progress we can capture from the channel places. Channel places are not a good source of progress, partly because of the overtaking property of the channels. This explains why we only get a factor of 3 to 4 reduction in peak states stored.

From experiments we have noticed that the initial sequence number received (ISR) plays a crucial role in progress. The S_ISR is the first sequence number the server receives (in a Request packet). The client's ISR is in the first Response packet the client receives before moving to PARTOPEN. If the model is modified to store the values of ISR in the places Server_State and Client_State throughout the connection, including when the entity is in TIMEWAIT and CLOSED_F, we find something interesting. For example, when the initial sequence numbers sent (ISS) by both the client and the server are equal to five and one retransmission is allowed for the Request packet, the state space (total 19,602 nodes) is roughly divided into two large groups: group A, 11,930 nodes with S_ISR = 5 and group B, 7,612 nodes with S_ISR = 6; and one small group of 60 nodes without S_ISR. The size of group A is 60.86% of the total state space and the size of the group B is 38.83% of the total state space. Using only S_ISR as the progress measure, the sweep-line will finish working on group A before it starts working on group B. In this case the peak states stored can be reduced to 60.91% (11,939 nodes), where some states from group B are generated (but not explored) while exploring the states of group A.

Using [S_ISR, C_ISR] as the progress measure, group A is divided again into another two large groups (6,686 nodes with C_ISR = 5 and 4,684 nodes with C_ISR = 6) and one small group (560 nodes) with no C_ISR. However because there is no loss and the server cannot retransmit the Respond packet, group B is divided into one large group with C_ISR = 5 (7,158 nodes) and a small group (454 nodes) with no C_ISR. The peak states stored is further reduced to 36.52% (7,158 nodes). The clue is that every new progress measure added reduces the number of states that have the same progress vector.

We can continue to successively divide each group again by recording the sequence numbers of the Ack packets (S_IACK, C_IACK) that cause the server and the client to enter OPEN. Unfortunately, while S_ISR and C_ISR are the parameters of the specification model, S_IACK and C_IACK are not. By adding these two parameters, the specification model is modified to produce a different model, raising two issues. Firstly, does the new model (denoted the *augmented* model) have the same behaviour as the original specification model? Because the added parameters are solely used to measure progress and are not used in any protocol operations, both models exhibit the same behaviour. Secondly, the added parameters increase (explode) not only the total number of states but also the amount of memory used by each node in the state space. This is very harmful for state space analysis. Although the total number of states increases, the number of states in each subset of the partition decreases. During a sweep through one subset, the progress measure described in Section 3.2 can be used to further subdivide it. Thus not only is there a reduction in the peak number of states stored but also the potential for a reduction in execution time due to less time being spent comparing newly generated states with the states currently in memory (due to a reduced number of stored states).

3.5 Progress mappings for the augmented model

We record the latest sequence number of the packets sent for *each* packet type, we can exploit the progress from sequence numbers of Sync and Reset packets and further divide the state space to an even finer grain.

Figure 8 shows the modification to the declarations of the state variables. A new colour set called SNV (Sequence Number Vector) is defined in place of ISN. This records the additional progress variables: the latest sequence number of the packet sent for each packet type, and ISR. The colour set SNV is attached to every state, including idle states and REQUEST.

```
(* Modified DCCP state variables *)
color SN = IntInf;
color RCNT = int;(*Retransmission Counter*)
color GS = record GSS:SN48 * GSR:SN48 * GAR:SN48;
color SNV = record ISS:SN48 * SNReq_Resp:SN * ISR:SN48 * SNData_L:SN
           * SNAck_L:SN * SNCloseReq:SN * SNClose:SN * SNSync:SN * SNReset:SN;
color IDLE = with CLOSED_I | LISTEN | TIMEWAIT | CLOSED_F;
color IDLE_STATE = product IDLE*SNV;
color RCNTxGSSxSNV = product RCNT*SN48*SNV; (* counter,gss,SNV *)
color ACTIVE = with RESPOND | PARTOPEN | S_OPEN | C_OPEN | CLOSEREQ | C_CLOSING |S_CLOSING;
color ACTIVExRCNTxGSxSNV = product ACTIVE*RCNT*GS*SNV;
color CB = union IdleState:IDLE_STATE
               + ReqState:RCNTxGSSxSNV
               + ActiveState:ACTIVExRCNTxGSxSNV;
```

Fig. 8. DCCP's control block.

Using progress functions similar to those defined in Section 3.2, we can extract an additional 14 dimension progress vector (Equation 8) to measure progress in the augmented model. The ordering of progress values is arranged according to the point when each variable first appears in the typical scenario shown in Fig. 2 (connection setup followed by closing).

$$\begin{aligned}\varphi_a(M) = [&\varphi^c_{SNReq\_Resp}(M), \varphi^s_{ISR}(M), \varphi^s_{SNReq\_Resp}(M), \varphi^c_{ISR}(M), \varphi^c_{SNAck\_L}(M),\\ &\varphi^s_{SNAck\_L}(M), \varphi^s_{SNData\_L}(M), \varphi^s_{SNCloseReq}(M), \varphi^c_{SNClose}(M), \varphi^s_{SNClose}(M),\\ &\varphi^s_{SNSync}(M), \varphi^c_{SNSync}(M), \varphi^s_{SNReset}(M), \varphi^c_{SNReset}(M)]\end{aligned} \tag{8}$$

Thus the overall progress vector for the augmented model is:

$$\varphi_{DCCP}(M) = [\varphi_a(M)\ ,\ \varphi_s(M)] \tag{9}$$

Although the analysis result using the proposed progress measures are very promising, there is still one drawback. The number of terminal markings increases rapidly with respect to the

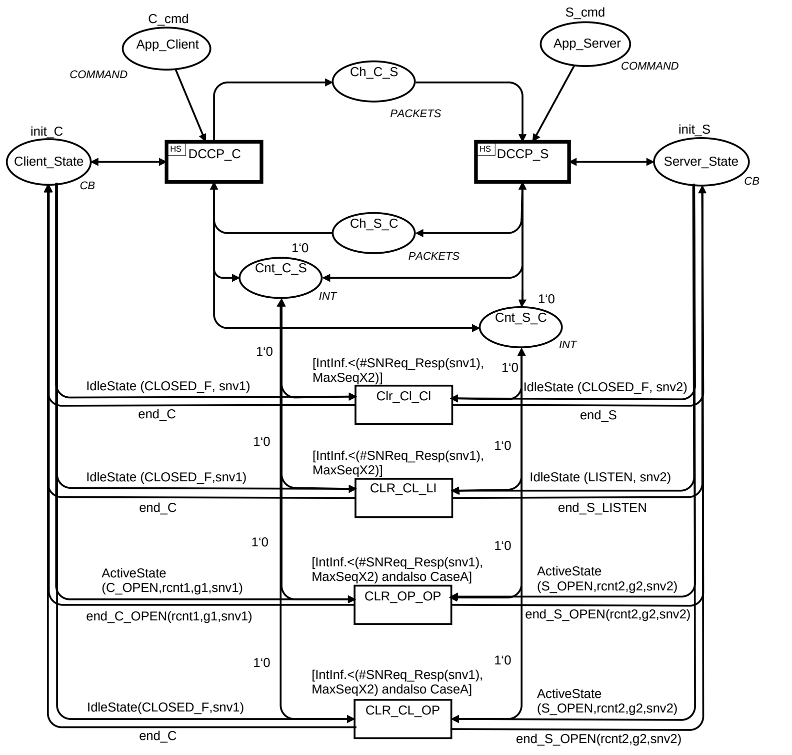

C_cmd
App_Client
COMMAND
S_cmd
App_Server
COMMAND
Ch_C_S
PACKETS
init_C
Client_State
CB
HS DCCP_C
HS DCCP_S
init_S
Server_State
CB
Ch_S_C
PACKETS
1'0
Cnt_C_S
INT
1'0
Cnt_S_C
INT
[IntInf.<(#SNReq_Resp(snv1), MaxSeqX2)]
IdleState (CLOSED_F, snv1)
Clr_Cl_Cl
IdleState (CLOSED_F, snv2)
end_C
end_S
[IntInf.<(#SNReq_Resp(snv1), MaxSeqX2)]
IdleState (CLOSED_F,snv1)
CLR_CL_LI
IdleState (LISTEN, snv2)
end_C
end_S_LISTEN
[IntInf.<(#SNReq_Resp(snv1), MaxSeqX2) andalso CaseA]
ActiveState (C_OPEN,rcnt1,g1,snv1)
CLR_OP_OP
ActiveState (S_OPEN,rcnt2,g2,snv2)
end_C_OPEN(rcnt1,g1,snv1)
end_S_OPEN(rcnt2,g2,snv2)
[IntInf.<(#SNReq_Resp(snv1), MaxSeqX2) andalso CaseA]
IdleState(CLOSED_F,snv1)
CLR_CL_OP
ActiveState (S_OPEN,rcnt2,g2,snv2)
end_C
end_S_OPEN(rcnt2,g2,snv2)

Fig. 9. Augmented model: the top level page.

number of new variables added. To avoid this drawback we introduce four transitions and two places as shown in Fig. 9, to merge redundant terminal markings into the terminal markings that exist in the original state space. The places Cnt_C_S and Cnt_S_C are used to count the number of packets in channels. The clean up operation takes place when there is nothing left in both channels.

4 Experimental results

4.1 Initial configurations

We analyse two DCCP connection management models using Design/CPN version 4.0.5 with the occurrence graph tool and the prototype sweep-line library from [5] able to handle progress vectors, on a Pentium-IV 2.6 GHz computer with 1GB RAM. The first model is the specification model as described in Section 2. The second model is the augmented model as described in Section 3.5. The DCCP-CM models are initialised by distributing tokens to places App_Client, App_Server, Client_State and Server_State of the model to create the initial markings. Table 2 shows the initial markings for the Application places for both the client and server. We present the analysis results of three cases. In all cases, the client and server are initially in CLOSED_I, both channel places are empty and ISS values on both sides are set to $2^{48} - 3$ to allow sequence numbers to wrap to zero.

Case A is for connection establishment. The client issues an "active Open" command while the server issues a "passive Open" command. When limiting the maximum number of retransmissions to one, connection establishment can fail due to a deadlock when sequence

Table 2. Initial configurations.

	Initial Markings	
Case	App_Client	App_Server
A	1‘active open	1‘passive open
B	1‘active open	1‘passive open ++ 1‘active close
C	1‘active open	1‘passive open ++ 1‘server active close

numbers wrap [24]. In this paper we use the sweep-line to extend the analysis to include a scenario when the maximum number of retransmissions is two to determine if the undesired deadlocks still exist. Case B and C cover the case when the server issues a close command while the connection is being established. We select configurations B and C to determine if the application on the server can clear the deadlocks by issuing either an “active close” or a “server active close” command.

4.2 Analysis results

The analysis results for the DCCP connection management CPN specification and augmented models in various configurations are shown in Tables 3 and 4. All progress measures used are monotonic. The first column in Table 3 shows the configurations being analysed, where the 4-tuple is the maximum number of retransmissions allowed for Request, Ack, CloseReq and Close packet types respectively. An “x” means the retransmission of those packet types never happens in that configuration. Columns 2-3 show the analysis results of the specification model when using a constant progress value (Sweep-Line$_C$) which simulates conventional reachability analysis. Columns 4-6 show the results of the specification model using the progress measure φ_s described in Section 3.2 (Sweep-Line$_S$). Columns 7-9 show the result of the augmented model using the progress measure φ_{DCCP} described in Section 3.5 (Sweep-Line$_A$). Comparison of space[2] and time[3] used between Sweep-Line$_S$ and Sweep-Line$_C$ are shown in columns 10-11. Comparison of space and time used between Sweep-Line$_A$ and Sweep-Line$_C$ are shown in columns 12-13. A “-” means the full state space cannot be generated due to computer memory limits. Table 3 shows *five* cases where Sweep-Line$_S$ cannot generate the state space.

Reduction of the peak number of stored states While Sweep-Line$_S$ reduces the peak number of stored states to about 30-40%, Sweep-Line$_A$ gives a more promising result. The larger the state spaces are, the greater the reduction. In most cases the reduction is better than a factor of 10 (10%). The best result shown in Table 3 is 0.36% (277 times smaller than the original state space). However, it is not immediately clear how much this translates to a reduction in real memory usage, because each state in the augmented model stores slightly more information. However, Sweep-Line$_A$ can finish the exploration in some configurations, such as A-(2,2,x,x) and A-(2,3,x,x), while Sweep-Line$_S$ and Sweep-Line$_C$ cannot. This leads us to believe that, pragmatically, Sweep-Line$_A$ also provides a significant reduction in memory usage when the state spaces are large.

Execution time comparison Generally Sweep-Line$_S$ and Sweep-Line$_A$ have longer exploration times than Sweep-Line$_C$ because of the overhead computing the progress mappings

[2] (the number of peak states/the total number of states in the original state space)*100
[3] (the exploration time using progress vector φ_s /the exploration time using constant progress value)*100

Table 3. Sweep-line analysis results of DCCP connection management.

Config.	Sweep-Line$_C$ constant progress value		Sweep-line$_S$ specification model			Sweep-line$_A$ augmented model			(S/C)*100		(A/C)*100	
	nodes	hh:mm:ss	total nodes	peak nodes	hh:mm:ss	total nodes	peak nodes	hh:mm:ss	% space	% time	% space	% time
A-(0,0,x,x)	102	00:00:00	102	34	00:00:00	140	15	00:00:00	33.33	-	14.71	-
A-(0,1,x,x)	209	00:00:00	209	71	00:00:00	314	34	00:00:00	33.97	-	16.27	-
A-(0,2,x,x)	602	00:00:00	602	218	00:00:00	983	119	00:00:01	36.21	-	19.77	-
A-(1,0,x,x)	2,397	00:00:01	2,397	918	00:00:02	4,870	87	00:00:03	38.30	200.00	3.63	300.00
A-(1,1,x,x)	11,870	00:00:08	11,870	4,435	00:00:10	29,212	288	00:00:23	37.36	125.00	2.43	287.50
A-(1,2,x,x)	61,239	00:01:05	61,239	24,289	00:01:22	172,307	1,096	00:02:46	39.66	126.15	1.79	255.38
A-(2,0,x,x)	116,745	00:03:03	116,745	42,486	00:03:24	362,528	1,263	00:05:09	36.39	111.48	1.08	168.85
A-(1,3,x,x)	296,961	00:10:35	296,961	123,463	00:12:51	934,049	4,167	00:17:58	41.58	121.42	1.40	169.76
A-(2,1,x,x)	964,862	01:41:29	964,862	354,710	01:59:47	3,970,455	6,142	01:09:03	36.76	118.03	0.64	68.04
A-(2,2,x,x)	-	-	-	-	-	31,872,051	34,059	11:46:59	-	-	-	-
A-(2,3,x,x)	-	-	-	-	-	219,200,989	161,461	120:07:23	-	-	-	-
B-(0,0,x,0)	476	00:00:00	476	153	00:00:00	745	45	00:00:00	32.14	-	9.45	-
B-(0,0,x,1)	1,505	00:00:01	1,505	515	00:00:01	2,875	129	00:00:02	34.22	100.00	8.57	200.00
B-(0,1,x,0)	1,636	00:00:01	1,636	532	00:00:01	2,962	152	00:00:02	32.52	100.00	9.29	200.00
B-(0,1,x,1)	7,479	00:00:05	7,479	2,874	00:00:06	15,580	518	00:00:14	38.43	120.00	6.93	280.00
B-(1,0,x,0)	24,927	00:00:20	24,927	9,093	00:00:23	72,380	394	00:00:57	36.48	115.00	1.58	285.00
B-(1,0,x,1)	203,168	00:04:38	203,168	74,815	00:05:33	723,963	2,743	00:11:43	36.82	119.78	1.35	252.88
B-(1,1,x,0)	218,951	00:06:27	218,951	77,131	00:06:59	876,163	1,559	00:13:45	35.23	108.27	0.71	213.18
B-(2,0,x,0)	-	-	-	-	-	12,568,700	9,154	03:39:20	-	-	-	-
B-(1,1,x,1)	-	-	-	-	-	13,241,057	21,479	04:17:28	-	-	-	-
C-(0,0,0,0)	666	00:00:00	666	235	00:00:00	1,089	45	00:00:01	35.29	-	6.76	-
C-(0,0,0,1)	1,245	00:00:01	1,245	449	00:00:01	2,156	92	00:00:02	36.06	100.00	7.39	200.00
C-(0,1,0,0)	3,270	00:00:02	3,270	1,148	00:00:02	6,244	146	00:00:05	35.11	100.00	4.46	250.00
C-(0,1,0,1)	9,080	00:00:06	9,080	3,321	00:00:08	20,150	430	00:00:18	36.57	133.33	4.74	300.00
C-(0,0,1,0)	8,890	00:00:05	8,890	3,550	00:00:07	17,536	233	00:00:14	39.93	140.00	2.62	280.00
C-(1,0,0,0)	45,368	00:00:40	45,368	17,214	00:00:46	159,818	394	00:02:05	37.94	115.00	0.87	312.50
C-(0,1,1,0)	79,320	00:01:10	79,320	30,774	00:01:30	169,728	1,341	00:02:44	38.80	128.57	1.69	234.29
C-(0,0,1,1)	127,195	00:02:03	127,195	49,737	00:02:40	289,062	2,573	00:04:54	39.10	130.08	2.02	239.02
C-(1,0,0,1)	305,807	00:08:12	305,807	110,955	00:08:48	1,441,029	2,798	00:24:25	36.28	107.32	0.91	297.76
C-(1,1,0,0)	477,764	00:19:45	477,764	175,913	00:21:10	2,058,949	1,727	00:33:52	36.82	107.17	0.36	171.48
C-(1,0,1,0)	-	-	1,493,946	569,749	03:16:27	8,141,588	6,719	02:26:25	-	-	-	-
C-(1,1,0,1)	-	-	-	-	-	17,594,060	18,606	05:50:50	-	-	-	-

Table 4. Terminal markings.

Config.	Terminal Markings			
	Type-I	Type-II	Type-III	Type-IV
A-(0,0,x,x)	2	1	1	0
A-(1,0,x,x)	15	1	1	1
A-(2,0,x,x)	83	1	1	8
A-(0,1,x,x)	5	1	1	0
A-(0,2,x,x)	9	1	1	0
A-(1,1,x,x)	38	1	1	2
A-(1,2,x,x)	68	1	1	3
A-(1,3,x,x)	105	1	1	4
A-(2,1,x,x)	198	1	1	15
A-(2,2,x,x)	343	1	1	22
A-(2,3,x,x)	519	1	1	29

(11 functions for Sweep-Line$_S$ and 25 functions for Sweep-Line$_A$). When the original state space is small, Sweep-Line$_A$ has the longest exploration time because the state space of the augmented model is bigger and Sweep-Line$_A$ has more progress mappings to calculate. As the state space grows bigger, Sweep-Line$_A$ gradually becomes more efficient. When the size of the original state space is large, for instance in configuration A-(2,1,x,x), Sweep-Line$_A$ is faster than Sweep-Line$_S$ even though the augmented model has a bigger state space. This is because Sweep-Line$_A$ spends less time comparing new states to existing states (due to storing fewer states in memory at any one time).

4.3 Terminal marking classification

Terminal markings (dead markings) are states from which no action can occur. Undesired terminal markings are called deadlocks. Table 4 shows the terminal markings of Configuration A. All terminal markings have no packets left in the channels and hence there are no unspecified receptions. The terminal markings are classified into 4 types. Type-I terminal markings arise when both the client and server are in the OPEN state indicating that the connection has been successfully established. Terminal markings Types II and III arise in situations when the connection attempt fails because the back-off timer[4] expires. Both types of terminal markings are acceptable. In Type II terminal markings both sides are CLOSED. For Type III terminal markings, the client is CLOSED, but the server is in the LISTEN state. This can happen when the server is initially CLOSED (down for maintenance or busy) and rejects the connection request. The server then recovers and moves to the LISTEN state while the client finishes in CLOSED on receipt of the reset. These three types of terminal markings are expected. However, Type IV terminal markings are undesired deadlocks where the client is CLOSED but the server stays in OPEN. Hence, allowing up to two retransmissions of each packet type has not eliminated the deadlocks for the case where ISS is $2^{48} - 3$.

Every scenario of configuration B and C has two Type-II and one Type-III terminal marking. One terminal marking of Type-II has a close command left in the place App_Server while the other has no token left in this place. There are no Type IV terminal markings in configurations B and C. This shows that these deadlocks can be overcome by the server closing the connection.

These experiments were conducted initially for an ISS of $2^{48} - 3$. It is infeasible to generate the state space for every one of the 2^{48} values of ISS (0 to $2^{48} - 1$), however, the initial experiments have been followed up with further experiments for other ISS values that cause sequence numbers to wrap. All give the similar deadlock results. Thus we conjecture that when sequence numbers wrap: 1) when the maximum number of retransmissions is two, connection establishment still has an undesired deadlock; and 2) when the maximum number of retransmissions is one, the deadlocks do not occur when the application on the server issues a close command (either "active close" or "server active close").

5 Conclusions and future work

This paper presents the sweep-line analysis of DCCP connection management CPN models operating over reordering channels with no loss. We give some insight into how to determine

[4] After retrying for a period (measured by a "back-off" timer), the client will send a DCCP-Reset and will "back off" to the CLOSED state [14].

sources of progress for our DCCP-CM CPN model. We believe this approach could be applied to other transport protocols. While the main stream approaches try to reduce the size of the state space, we explode it by augmenting the model with additional state information in such a way that the exploded state space has a structure which is easier for the sweep-line to explore. This gives the very promising result of significant reduction in both peak states stored and exploration time when analysing large state spaces. The efficiency of this method increases as the original state space gets bigger.

We also successfully extended the analysis of DCCP connection management to *five* configurations (see Table 3) that were previously out of reach. In this paper we confirmed for an ISS value of $2^{48}-3$ that when the maximum number of retransmissions is two, that connection establishment still has an undesired deadlock. We also confirmed for this ISS that when the maximum number of retransmissions is one, the deadlocks do not occur when the application on the server issues a close command (either "active close" or "server active close"). It is infeasible to perform this kind of analysis for all values of ISS, however our experiments so far suggest that this behaviour holds for values of ISS where sequence numbers wrap.

The work presented in this paper provides some evidence that it may be possible to exploit the full potential of the sweep-line method. We hope it will be possible in the future to craft progress mappings that will only need several thousand states to be stored in memory, even when the state space is huge (i.e. greater than 10^9 states), and so make the sweep-line a practical verification technique. The development of a structured approach to obtain such progress mappings and to perform the necessary model transformations is an open research question.

6 Acknowledgments

The authors are grateful to the anonymous reviewers whose comments have helped to improve the quality of the paper, and who have provided some interesting challenges for future work.

References

1. J. Billington, G.E. Gallasch, L.M. Kristensen, and T. Mailund. Exploiting Equivalence Reduction and the Sweep-line Method for Detecting Terminal States. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):23–37, January 2004.
2. S. Christensen, L.M. Kristensen, and T. Mailund. A Sweep-Line Method for State Space Exploration. In *Proceedings of TACAS 2001*, volume 2031 of *Lecture Notes in Computer Science*, pages 450–464. Springer-Verlag, 2001.
3. CPN ML: An Extension of Standard ML. `http://www.daimi.au.dk/designCPN/sml/cpnml.html`.
4. E.A. Emerson and A.P. Sistla. Symmetry and Model Checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.
5. G. E. Gallasch, B. Han, and J. Billington. Sweep-line Analysis of TCP Connection Management. In *Proceedings of ICFEM'05*, volume 3785 of *Lecture Notes in Computer Science*, pages 156–172. Springer-Verlag, 2005.
6. G.E. Gallasch, C. Ouyang, J. Billington, and L.M. Kristensen. Experimenting with Progress Mappings for the Sweep-Line Analysis of the Internet Open Trading Protocol. In *Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, DAIMI PB 570, pages 19–38. Department of Computer Science, University of Aarhus, October 8-11, 2004. Available via http://www.daimi.au.dk/CPnets/workshop04/cpn/papers/.
7. G.E. Gallasch, S. Vanit-Anunchai, J. Billington, and L.M. Kristensen. Checking Language Inclusion On-The-Fly with the Sweep-line Method. In *Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, DAIMI PB 576, pages 1–20. Department of Computer Science, University of Aarhus, October 24-26, 2005. Available via http://www.daimi.au.dk/CPnets/workshop05/cpn/papers/.

8. P. Godefroid, G.J. Holzmann, and D. Pirottin. State-Space Caching Revisited. *Formal Methods in System Design*, 7(3):227–241, 1995.
9. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In *Proceedings of ICATPN'02*, volume 2360 of *Lecture Notes in Computer Science*, pages 182–202. Springer-Verlag, 2002.
10. G.J. Holzmann. Algorithms for Automated Protocol Validation. *AT&T Technical Journal*, 69(2):32–44, 1990.
11. G.J. Holzmann. An Analysis of Bitstate Hashing. *Formal Methods in System Design*, 13(3):287–305, 1998.
12. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. Springer-Verlag, 1992.
13. K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9(1/2):7–40, 1996.
14. E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol, RFC 4340. Available via , http://www.rfc-editor.org/rfc/rfc4340.txt, March 2006.
15. L.M. Kristensen and T. Mailund. A Generalised Sweep-Line Method for Safety Properties. In *Proceedings of FME'02*, volume 2391 of *Lecture Notes in Computer Science*, pages 549–567. Springer-Verlag, 2002.
16. L.M. Kristensen and T. Mailund. Efficient Path Finding with the Sweep-Line Method using External Storage. In *Proceedings of ICFEM'03*, volume 2885 of *Lecture Notes in Computer Science*, pages 319–337. Springer-Verlag, 2003.
17. 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.
18. R. Milner, R. Harper, and M. Tofte. *The Definition of Standard ML*. MIT Press, 1990.
19. Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.
20. A.N. Parashkevov and J. Yantchev. Space Efficient Reachability Analysis Through Use of Pseudo-Root States. In *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 50–64. Springer-Verlag, 1997.
21. D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.
22. A. Valmari. A Stubborn Attack on State Explosion. In *Proceedings of CAV'90*, volume 531 of *Lecture Notes in Computer Science*, pages 156–165. Springer-Verlag, 1990.
23. A. Valmari. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.
24. S. Vanit-Anunchai and J. Billington. Effect of Sequence Number Wrap on DCCP Connection Establishment. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 345–354, Monterey, California, USA, September 11-13, 2006. IEEE Computer Society Press.
25. S. Vanit-Anunchai, J. Billington, and T. Kongprakaiwoot. Discovering Chatter and Incompleteness in the Datagram Congestion Control Protocol. In *Proceedings of FORTE'05*, volume 3731 of *Lecture Notes in Computer Science*, pages 143–158. Springer-Verlag, 2005.
26. P. Wolper and P. Godefroid. Partial Order Methods for Temporal Verification. In *Proceedings of CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246. Springer-Verlag, 1993.
27. P. Wolper and D. Leroy. Reliable Hashing without Collision Detection. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 59–70. Springer-Verlag, 1993.

A Coloured Petri Net Model of Cooperative Arrival Planning in Air Traffic Control

Hendrik Oberheid

German Aerospace Center
Institute of Flight Guidance, Braunschweig, Germany
hendrik.oberheid@dlr.de

Abstract: A Coloured Petri Net model is presented which simulates a potential future arrival planning process in air traffic control. The model is focused on the sequence planning step of the cooperative arrival planning procedure and was built in order to acquire a better understanding of how the behaviour of individual actors (i.e. aircraft) within the cooperation influences the outcome of the overall sequence planning process. A peculiarity of the CP-net from a modelling point of view lies in the fact that state space analysis is used repeatedly during each cycle of the planning process to generate and evaluate the potential solutions to the sequence planning problem. The results gained through queries on the state space are then re-fed into the simulation of the next planning cycle. Within the scope of the paper different analysis and visualisation approaches are discussed which can be used to trace the time-dependent system behaviour over one single planning cycle or a number of consecutive cycles. The results of the simulation shall in future be used to build realistic scenarios and assumptions on how different actors will interact with the system from a human factors point of view.

1 Introduction

One of the most challenging tasks of air traffic control consists in managing the arrival traffic at a highly frequented airport. For an increasing number of large airports this task is therefore supported by automated controller assistance tools, so called AMANs (Arrival Managers). Depending on the capabilities of the specific AMAN the tool will nowadays support the human controller in one or both of the following tasks: *1. sequencing*, i.e. establishing a favourable arrival sequence (sequence of aircraft) according to a number of optimisation criteria *2. metering* that is to calculate for each individual aircraft the target times over (TTO) certain points (fixes) of the respective arrival route appropriate for the aircraft‘s position in the sequence.

Apart from sequencing and metering future AMANs will additionally be able to support the controller in finding an efficient and conflict free route for each aircraft from its current position to the runway threshold *(trajectory generation)* [2, 12]. The AMAN could also provide detailed recommendations to the controller with regard to individual clearances[1] he should give to an aircraft in order to lead the aircraft along that route as planned *(clearance generation)*. A simplified view of the arrival planning process including the main processes and information flow is illustrated in Figure 1[2].

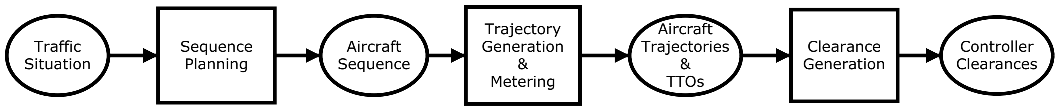


Figure 1: Simplified view of arrival planning process

A critical determinant for the quality of the arrival planning process is generally the type and accuracy of information on which the planning is based. Most AMANs currently in operation rely on a combination of radar data and flight plan information as their main input to the planning process. As an enhancement to that the DLR Institute of Flight Guidance is developing an Arrival Manager named 4D-CARMA (4D Cooperative Arrival Manager) [1] which makes use of data received directly via datalink from individual aircraft to feed the planning process. This data includes components such as estimated earliest and latest times over certain fixes as well as information on the planned descent profile which are computed by the aircraft‘s own flight management system.

The tighter cooperation of airborne and groundside envisioned with 4D-CARMA and the higher accuracy of data received directly from the airborne side is expected to increase both the capacity of the airspace and the economic efficiency (e.g. lower fuel consumption) of individual flights. It is also expected that greater flexibility in the planning process will be reached and chances will improve to deliver *user-preferred trajectories* (i.e. “aircraft/airline-preferred trajectories”) and accommodate airlines wishes with regard to the prioritisation of specific flights in the sequence. However, as the enhanced system will base its planning on information received from external actors it is important to develop a thorough understanding on how each piece of information submitted affects the overall system behaviour. Not only is it necessary to consider the consequences on the overall outcome arising from submitting certain data, it is also essential to consider the incentives the system provides for each individual actor to fully comprehend the role and behaviour of the actor while interacting with the system. The CPN model of cooperative arrival planning presented in this paper and the analysis and visualisation functions developed in this context shall contribute to acquire such an understanding.

[1]A *clearance* is an authorization or permission given by air traffic control for an aircraft to proceed along a certain route segment or to execute some other action specified in the body of the clearance
[2]Note that in practice a number of dependencies exist between those stages and feedback loops may have to be added between the different components which are not considered in this figure

2 Sequence Planning

The CPN model introduced in this paper is focused on the sequence planning aspects of arrival management. The sequence planning phase is the natural starting point for the modelling of an AMANs behaviour, as it precedes the other planning phases (trajectory generation and metering) which base their planning on the assumed sequence (see Figure 1). Important decisions about the outcome of the overall process (particularly in terms of punctuality, costs and fuel consumption) are made during the sequence planning stage when the position of an aircraft in the sequence is determined which can later only be altered under certain circumstances and with considerable effort. Besides these operational aspects, the sequence planning process is also the most straightforward to model and analyse in the form of a Coloured Petri Net since Sequence Planning is inherently a graph theoretic problem with discrete results (a sequence). By contrast the later stages of arrival management (i.e trajectory generation) contain comparably larger shares of continuous-time computations which may more adequately be described by other modelling approaches.

The task of sequence planning itself can be further divided into the following four subtasks, which form the different parts of the CPN model presented below (Figure 2):

- step 1: Sequence Generation
- step 2: Sequence Evaluation
- step 3: Sequence Selection
- step 4: Sequence Implementation

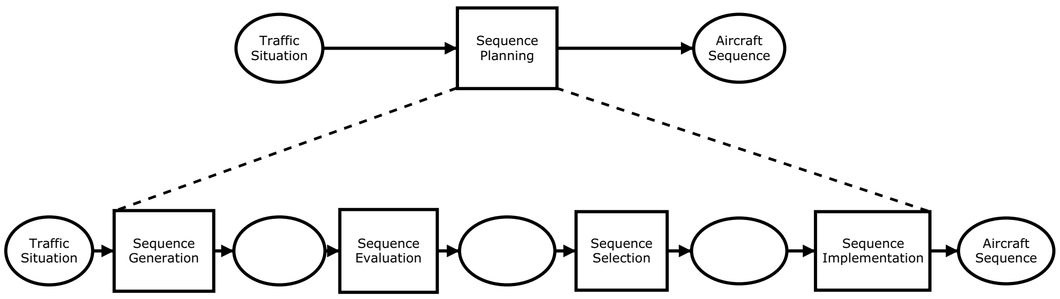


Figure 2: Four step scheme of sequence planning

The four subtasks will first be described with regard to the general problem statement of each step before the concrete modelling of the different planning stages in the form of a Coloured Petri Net model is presented in the following section.

2.1 Sequence Generation

The general function of the sequence generation step consists in creating a set of candidate sequences for a number of aircraft within a defined timeframe. Basically this is achieved by building all possible permutations of aircraft within a certain timeframe (using a “sliding window” technique) and then computing the respective arrival times for each aircraft of a candidate sequence while maintaining the required temporal separation between consecutive aircraft. Note however that for set of no_of_ac aircraft one will get

$$no\_of\_cseq = no\_of\_ac! \tag{1}$$

candidate sequences. This means 720 candidates for a moderate set of 6 aircraft and 40320 candidates for a operationally more relevant set of 8 aircraft. It is thus highly desirable to constrict the number of candidates by some intelligent search function and exclude the less favourable candidate sequences as soon as possible [9].

2.2 Sequence Evaluation

Given the set of candidate sequences the task of the evaluation step is to compute for each candidate a quality value which reflects the desirability/feasibility of practically realizing that candidate sequence from an operational point of view. In order to compute that quality value the evaluation step makes use of a number of evaluation functions which rate among other things (i) if an aircraft in the sequence is able to reach its submitted earliest time of arrival, (ii) if an aircraft is able to land before its submitted latest time of arrival and thus without the need for entering a holding, or (iii) if an aircraft would leave a potential holding pattern according to its relative position in the holding etc.. Another important aspect which is also rated by the evaluation step is if (iv) the “new” candidate sequence is stable with regard to the “old” sequence which has been implemented in the preceding planning cycle. In the ideal case the old and the new sequence are identical, meaning that all aircraft maintain their exact position in the sequence and their target time of arrival (TTA). If in turn the TTA of an aircraft changed according to the candidate sequence this would be rated negatively as it would generally mean additional workload for the controller and pilot to rearrange the sequence and might also hamper flight efficiency. The operational requirement for a stability criterion and the associated need to base the selection of a candidate sequence by some means or another on the results of the preceding planning cycle make the whole sequence planning a recursive process.

Mathematically this can be formulated as follows:

$$\begin{aligned} s_i &= f(x_i, s_{i-1}) \qquad (2)\\ s_i &= \text{Sequence implemented in step i}\\ x_i &= \text{Traffic situation and other planning inputs in step i}\\ s_{i-1} &= \text{Sequence implemented in step i-1} \end{aligned}$$

The recursive nature of the function adds significantly to the complexity of the planning process because it makes the consequences of modifying/adding certain rating functions or adjusting the weighting between these functions harder to predict. The recursion also makes it more difficult to foresee the effects of certain behaviour of individual actors (e.g. content of submitted data, estimates provided, time of data submission from airborne side) on the system behaviour. To be able to examine these effects in a controlled fashion is one of the main purposes of the CPN model introduced below.

2.3 Sequence Selection

Given the set of candidate sequences together with the ratings computed in the evaluation step, the task of the sequence selection step is to pick the one sequence which will finally be implemented in practice. It is unlikely (mainly for human factors reasons) that the selection task will be totally automated in the near future and that the selection will be executed without the chance for human intervention. This is because the final responsibility for the decision will for a long time remain with the human controller, thus the controller will also have to be in control. A more realistic scenario might therefore consist in an automated support system pre-selecting a limited set of favourable candidate sequences and then having the human controller take the final decision if/which one of the proposed candidates is selected for implementation.

2.4 Sequence Implementation

The practical implementation of the sequence is to a large degree a manual controller task. The controller gives instructions and clearances to the flightcrew with regard to the aircraft arrival routing, flight profile and speed. The cockpit crew acknowledges the clearances and configures the aircraft to follow the instructed flight path. Nowadays nearly all communication between air traffic control and cockpit crew is still carried out via voice radio. In future this verbal communication might more and more be substituted by a datalink connection.
As mentioned above when discussing the recursive nature of the planning process, as soon as a sequence is implemented it becomes an important input to for the next planning cycle executed thereafter.

3 The CPN model of Cooperative Sequence Planning

This section presents the CPN model developed to analyse the behaviour of the cooperative sequence planning process as outlined above. The entire model currently consists of 16 pages arranged in 5 hierarchical layers. While 15 of the 16 pages form a tree structure departing from the top level page *SequPlan*, the page *ProgressionGraph* stands independently on the same level as *SequPlan*. It serves to integrate the results gained on other pages. An overview of the page structure of the model is depicted in Figure 3. The specific function of the individual pages is explained below, starting from the toplevel page named *SequPlan* and then proceeding to the lower level pages of the tree structure. References are made for each page on how it relates to the four step scheme of sequence planning introduced in Figure 2.

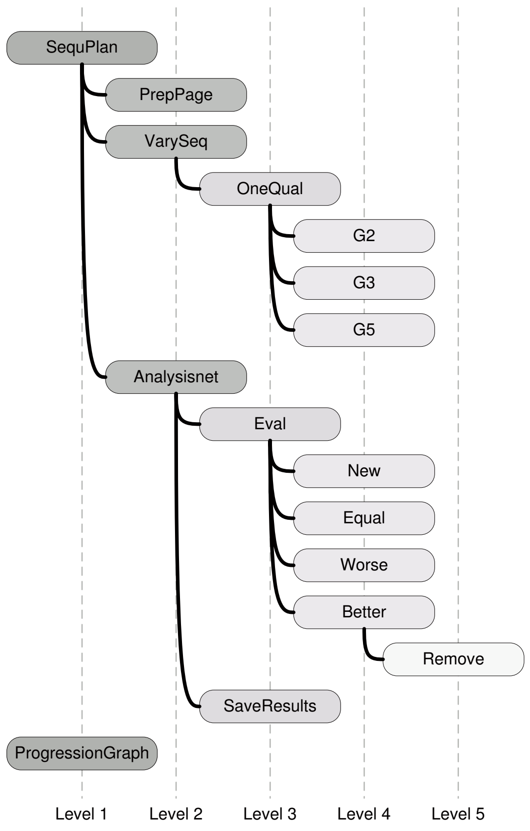


Figure 3: Page hierarchy of the CPN model

SequPlan page

The SequPlan page is depicted in Figure 4. It features three main substitution transitions named *Preparation*, *Sequence Generation & Evaluation* and *Sequence Selection* and one simple transition named *Sequence Implementation*. The three substitution transitions hide the different model parts which are needed in order to simulate one cycle of a sequence planning process. The simple transition Sequence Implementation loops back the results from each planning cycle, that is, it reinserts a sequence which is considered as implemented back to initialise the following planning cycle. As the SequPlan page is the toplevel page in the hierarchy, it holds together all four stages of the sequence planning process introduced in Figure 2.

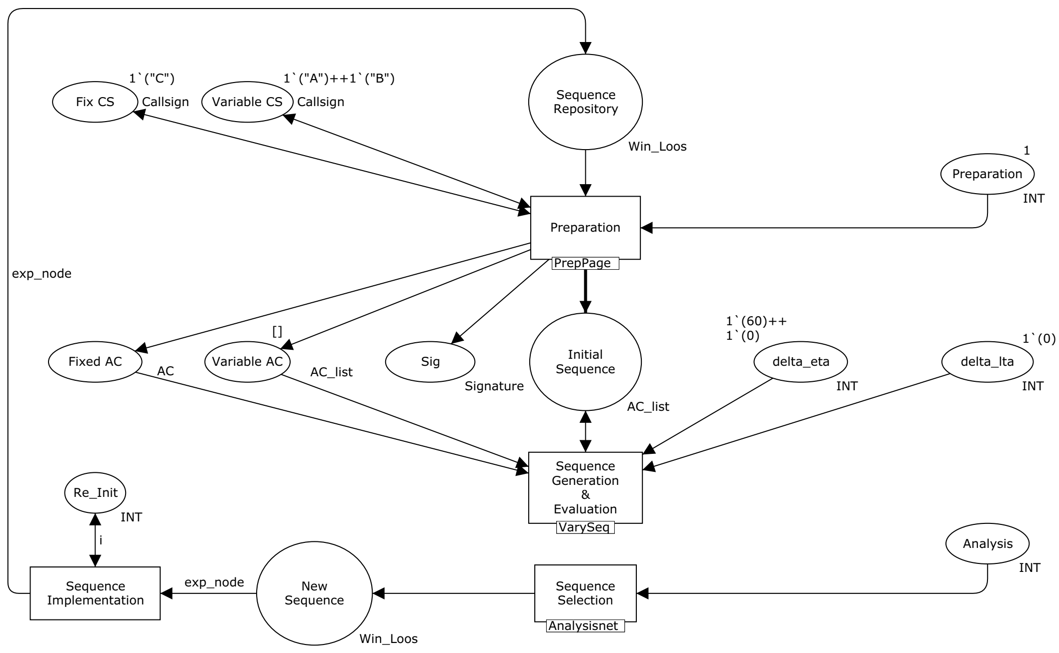


Figure 4: SequPlan page

PrepPage

With regard to the four stage model of sequence planning introduced in section 2 the function of the PrepPage is to prepare the input data for the first step *sequence generation* of the Sequence Planning.
The Subnet hidden by the substitution transition *PrepPage* is shown in Figure 5. Its first function is to pre-process the output of the preceding planning cycle (tokens on the Place *Sequence Repository)* and to extract from the complex colorset Win_Loos the pieces of information relevant for the next planning cycle. A second function of the PrepPage is to define the set of input data/traffic situations (potential evolvement of the traffic situation) considered in the next planning cycle. As mentioned above, one of the main purposes of the model is to examine how the sequence planning (including its individual rating functions) responds to certain inputs/behaviour of involved aircraft. In order to analyse these effects, for each step it has to be defined which aircraft will be free to alter their inputs before the next planning cycle and which aircraft have to keep their inputs constant. This decision is also modelled on the *PrepPage*. The aircraft which are free to vary their estimates are collected in a list on the place *Variable AC* while the aircraft which must keep their estimates unchanged are represented as a multiset of tokens on the place *Fixed AC*. The concrecte range of possible variations of input data (in the current state of the model the *Earliest Time of Arrival* (ETA) and *Latest Time of Arrival* (LTA)) is specified directly through tokens on the places *delta_ eta* and *delta_ lta* on the *SequPlan* page (see Figure 4).

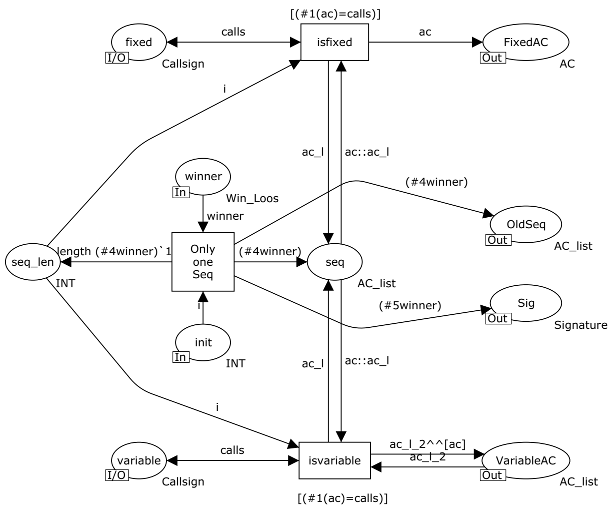


Figure 5: PrepPage

VarySeq page

The function of the *VarySeq page* is threefold:

1. It computes the different *input configurations* which describe the potential evolvement of traffic situation on the basis of the list of *VariableAC* and the tokens on the places *delta_eta, delta_lta*. Each input configuration defines one potential traffic situation by specifying the ETAs and LTAs of all involved aircraft at a certain point in time. The number of configurations results from equation 3:

$$\begin{aligned} no\_of\_config &= no\_var\_ac * tok\_delta\_eta * tok\_delta\_lta \qquad (3) \\ no\_var\_AC &= \text{Number of variable aircraft} \\ tok\_delta\_eta &= \text{Number of tokens on place } delta\_eta \\ tok\_delta\_lta &= \text{Number of tokens on place } delta\_lta \end{aligned}$$

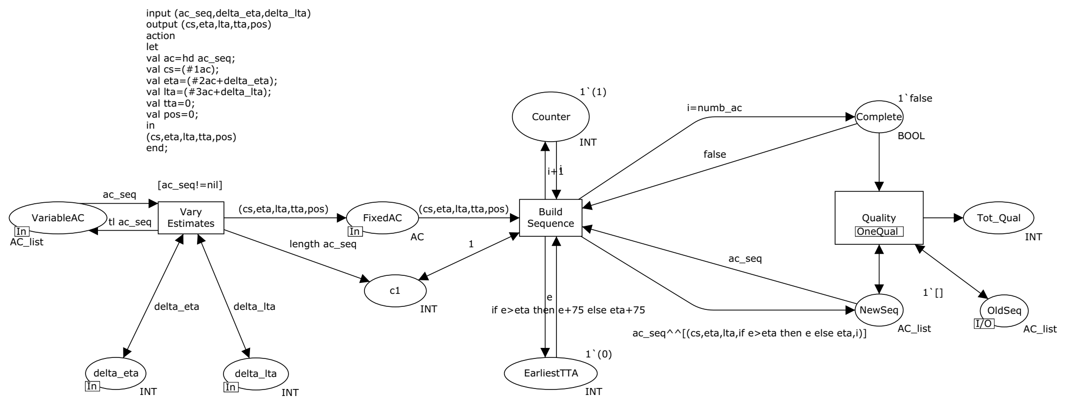
   The functionality is realised by the transition Vary Estimates (see Figure 6) and is still considered as a part of the data preparation which takes place before the execution of a planning cycle.

2. It generates the full set of candidate sequences by computing all different permutations for a set of aircraft assuming the different configurations. The number of candidate sequences is

$$no\_of\_cseq = no\_of\_config * no\_of\_ac! \qquad (4)$$

   This action is realised in the net by the transition *BuildSequence* and corresponds to the sequence generation step in Figure 2.

3. It calls for each candidate sequence the evaluation module (page *OneQual*) which subsequently calculates the quality values which will later decide about the implementation or dismissal of the candidate sequence. This is realised by the substitution transition *OneQual* (see Figure 6) and corresponds to the sequence evaluation stage of Figure 2.



```
input (ac_seq,delta_eta,delta_lta)
output (cs,eta,lta,tta,pos)
action
let
val ac=hd ac_seq;
val cs=(#1ac);
val eta=(#2ac+delta_eta);
val lta=(#3ac+delta_lta);
val tta=0;
val pos=0;
in
(cs,eta,lta,tta,pos)
end;
```

Figure 6: VarySeq page

Keeping in mind the very large number of potential candidate sequences (equation 4) it was considered unpractical to represent each candidate sequence as a single token and then having these tokens flow simultaneously through the evaluation functions of the net. Instead of this, a state space based approach was chosen which uses reachability analysis to generate and evaluate the different candidates which can be generated for a certain input received from the *PrepPage.* Choosing this approach neither the *PrepPage* nor the *VarySeq* page with its subpages are in fact ever actually simulated. Instead of this all possible simulation runs are calculated by the state space tool. The relevant nodes of the state space where the evaluation of a certain candidate is complete and the quality value is computed can later easily be identified as dead nodes in the reachability graph. The relevant data associated with the candidate sequences and their respective quality values can be read out by analysing the marking of the dead nodes, a task which is performed by another module, the *Analysisnet* as will be explained further down.

The performance gain expected through using the state space tool for the generation however is achieved at the price of having to enter and calculate the state space repeatedly for each iteration of the sequence planning. This is currently done manually (that is, we trigger the calculation by pressing the associated button of the CPN Tools user interface) [7] which is only practical for a relatively small number of iterations. To be able to analyse the sequence planning behaviour over a large number of planning cycles it would be interesting to develop some means of triggering the (re)calculation of the state space automatically, as a function of some state of the net.

OneQual page

The OneQual page is shown in Figure 7. Its first function is to call the different rating functions (substitution transitions *g2,g3,g5*) used to evaluate the candidate sequences. It then merges the individual quality values calculated by those rating functions to one total quality value which will later decide which candidate sequence is actually implemented (transition *Merge*). As the evaluation of stability criterion *g3* depends on the sequence implemented in the preceding planning cycle, the OneQual page has two main input places, the *NewSeq* place which contains the candidate sequence to be evaluated and the *OldSeq* place whith the sequence implemented before. The control flow over places *c1-c4* serves simply to avoid unnecessary concurrency between the different rating functions which would enlarge the state space but has no effect on the final result as the different rating functions are independent of each other. With regard to the four step scheme of sequence planning introduced in Figure 2 the OneQual page is the one which implements the sequence evaluation step.

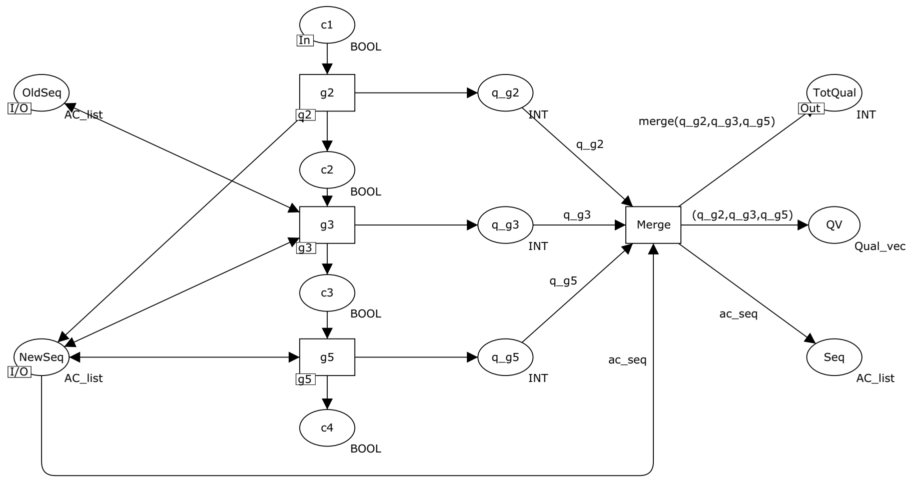


Figure 7: OneQual page

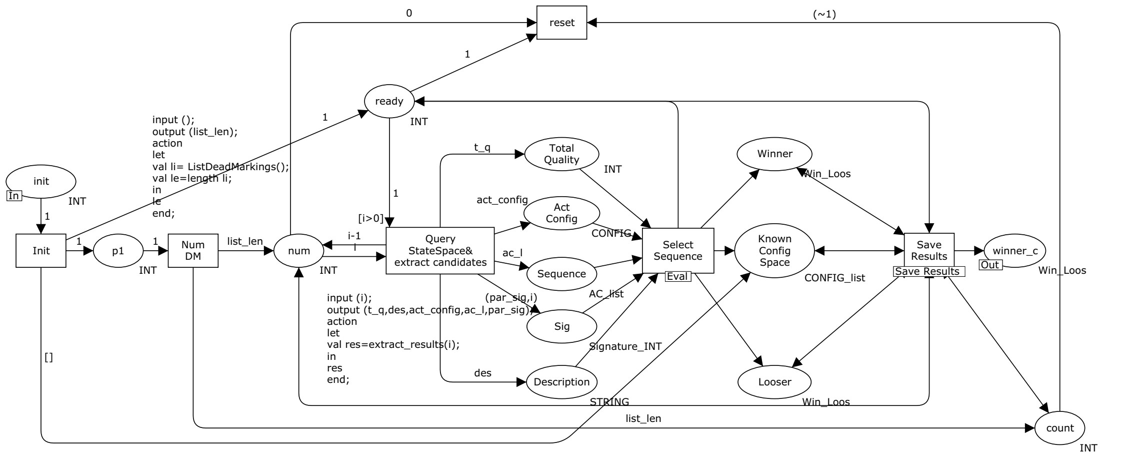
Pages g2,g3,g5 - (Rating functions)

Pages g2 to g5 implement the individual rating functions used to evaluate the candidate sequences. No detailed description of the exact mathematical formulation of the individual functions can be given here, however in general criterion *g2* will rate wether an aircraft in the sequence is able to reach its submitted earliest time of arrival, while *g5* will rate if an aircraft is able to land before its submitted latest time of arrival and thus without the need for entering a holding. The stability criterion responsible for the recursive nature of the sequence planning process is implemented on page *g5* (see also section 2.2 sequence evaluation). A range of additional rating functions have been designed and tested during the development of 4D-CARMA, which have not been modelled yet in the CP-net but would generally be straightforward to integrate in the architecture of page *OneQual.* All individual rating functions output a scalar quality value of type integer from which a weighted total quality value is calculated by the transition *Merge* on the page *OneQual*.

Analysisnet page

The *Analysisnet* page is a direct subpage of the *SequPlan* page and thus located on the same hierarchical level as the *PrepPage* and the *VarySeq* page. The task of the *Analysisnet* is to select from the all candidate sequences (which were generated and evaluated through the *VarySeq* page and its subpages) the sequence(s) with the best total quality values for each possible *input configuration*. These are the sequences which would in practice be implemented, assuming that the system is not overruled by a differing decision of the air traffic controller. The main task of the *Analysisnet* page is thus sequence selection (see Figure 2, four stage scheme of sequence planning).

On the *SequPlan* page (Figure 4), it can be seen that the *Analysisnet* receives no direct input from *VarySeq* at all. The only input it has is a single integer value to initialise/activate the net. This is because all input data the *Analysisnet* needs from *VarySeq* is gained by making queries on *VarySeqs* State Space. Note that while *VarySeqs* State Space is generated the *Analysisnet* has to be deactivated (no enabled transition instances) so that its own functionality is not included in the State Space. This is achieved by having an empty marking on the input place of *Analysisnet* while generating the State Space of *VarySeq*.



```
input ();
output (list_len);
action
let
val li= ListDeadMarkings();
val le=length li;
in
le
end;
```

```
input (i);
output (t_q,des,act_config,ac_l,par_sig);
action
let
val res=extract_results(i);
in
res
end;
```

Figure 8: Analysisnet page

The main functionality of the Analysisnet is included in the transitions *QueryState-Space* and *SelectSequence*. The action part of the *QueryStateSpace* transition first searches for the dead markings of the StateSpace. Each dead marking represents a state where all rating functions for the candidate have been successfully executed and the quality value of a candidate sequence has been computed. Hence the information about the candidate sequence itself (order of aircraft, target times of arrival etc.), the quality value, the input configuration and some other information are extracted from the state space node by querying the marking or the respective places of the VarySeq page and are then shown on the output places of the Transition. The job of the *SelectSequence* transition is to find among all candidate sequences and for each input configuration the "winner(s)" that is the sequence(s) with the highest quality value. Thus it runs one by one through all the candidates while always storing the best known candidate(s) for each input configuration on the *Winner* place and dumping the tokens representing candidates with lower quality values on the *Loser* Place. In detail, this comparison and data handling is realised by the nets on the subpages *new, equal, worse, better*, and *remove* of *SelectSequence* transition, which will not be further explained here.

Once all dead markings (or candidate sequences respectively) of the State Space have been inspected and the optimal sequence for each input configuration has been found the transition and its associated subpage *Save Results* writes the results of the analysis to different textfiles. Three different write-in-file monitors are thereby used to save the winning candidates, the losing candidates and a description of the examined input configurations in three separate files. The *winner_file* and the *input_configuration* file can later be used for analysis and visualisation purposes. The *loser_file* is mainly used during the model development stage, to verify that the model examines the complete set of candidate sequences correctly. Some preliminary results of such an analysis and possible ways of visualising the results are discussed in section 4.1.

ProgressionGraph page

Apart from the write-in-file monitors of the SaveResults page discussed in the last paragraph the *ProgressionGraph* provides a second means to access and process the results of the simulation. The fundamental difference between the two concepts lies in the fact that while the monitor-based solution of the *SaveResults* page considers the behaviour of one single sequence planning cycle in isolation, the *ProgressionGraph* serves to integrate the results of a number of consecutive planning cycles. In order to do that the *ProgressionGraph* relies on some kind of signature which is saved with every sequence (every "winning sequenc" in particular). This signature consists of a simple string typed variable which is on the one hand unique for that winning sequence *w_seq* itself, but on the other hand reveals from which old sequence *old_seq* the winning sequence originates. A possible signature of *w_seq* might for example look like this: "1>105>4>46>89". The signature of the sequence *old_seq* implemented in the preceding planning cycle would then be "1>105>4>46", the very first sequence considered in the simulation would in fact have just the signature "1". With each iteration of the sequence planning process one additional chunk (number) is appended to the signature with denotes the number of the respective dead marking in the state space.

What the *ProgressionGraph* page does is two things:

1. It checks if the signature of a certain sequence on the place Successor place *Suc* fits to the signature of the token on the PredecessorPlace *Pred*. If this is the case, the transition *progress* can fire which will cause the former *successor* sequence to become the new *predecessor* (and the former predecessor be returned to the successor place). Performing this action traces a possible evolvement of the sequence over time.

2. It checks if a) the input configuration (estimated times ETA and LTA) of the aircraft in a sequence of the *predecessor* place is equal to the input configuration of the sequence on the *successor* place but b) the resulting sequences (i.e. order of aircraft) are different. In this case the two tokens can be interchanged by firing the transition *same_config*. In practice this connects two states where all aircraft submitted the identical estimates but received different results from the sequence planner due to the time dependent behaviour of the sequence planning function. This effect will be further explained in section 4.2., the two sequences are denoted as “equally configured”.

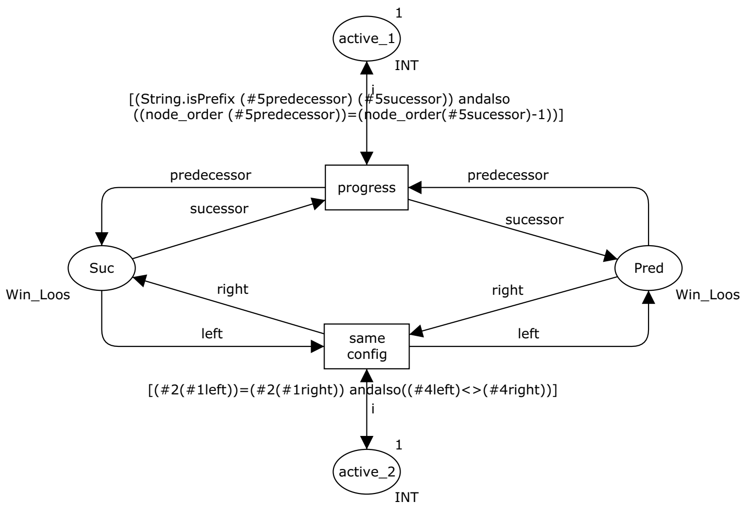

1
active_1
INT
i
[(String.isPrefix (#5predecessor) (#5sucessor)) andalso
((node_order (#5predecessor))=(node_order(#5sucessor)-1))]
predecessor
progress
predecessor
sucessor
sucessor
Suc
Pred
Win_Loos
Win_Loos
right
right
left
same
config
left
[(#2(#1left))=(#2(#1right)) andalso((#4left)<>(#4right))]
i
1
active_2
INT

Figure 9: ProgressionGraph page

The functionality of the page *ProgressionGraph* is particularly useful to analyse the time-dependent behaviour of the sequence planning function which stems from the recursive nature of the sequence planning procedure. In order to examine this behaviour the State Space is generated for the *ProgressionGraph* page (all other pages would have to be without any enabled transitions) and the results are exported to GraphViz using the OGtoGraphViz interface. Some Visualisations, although created with a slightly modified version of OGtoGraphViz are presented below in section 4.2.

4 Preliminary Simulation Results

The section presents some preliminary results achieved through simulations of the above model. The results contain both output produced with the monitor-based analysis to examine one single planning cycle and with the second approach realised on the *ProgressionGraph* page to examine the behaviour of the system over a number of consecutive planning cycles.

For the purpose of this paper a very small example sequence was chosen, which contains only three aircraft ("A","B","C"). It is assumed that the initial state of the sequence planning process is as shown in figure 10.

Aircraft	ETA [3]	LTA [4]	TTA [5]	POS [6]
A	110	240	110	1
B	120	250	185	2
C	123	207	260	3

Figure 10: Initial state of example sequence

All time values for ETA, LTA and TTA are assumed to be in seconds, however only the relative differences between those numbers are practically relevant, the absolute values are arbitrary.

4.1 Monitor-based analysis of a single planning cycle

For the monitor-based analysis it was considered that aircraft "A" and "B" would be free to vary their ETAs before the next planning cycle for one of the following values: $delta\_eta \in \{-90, -60, -30, 0, 30, 60, 90\}$. Meanwhile the ETA of "C" would remain constant as well as the LTAs of all three aircraft "A","B","C". Choosing a negative delta_eta (earlier ETA) would in practice represent a situation where either an aircraft experiences tailwinds or where it chooses a slightly higher speed signifying higher fuel consumption but a potentially shorter flight time (assuming there is no delay at the airport). In contrast choosing a positive delta_eta (later ETA) could mean that an aircraft experiences headwinds or chooses a more economic cruising speed paying the price of a potentially delayed arrival time.

The purpose of the simulation is then to examine how the revised ETA influences the aircrafts' position in the sequence as planned by the AMAN, as well as the target time of arrival the sequence planner would calculate for each aircraft on the basis of the new input configuration.

Figure 11 shows the planned position of aircraft "A" and "B" in the sequence plotted against the submitted earliest times of arrival (ETAs) of "A" and "B". The ETA of "A" and "B" are drawn on the horizontal axes, the aircrafts' position in the sequence is indicated on the vertical axis by the triangles (aircraft "A") and squares (aircraft "B") in the graph. Note that only the squares and triangles themselves represent actual and discrete simulation results from the CPN model. By contrast the mesh surface was fitted

[3]ETA=Submitted Earliest Time of Arrival, that is earliest time the aircraft could reach the airport according to its own estimation

[4]LTA=Submitted Latest Time of Arrival, that is latest time the aircraft could reach the airport according to its own estimation (without entering a holding)

[5]TTA=Target Time of Arrival planned by sequence planner

[6]P0S=Position of aircraft in the sequence planned by sequence planner

to the measured data afterwards in order to grasp their position in three-dimensional space more easily. A continuous change between two positions as the cubic interpolation function suggests is in practice of course not possible. Rather the continuous values can be interpreted as a representation of the uncertainty on how the Sequence Planner would react between the two discrete measurements where the exact system behaviour was simulated. Of course for every single configuration of the free variables the Sequence planner would in practice return two differing positions for aircraft “A” and “B” making an intersection of the two surfaces (two aircraft with equal position) impossible if an infinite number of values were computed.

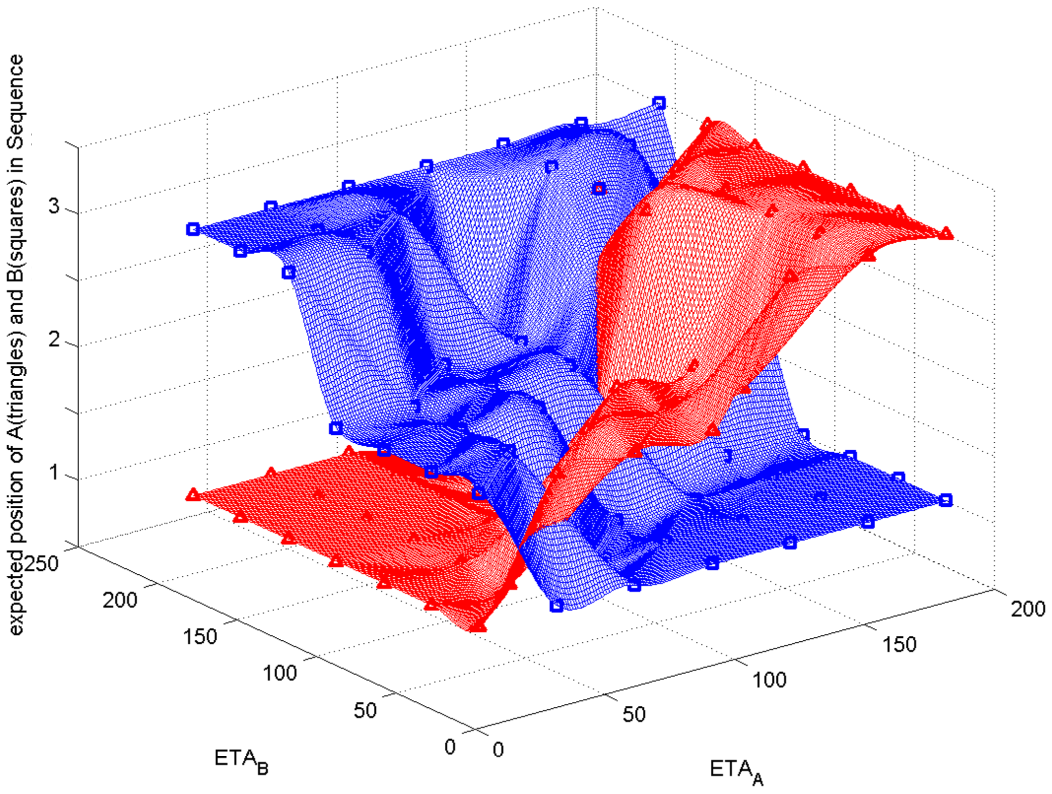


Figure 11: Planned aircraft position (POS) as a function of submitted earliest time of arrival (ETA) of “A” and “B” (ETA of “C” constant).

Figure 12 shows the planned target time of arrival of aircraft “A”,“B” and “C” plotted against the submitted earliest time of arrival of “A” and “B”. The TTAs actually simulated with the CPN models are represented by the triangles, squares and circles for “A”, “B” and “C” respectively. Some of the restrictive comments made with regard to the interpretation of the interpolated surfaces in Figure 11 remain valid for Figure 12. For example the intersections between surfaces are still not valid in a strict sense (since that would mean more than one aircraft with the same target time of arrival, which would necessarily violate separation criteria) and are due to the characteristics of the interpolation function. A difference with Figure 11 however lies in the fact that continuous values of the TTA are now necessary, as the target time is in fact a continuous variable.

From an operational point of view or air traffic management the Target Time of Arrival (TTA) is in most cases more meaningful than the planned position of an aircraft in the sequence. That is, it is normally more important for the flightcrew or the airline to know at what time the aircraft will land than which other aircraft will be in front or behind in the sequence. From the perspective of the individual aircraft the gradient of the surface could then basically be interpreted as an incentive to adjust its behaviour (speed/estimates) in one way or the other [11, 5]. From the perspective of the air traffic control (or the perspective of the AMAN alternatively) the most upper surface of the three is generally the most interesting, as it reveals the time when the last aircraft in the considered sequence will be landed.

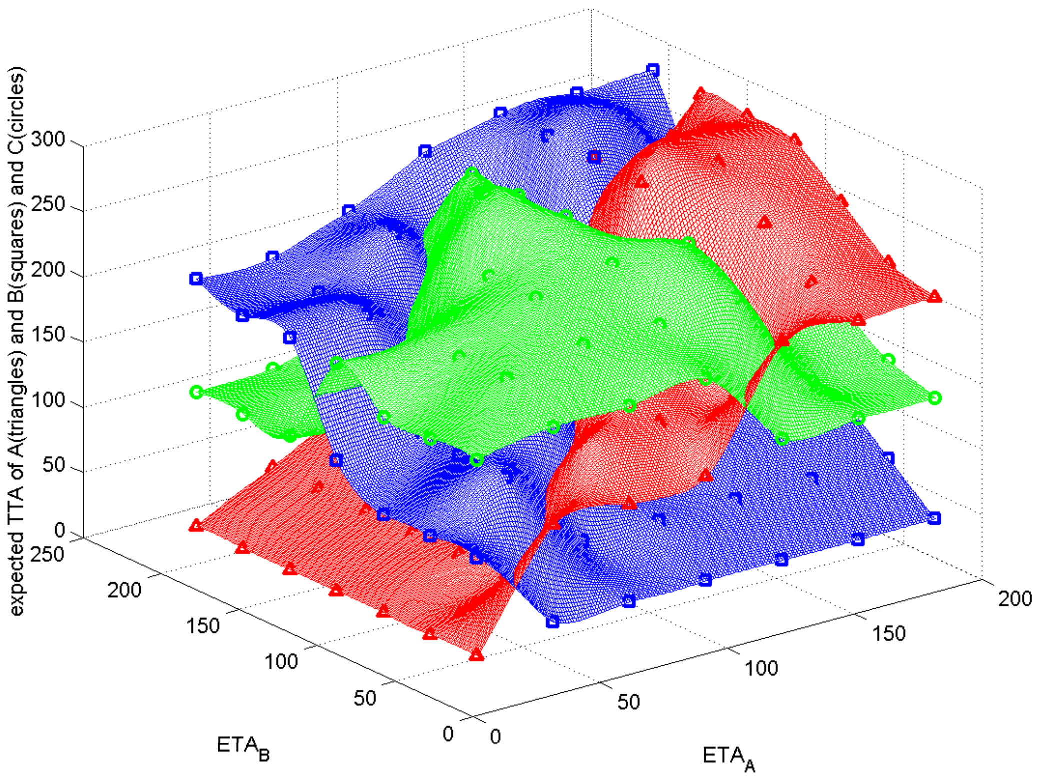


Figure 12: Planned target time of arrival (TTA) as a function of submitted earliest time of arrival (ETA) of "A" and "B" (ETA of "C" constant).

4.2 Analysis of system behaviour over consecutive planning cycles

Figure 13 shows a section of a graph which describes the system behaviour of the sequence planner over two consecutive planning cycles. The same initial state of the example sequence is assumed as in the previous discussion of the monitor-based solution in section 4.1. However to keep the number of nodes within manageable limits for the context of this paper the range of potential values for delta_eta was further constrained to be $delta\_eta \in \{-60, 0, +60\}$ in the first planning cycle and $delta\_eta \in \{0, +60\}$ in the second planning cycle. As in section 4.1 it is assumed that aircraft "A" and "B" are free to vary their earliest time of arrival while aircraft "C" was obliged to keep its ETA constant. The latest time of arrival (LTAs) of all three aircraft is also held constant in both planning cycles.

After generating the *ProgressionGraph* (describing the progression/evolvement of an aircraft sequence over time and) with the state space tool, the visualisation of the Graph was realised using the GraphViz software [8] in conjunction with a somewhat customised version of the OGtoGraphViz interface [6] supplied with CPN Tools. The customisation consisted basically in the following:

- The edgelabels between two nodes were rewritten so as to contain the *delta_eta* which was used to revise the estimated time of an aircraft between to planning steps. Where *delta_eta* equals zero the value is omitted in the edgelabel.
- The nodelabels were configured to contain a description of the sequence itself (thus indicating the position of each aircraft in the sequence) as shown in Figure 13. Using appropriate switches more information can be shown in the nodelabels for example the respective target times of arrival or latest time of arrival submitted by the aircraft.
- To graphically indicate the branches and nodes of the tree which are favourable from a certain aircraft's perspective a function was developed which sets the linewidth of an individual node depending an aircraft's position in the sequence. In Figure 13, the more favourable the position for aircraft "B" the broader the line surrounding the label of the node is drawn.
- A second type of edge drawn in dotted lines was used to connect nodes which are "equally configured", that is which contain sequences with identical input configuration, but with a different order of aircraft implemented by the sequence planner. The edges between equally configured nodes were detected by the transition *same_config* on the *ProgressionGraph* page. Where such edges exist between two nodes, they naturally run in both directions. In Figure 13 it can be seen for example that dotted edges exist between the first-order node where aircraft "B" revised the Earliest Time of Arrival by *delta_eta=+60* and second-order node where "A" in the first planning cycle revised its ETA by *delta_eta=-60* and then in the second planning cycle both "A" and "B" revised their ETA by *delta_eta=+60*. The addition of the revision values over the two consecutive cycles in both cases leads

to the same configuration (thus the two nodes are "equally configured"), however the resulting sequence is "ABC" for the first-order node and "ACB" for the second order node. Thus by switching "A"s ETA back and forth ending up with the same configuration we have come to a different final result than by keeping "A's" ETA constant.

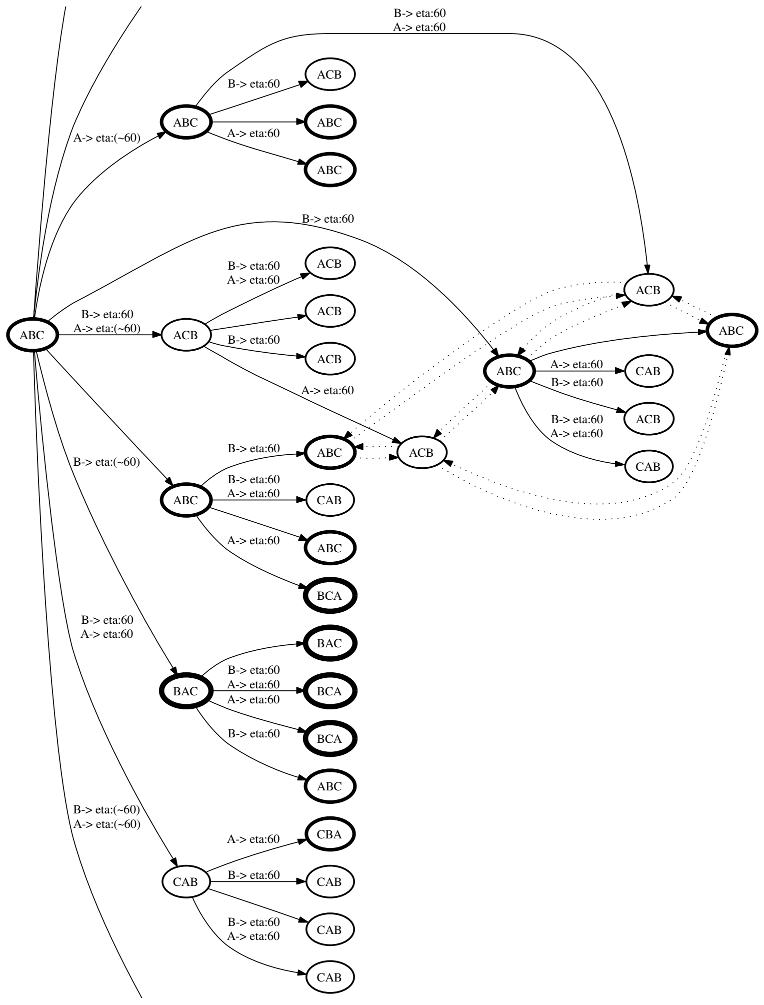


Figure 13: Planned sequence as a function of submitted earliest time of arrival over two consecutive planning cycles

5 Conclusions

The paper introduces a CPN model of a potential future sequence planning process for cooperative arrival planning in air traffic control. The basis of the cooperation consists in that the airborne side, that is the involved aircraft, provide information for example with regard to their estimated earliest and latest times of arrival at the airport. This information is then used by the sequence planner (on the ground) to establish a favourable sequence in which the aircraft will be led to the runway. The model was built in order to achieve a better understanding of how the sequence planning responds to certain behaviour of individual actors/aircraft involved in the process. The basic sequence planning procedure assumed and modelled in this paper (i.e. the core sequence planning algorithm) was developed elsewhere and has already been implemented before in an even more complete prototype system using conventional programming languages [9, 10, 12]. Thus it can be argued that the model somewhat duplicates the functionality of the existing prototype. The duplication is justified by the effective and intuitive means of analysing the model provided with CPN Tools. Some of the benefits with respect to analysis were already pointed out in section 4. For example nodes were searched in the state space which represented identical input configurations of all aircraft, but led to differing responses of the planning system due to the time-dependent nature of the planning process. The extensive possibilities and flexible manner of analysing the system behaviour of the CP-net still appears as a strong argument in favour of the model. It remains to be seen how the model performs when confronted with a more comprehensive set of input data and particularly with aircraft sequences of an operationally relevant length (in the order of magnitude of 8 aircraft). While some interesting effects of sequence planning can definitely be examined on fairly small example sequences, other investigations might require longer sequences and run-time may become a critical issue. However there should also be some room for performance improvement of the current model for example by transferring some purely sequential processes from the net pages to separate functions. To be able to more easily analyse the planning systems behaviour over a number of consecutive planning cycles it would be beneficial if the (re)entering of the state space could be triggered automatically as a function of some state of the model. Currently the (re)entering of the state space tool has to be initiated by hand via the graphical user interface of CPN Tools which is only practical for a small number of iterations. Future work on the model should therefore include an effort to realise by some means the automatic (re)entering of the state space tool, or, if this approach turns out to be technically impracticable, to restructure the model in a way so that only one calculation of the state space is required to analyse consecutive planning cycles.

References

[1] U. Büchner, B. Czerlitzki, H. Hansen, H. Helmke, S. Pahner, A. Pfeil, M. Schnell, H. Schnieder, P. Theis, and M. Uebbing-Rumke. KOPIM-AIRCRAFT - Entwicklungsstatus von boden- und bordseitigen Systemen und von operationellen ATM-Verfahren und Konzepten für ein kooperatives ATM. 2005.

[2] B. Czerlitzki, H. Uebbing-Rumke M. Helmke, R. Edinger C. Stump, and J. Temme M. Strohmeyer. Konzept und Spezifikation der Bord-Boden-Kopplung. 2005.

[3] Eurocontrol. Phare Advanced Tools Arrival Manager Final Report. Technical report, 1999.

[4] Eurocontrol. Summary Report of the EVP AMAN Rome Real Time Simulation 2004. Technical report, Bretigny sur Orge, France, 2005.

[5] T. Günther. Validierung der Interdependenzen eines Systems zur Ankunftszeitoptimierung von Flugzeugen in Ergänzung zu einem Arrival Management an einem Verkehrsflughafen. Technical report, 2004.

[6] C. T. Help. *OGtoGraphVizInterface.* 2006.

[7] K. Jensen, S. Christensen, and L. M. Kristensen. CPN Tools - State Space Manual. Technical report, Aarhus, Denmark, 2006.

[8] A. Research. *GraphViz Manual.* 2006.

[9] D. Schwarz. Anflugssequenzplanung mit dem A* Algorithmus zur Beschleunigung der Sequenzsuche. Technical report, 2005.

[10] H. Sievert. Erweiterung eines Anflugplanungssystem-Prototyps. Technical report, 2004.

[11] G. T. and H. Fricke. Potential of Speed Control on Flight Efficiency. In *ICRAT - Second International Conference on Research in Air Transportation*, pages 197–201, 2006.

[12] T. Zielinsky. Erkennung und Lösung von Konflikten bei Anflug-Trajektorien. Technical report, 2003.

A CPN Model of a SIP-Based Dynamic Discovery Protocol for Webservices in a Mobile Environment

Vijay Gehlot and Anush Hayrapetyan

Center of Excellence in Enterprise Technology
Department of Computing Sciences
Villanova University, Villanova, PA 19085, USA
vijay.gehlot@villanova.edu, anush.musoyan@villanova.edu

Abstract

The Session Initiation Protocol (SIP) was conceived for internet telephony. Owing to its simplicity, flexibility, and reuse capability, SIP is now viewed as an enabler of converged communications across distributed networked entities. In addition, several other protocols or applications have been defined in terms of the basic SIP components. Of particular interest to us is the SIP-based Discovery protocol that has been developed for the Multi-Channel Service Oriented Architecture (MCSOA). MCSOA is an enhanced Service Oriented Architecture (SOA) geared towards the Net-Centric Enterprise Solutions for Interoperability (NESI) initiative of the US Department of Defense (DoD) and Defense Information Systems Agency (DISA). The Net-Centric Enterprise Services (NCES) defined by NESI are a set of net-centric services, nodes, and utilities for use in DoD domain and mission-related enterprise information systems. The purpose of the MCSOA Discovery protocol, when compared to the traditional SOA/UDDI (Universal Description, Discovery and Integration) approach, is to enable location transparency, availability awareness, service guarantees, context-based routing, and fault-tolerance. In this paper we give details of a CPN model of the MCSOA Discovery protocol. The work presented is part of a larger project to integrate a CPN-based model-driven approach into MCSOA software development. We also briefly discuss our experience with the CPN Tools in the context of this project.

1 Introduction

Service Oriented Architectures (SOAs) are attractive for enterprise systems because of their inherent loose coupling and reusability aspects. In order to capitalize on the flexibility and interoperability afforded by SOAs, the US Air Force Electronics Systems Center and the Defense Information Systems Agency undertook a collaborative initiative to define a framework known as Net-Centric Enterprise Solutions for Interoperability (NESI). It is envisioned that NESI will add value not only to day-to-day enterprise-level business operations but also to command and control as well as logistics and warfare. Figure 1 gives an instance of this broad scale interoperability and integration proposed under the NESI initiative [15].

These data interoperability capability requirements coupled with specific defense needs add a new dimension to conventional SOAs. In particular, SOAs for defense must be able to address the following:

- Service guarantees
- Fault tolerance
- Dynamic service discovery
- Interoperable multiple connection types
- Availability awareness
- Load balancing
- Security

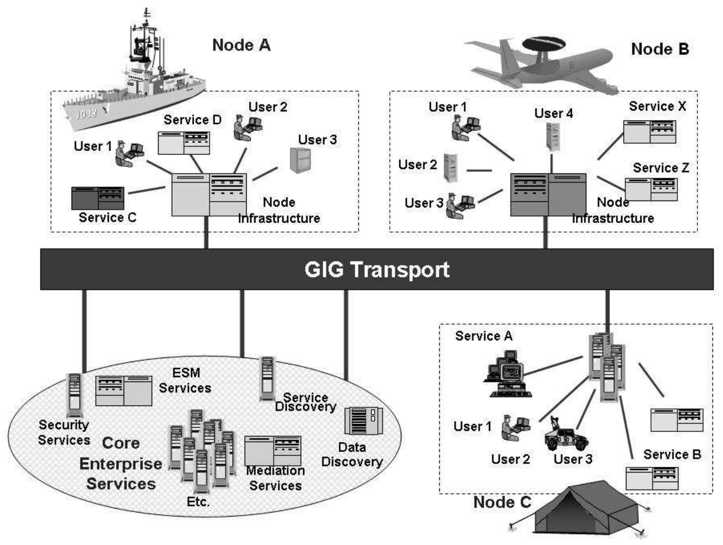


Figure 1: Node interoperability in NESI Net-Centric Enterprise Architectures.[a]

[a]The depicted *Global Information Grid (GIG)* is the future *US DoD* transport mechanism that will include an integrated heterogeneous network consisting of the mobile and fixed networking components at ground-based, air-based and space-based locations.

To address these needs, Gestalt LLC, a defense contractor, proposed a Multi-Channel Service Oriented Architecture (MCSOA) [14]. Born out of this effort was a Villanova University research project called ARCES (Applied Research for Computing Enterprise Services) to integrate a model-driven approach into MCSOA software development and to provide capabilities for system analysis. Our decision was to use Colored Petri Nets (CPN) for this purpose [8]. What sold the idea to our sponsors was extensively documented work by other researchers in the industrial uses of CPN in various large-scale real-life systems including military systems [13]. In addition, CPNs are well supported by the great deal of theoretical research that has gone into defining them, the associated analysis techniques developed for them, and the CPN Tools system that allows flexible experimentation with them. Finally, availability of an active CPN Tools support team at the University of Aarhus worked towards our advantage. In this paper we give details of a CPN model of the MCSOA Discovery protocol that was created as part of our initial efforts to integrate CPN into MCSOA development. CPNs have been used to model a variety of protocols. For example, see [9], [5] and [1]. However, to the best of our knowledge, there have been no reported CPN models of SIP or SIP-based protocols to date.

The remainder of this paper is organized as follows. The next section contains some details of the MCSOA architecture, the Session Initiation Protocol (SIP) Presence model and the MCSOA Discovery protocol that is built on top of it. Section 3 contains details of our CPN Discovery model. We present our conclusions in Section 4 where we also briefly discuss our experience with the CPN Tools in the context of this project. Finally, to enhance readability, we provide a list of abbreviations used throughout the paper in Appendix A.

2 MCSOA, Presence and Discovery

From a software point of view, MCSOA is an implementation of a SOA. It can be viewed as a development, deployment, and discovery framework for SOA applications. There are three main components of a SOA:

1. The *Service Provider* publishes and makes a service available.
2. The *Service Consumer* invokes and uses a service.
3. The *Service Discovery* mechanism maintains information about services and their providers and allows for discovery of services.

The conventional webservices based view of a SOA uses HTTP/SOAP for transport and communications, and the discovery mechanism is based on the UDDI framework which makes it static [6]. This structure is too restrictive for general defense applications where multiple protocols and transport mechanisms are used for a

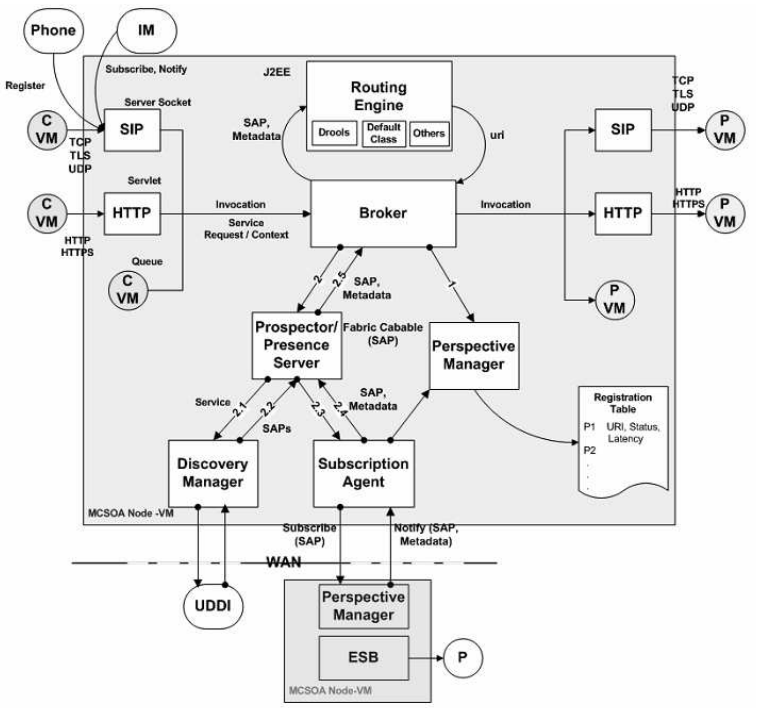


Figure 2: The MCSOA Node Architecture.

variety of communications. MCSOA solves this *multi-channel* problem by defining a *node* and a fixed intra-nodal *transport mechanism* and providing appropriate wrapper mechanisms for different channels. Providers and consumers of services connect through a MCSOA node. MCSOA's internal transport is based on the Session Initiation Protocol (SIP) [17, 2]. Since SIP is essentially a signaling protocol, using it as a wrapper poses no technical difficulties. Furthermore, there is also an added benefit of using SIP: MCSOA is able to leverage SIP-based Presence capabilities [3] to define its dynamic discovery mechanism. To be able to do dynamic discovery and to provide other facilities, MCSOA adds the following fourth component to the basic above listed repertoire of SOA components:

4. The *Service Broker* is an intermediary between a service consumer and a service provider.

In MCSOA architecture, the broker is part of a MCSOA node. Several MCSOA nodes can be connected to each other forming a MCSOA *fabric*. The fabric is a dynamic concept in that new nodes may join the fabric through the MCSOA discovery process. Furthermore, the services available at a given node may also change dynamically. For example, a surveillance aircraft connected to, say, node A, moves out of range and reconnects to, say, node B, to provide the same set of services. This detachment and re-attachment is handled by the underlying SIP protocol.

The SIP architecture used for *Presence* consists of the following three entities [4]:

1. *Presentity* is the provider of presence information, that is, an entity for which presence information is being tracked.

2. *Watcher* is the consumer of presence information, that is, an entity interested in presentity status information.

3. *Presence Server* maintains and manages presence information, that is, it keeps track of presence information of presentities and sends notifications to watchers of the changes in status.

Figure 2 depicts the internal organs of a MCSOA node. A complete description of this architecture is beyond the scope of this paper. In this figure, the component named Prospector/Presence Server implements the SIP Presence Server capabilities. Internally SIP messages are used to establish subscriptions and to communicate presence changes. SIP *REGISTER* messages are used to establish initial contact with MCSOA. Under SIP, registration creates basic presence information and is handled by a special entity called *Registrar* (component labeled Perspective Manager in the figure). In MCSOA, as is typically the case in most SIP-related presence

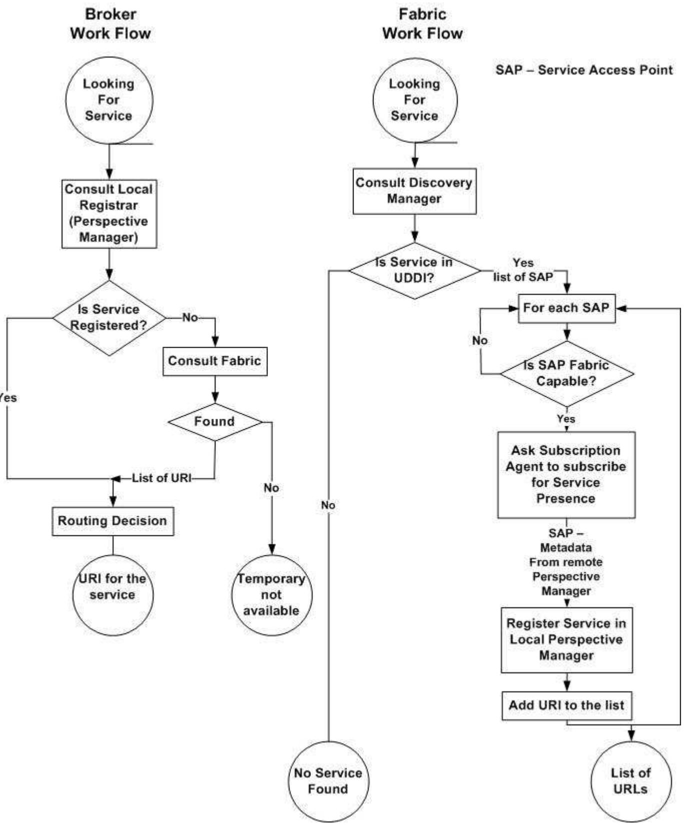


Figure 3: MCSOA Dynamic Discovery Process.

models, the Presence Server and the Registrar are co-located. SIP *NOTIFY* messages are used to communicate presence information to Watchers. SIP *SUBSCRIBE* messages are used to declare an interest in presence information for particular presentities. Watchers send *SUBSCRIBE* messages to the Presence Server. The Presence Server sends a SIP *NOTIFY* response for each subscription request initially as well as whenever there is a change in the status of the subscribed presentity. Presentities send *REGISTER* messages to the Presence Server. Registration is processed by the Registrar component. The Registrar notifies the Presence Server when a presentity's presence status changes. Both the *SUBSCRIBE* and the *REGISTER* requests carry an expiration time. An entity may re-*SUBSCRIBE* or re-*REGISTER*.

The components Perspective Manager, Discovery Manager, and Subscription Agent shown in Figure 2 provide the discovery capability. Discovery is the process of finding a service provider for a consumer request. There are three possible situations for a request:

1. Consumer `bob@node1` is looking for service provider `alice@node1`. In this case the request is for a service at the local service access point. This case is handled by simply doing a lookup of the local registration table.

2. Consumer `bob@node1` is looking for service provider `alice@node2`. In this case the request is for a service at the remote service access point. This case is handled by simply forwarding the request to the remote node. Thus, effectively, the local node acts as a SIP *Proxy Server* in this case [17, 2].

3. Consumer `bob@node1` is looking for service provider `alice`. This constitutes a request without specific service endpoint and triggers the MCSOA discovery process depicted in the flow diagrams in Figure 3. As shown, it is a two step process:

   (a) The first step corresponds to intra-nodal activities and is termed *Broker Work Flow* in the diagram. In this case, the service Broker first consults the Perspective Manager/Registrar for service availability. This step may succeed owing to a previous discovery process or because of a local provider being available in the local registration table.

   (b) The second step is executed if the first one fails. This corresponds to inter-nodal activities and is termed *Fabric Work Flow* in the diagram. In this case, the service Broker consults the Prospector/Presence Server and uses its Discovery Manager to locate candidate providers for the desired service. This may yield zero or more candidate Service Access Points (SAPs). For each provider SAP, the Subscription agent interacts with the Perspective Manager of the remote node to subscribe to remote provider. Again this step may or may not succeed. In case of a success the local Perspective Manager updates the registration table with information about non-local providers of the service and forwards the request to the remote provider. Otherwise, a failure message is returned to the consumer.

In summary, when a request without specific service endpoint is received and such a service is not locally available, then the local node essentially becomes a watcher for the presence information of the service provider by making use of the underlying presence capabilities of each MCSOA node. The initial contact points may be obtained via a UDDI or any other location service lookup. Rediscovery may be avoided by updating the local registration table based on presence information gathered during the discovery process. We would like to point out that the simplicity of the work flows depicted in Figure 3 can be misleading, for, as we shall shortly see, the underlying details of this process are quite involved.

It should be clear from the discussion above that *Presence* is a crucial part of the *Discovery* process. We built a CPN model of *Presence* before we built the *Discovery* model. Details of the *CPN Presence model* are described in [7]. In the next section we present our CPN model of the *Discovery* process which is the focus of this paper.

3 CPN Discovery Model Description

This section presents a description of our CPN Discovery model. We start by giving a high level overview of the structure of the model in the first subsection. The remaining subsections give a more detailed description of the individual components making up the *CPN Discovery model*. Due to space limitations, a detailed description of some of the lower-level entities of the model will be omitted.

3.1 Hierarchy

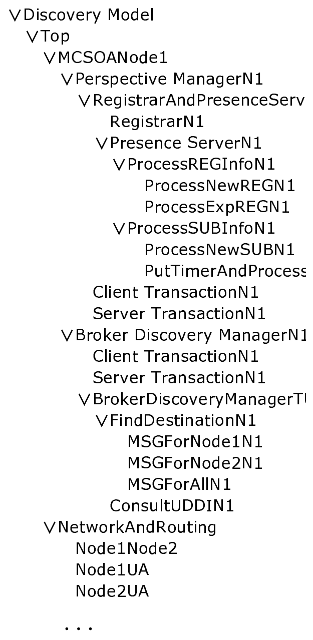
The *Discovery* CPN model consists of five main components: two *User Agents*, two *MCSOA nodes* and one *Network and Routing* component. Each *User Agent* is connected to a *MCSOA node*, which is responsible for processing the messages received from that *User Agent*. The two *MCSOA nodes* can also communicate with each other so as to enable them to use each others' services. The *Network and Routing* component ensures the communication between the *User Agents* and the *MCSOA nodes*, as well as that between the two *MCSOA nodes*.[1] Each of these five components contain further subcomponents, called *modules* or *pages*, which implement specific functionalities of the component. These pages may further contain subpages, thus creating a natural hierarchical structure for the *Discovery* CPN model. There are a total of 73 pages in the *Discovery* CPN model, each corresponding to a line in Figure 4.[2] The page Top is the outermost module of the hierarchy. Many pages have *substitution transition(s)* [8]. In the figure, such pages contain a "V" before their name. We will describe substitution transitions in more detail later.

The five components of the *Discovery CPN model* are identified as: $User\ Agent$ 1, $MCSOA\ Node$ 1, $User\ Agent$ 2, $MCSOA\ Node$ 2 and *Network and Routing*. The pages corresponding to these five components directly descend from the page Top. The two $User\ Agent$ pages $SIPUserAgentsN1$ and $SIPUserAgentsN2$, which are not visible in Figure 4, are responsible for sending registration (*REG*) requests for both consumers and providers, subscription (SUB) and message (MSG) requests from consumers to providers and notification (NTF) messages to either consumers or $MCSOA\ nodes$. Each message has an automatically generated unique *call ID* (as required by SIP [17]) associated with it, which is used to identify the responses for that message. $SIPUserAgentsN1$ communicates with $MCSOANode1$ page and $SIPUserAgents2$ communicates with $MCSOANode2$ page through *Network and Routing* component. The main purpose of the $MCSOA\ nodes$ is to handle the REG, SUB, MSG and NTF requests:

1. **Registration** : Each time a $MCSOA\ node$ receives a registration request, it adds the entity to its local registration table *REGTableN1*. If the entity is already present in $REGTableN1$, the local information about that entity gets updated. Also, a timer corresponding to each register request is set to expire

[1] In the future, the functionality of the Network and Routing component can be extended to include additional features, such as converting messages from one type to another, or message compression and decompression.

[2] We did not display the full hierarchy, since it would take too much space. We only included the structures of one of the MCSOA nodes and the *Network and Routing* component.



```
∨Discovery Model
  ∨Top
    ∨MCSOANode1
      ∨Perspective ManagerN1
        ∨RegistrarAndPresenceServ
            RegistrarN1
          ∨Presence ServerN1
            ∨ProcessREGInfoN1
                ProcessNewREGN1
                ProcessExpREGN1
            ∨ProcessSUBInfoN1
                ProcessNewSUBN1
                PutTimerAndProcess
          Client TransactionN1
          Server TransactionN1
      ∨Broker Discovery ManagerN1
          Client TransactionN1
          Server TransactionN1
        ∨BrokerDiscoveryManagerT
          ∨FindDestinationN1
              MSGForNode1N1
              MSGForNode2N1
              MSGForAllN1
            ConsultUDDIN1
    ∨NetworkAndRouting
        Node1Node2
        Node1UA
        Node2UA
      . . .
```

Figure 4: The partial hierarchy page — overview of the CPN *Discovery* model.

after the period of time specified by the expiration parameter of that *REG* request. When the timer expires, the corresponding entity is removed from the local *REGTableN*1 and becomes unknown to the *MCSOA node*.

2. **Subscription** : When a *MCSOA node* receives a subscription request, it adds the subscription to its *SUBList* table and sets a timer for that subscription. It also sends a response to the source of the *SUB* request with the same *call ID* as that of the incoming request, and a response *code of* 200 [17]. The purpose of a subscription message is to express interest in the presence information of a particular provider. Thus, after sending a subscription request, the consumer waits for a notification message *NTF* about the status of the provider. This message lets the consumer know whether or not the provider is registered, and also contains the time that remains before the expiration of the subscription. When the status of the provider changes, notification messages are sent to all consumers which have unexpired subscriptions in the *SUBList* table for that provider. When the registration time of the provider expires, all subscriptions for that provider are removed from the SUBList and put into the *PendingSUB* list. The latter stores the pending subscriptions for a provider.

3. **Message** : Handling *MSG* requests is the main task of the *Discovery* model. Two types of *MSG* messages can be received on each of the two *MCSOA nodes* - *MSG with a specific service end point* and *MSG without a specific service end point*. *MSG* with a specific service end point is used when the destination of the message, i.e. the *MCSOA node* for which the message is designated, is known in advance. In this case the *MCSOA node*, corresponding to the *User Agent* sending the *MSG* request, forwards the message directly to the destination *MCSOA node*. If the message is designated for the local *MCSOA node*, then it gets processed locally. *MSG* without a specific service end point is used when messages are not intended for any particular *MCSOA node*. In this case, the *MCSOA node* receiving the message consults its local *REGTable* first. If the provider specified in the message is found in the local *MCSOA node*, then the *MCSOA* forwards the message to the corresponding *User Agent* and waits for a response. Otherwise, a *Discovery Process* is triggered, during which a global static table (*UDDI* or any other suitable location server) is consulted to find a *MCSOA node* where a provider might be registered. If a provider is not found in the *UDDI*, a response with *code* 480 [17] is sent back along the reverse path to the source of the message. If the *UDDI* contains at least one record of that provider, the message is forwarded to the *MCSOA node* indicated by the *UDDI* for processing.

The messages exchanged among the various components of the *Discovery* model follow the *SIP* "format" of a message. The colorset *SIPMsg* associated with a message is of the form:

```
colset SIPMsg = union SIPReq : SReq + SIPResp : SResp;
```

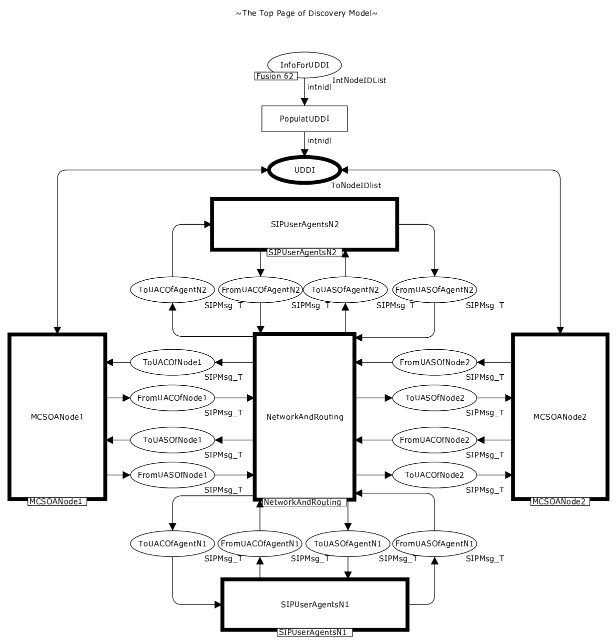


Figure 5: The *Top* page.

where

```
colset SReq = product CallID * FromNodeID * ToNodeID * SIPMethod *
          SIPBody * MsgSize * Expire;

colset SResp = product CallID * FromNodeID * ToNodeID * SIPRespCode
           * SIPBody * MsgSize * Expire;
```

Here we skip the explanation of the various other colorsets and functions declared in the model in order to keep the paper within the page limits.

3.2 The Page *Top*

Figure 5 depicts the page *Top*, which is the first page of the *Discovery* CPN model hierarchy (see Figure 4). The ellipses in the figure are called *places* and are used to model the states of the net. The rectangles in the figure are called *transitions*. The state of the CPN model is called a *marking* and is a distribution of *tokens* on the places of the CPN model. The *Top* page captures the high-level overview of the model and the aforementioned five primary components - *User Agent* 1, *MCSOA Node* 1, *User Agent* 2, *MCSOA Node* 2, and *Network and Routing* - correspond to the five large transition boxes in the figure. Below each large transition box, there is a smaller box, implying that these transitions are substitution transitions associated with the page named in the smaller box. From now on, we will refer to such transitions and their corresponding pages interchangeably. Also on the page *Top*, there is a transition to populate the place *UDDI*. This transition is enabled immediately after the *Start* page completes the initial loading of the input data from various input files.

3.3 The Description of the MCSOA Node

We now describe the structure of the first *MCSOA node*. The second one is analogous, hence we will omit it. We present only those components from Figure 2 that participate in the *Discovery process*, namely, *Perspective Manager*, *Broker Discovery Manager* and *UDDI*. We will initially give a high-level description of the *MCSOA node*, and then continue with a detailed description of the components making up the *MCSOA node* as depicted in the hierarchy shown in Figure 4. This structure of presentation will be used for the remaining components of the *Discovery model* as well.

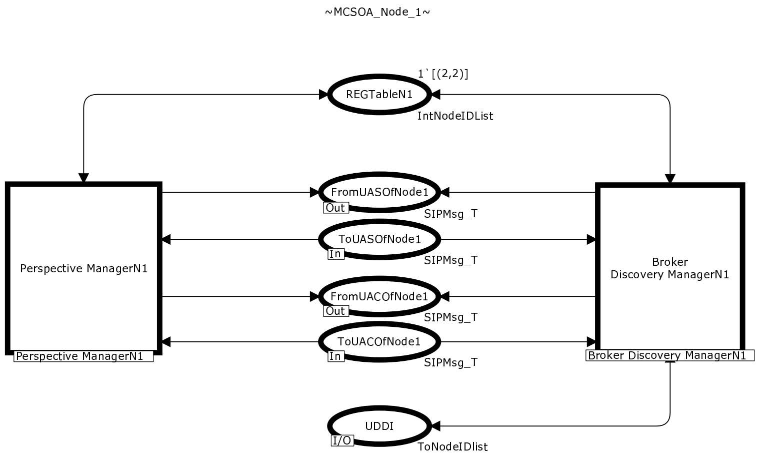


Figure 6: The $MCSOANode1$ subpage of Top page.

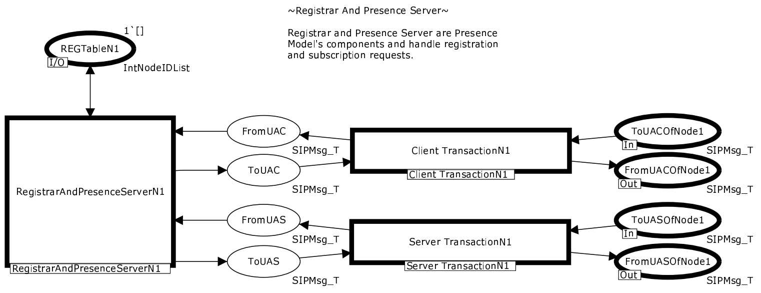


Figure 7: The *PerspectiveManagerN1* subpage of *MCSOANode1* page.

3.3.1 The *MCSOANode1* subpage of *Top* page

Figure 6 depicts the page $MCSOANode1$. A $MCSOA$ *node* consists of two main parts — *Perspective Manager (PM)* and *Broker Discovery Manager (DM)*. The *Perspective Manager* is responsible for handling *REG* and *SUB* requests and is designed according to the requirements of the *Presence Model* that are described in [17]. The *Broker Discovery Manager* handles the *MSG* requests, both with and without a specific service end points. Notice that the place $REGTableN1$, the local registration table of $MCSOANode1$, is connected to both the *PM* and the *DM*, whereas the *UDDI* is connected only to the *DM*, since the latter handles the discovery triggered by *MSG* messages. The place $REGTableN1$ is connected to the *PM*, since the *PM* needs to update it when a new registration or the expiration of a registration occurs. Also, the *PM* needs to consult $REGTableN1$ when a *SUB* message is received in order to find whether the provider mentioned in the *SUB* message is in $REGTableN1$ or not. On the other hand, the *DM* needs to consult $REGTableN1$ during *local routing* (i.e. routing where destination *MCSOA node* is the same as the source *MCSOA node*). The other four places ensure the communication between *MCSOA Node* 1 and other components of *Discovery* CPN model.

3.3.2 The *PerspectiveManagerN1* subpage of *MCSOANode1* page

Perspective Manager (*PM*) is the part of a *MCSOA node* where registration and subscription messages coming from the *User Agent* are handled and notification messages are generated (see Figure 7). The workload of the *PM* is distributed among its three subpages, corresponding to the large boxes in the figure. The $ClientTransactionN1$ and $ServerTransactionN1$ subpages, which are located on the right side of the figure, assure *SIP-compliant* communication between *MCSOA Node* 1 and the other components of the *Discovery* model.[3] All registration, subscription and notification messages that go through the *Client* and *Server Transaction* subpages are generated and analyzed in the $RegistrarAndPresenceServerN1$ page.

[3]$SIP - compliant$ communication means that in order to pass a message from one place to another, the message has to pass through the *Client Transaction* on the source side, and be received by the *Server Transaction* on the destination side. The response for that message travels from the *ServerTransaction* on the destination side to the *ClientTransaction* on the source side.

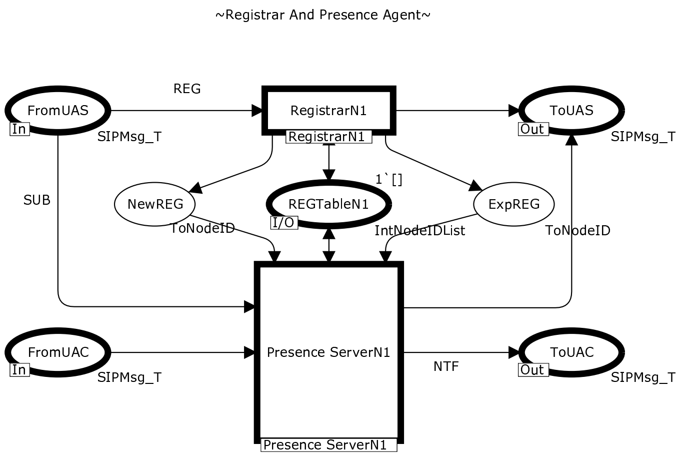


Figure 8: The *RegistrarAndPresenceServerN1* subpage of *PerspectiveManagerN1* page.

3.3.3 The *RegistrarAndPresenceServerN1* subpage of *PerspectiveManagerN1* page

The page *RegistrarAndPresenceServerN*1 (Figure 8) is responsible for handling *REG* and *SUB* messages originating from *User Agents* which have already passed through the *Server Transaction* of *PM*. The arc marked *REG* in the figure signifies that registration requests are handled by the *RegistrarN*1 page. Similarly, the arcs marked *SUB* and *NTF* imply that subscription requests are handled and notification requests are generated in the *PresenceServerN*1 page. Furthermore, the *RegistrarN*1 and *PresenceServerN*1 pages are also connected with each other via the *NewREG* and *ExpREG* places. When a new registration request for a provider is received by the *Registrar*, in addition to updating its registration table and putting a timer for that message, the *Registrar* also passes that request to *Presence Server*, which moves all the subscriptions for that provider from *PendingSUB* to *SUBList* (not shown in Figure 8 since they are part of the subpages of the depicted *RegistrarAndPresenceServerN1* page). When an expiration event occurs, a token is passed through the *ExpREG* place to the *Presence Server*. If a consumer's registration expires, then its all subscriptions are removed from both *PendingSUB* and *SUBList* tables. If a provider's registration expires, all the subscriptions for that provider are moved from the *SUBList* table to the *PendingSUB* table.

3.3.4 The *RegistrarN1* subpage of *RegistrarAndPresenceServerN1* page

The *RegistrarN*1 page (Figure 9) describes the actions taken when a new registration for either a consumer or a provider arrives. The registration message is of the form $SIPReq(cid, fr, to, m, b, s, exp)$ explained earlier in the paper, where *cid* is the *callID* of the message, fr and to are the source and the destination of the message, m denotes the type of the message and is equal to *REG* for registration messages, b is the body of the message and is empty for registration messages, s is the size of the message and finally *exp* is the expiration time. When a registration message is received, the *AcceptREGSendOK* transition becomes enabled.

After firing the *AcceptREGSendOK* transition, a response is generated with the *call ID* equal to that of the original registration message. This response is then sent through the *Server Transaction* of the *Registrar* to *User Agent* 1 with the success *code* 200 [17]. Meanwhile, a timer is set for the entity's registration, whose value is taken from the *exp* parameter (expiration time) of the *REG* message. This is done by putting a timed token in the place *Timer* with a time stamp equal to the registration expiration time.

In addition, firing the *AcceptREGSendOK* transition puts a token in the place *NewREG* whenever the entity that just registered is a provider. As was mentioned in the previous section, that token will be passed to the *Presence Server* in order to update the contents of the *SubList* and *PendingSUB* tables. Notice, that according to the SIP requirement document [17], there is no need to send consumers' registration messages to the *Presence Server*, because the subscriptions are moved between the *PendingSUB* and *SubList* tables only when providers register or unregister.

Finally, firing the *AcceptREGSendOK* transition updates the content of the *REGTableN*1 by deleting any previous registration of the newly received entity and adding the *ID* of that entity in *REGTableN*1. The expression assigned to the incoming arc to the *REGTableN*1 ensures that the same entity will not appear in the *REGTableN*1 more than once, even when more than one registration or unregistration requests are received for that entity.

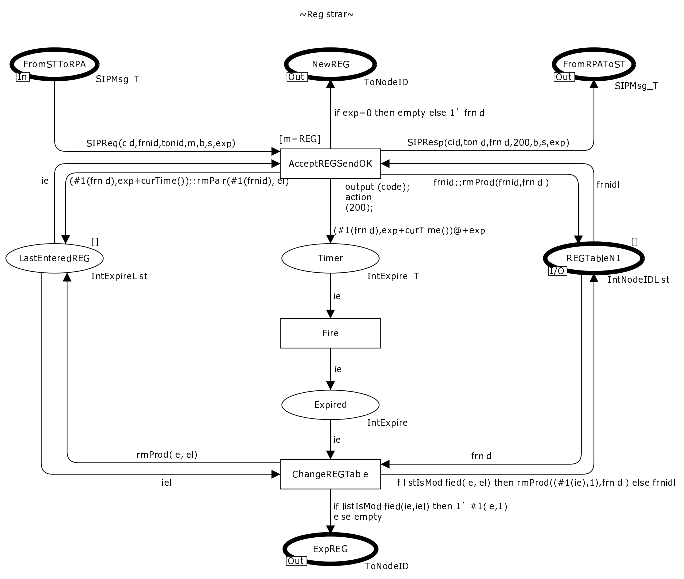

~Registrar~
FromSTToRPA
In
SIPMsg_T
NewREG
Out
ToNodeID
FromRPAToST
Out
SIPMsg_T
if exp=0 then empty else 1` frnid
[m=REG]
SIPReq(cid,frnid,tonid,m,b,s,exp)
SIPResp(cid,tonid,frnid,200,b,s,exp)
AcceptREGSendOK
iel
(#1(frnid),exp+curTime())::rmPair(#1(frnid),iel)
output (code);
action
(200);
frnid::rmProd(frnid,frnidl)
frnidl
(#1(frnid),exp+curTime())@+exp
[]
LastEnteredREG
IntExpireList
Timer
IntExpire_T
[]
REGTableN1
I/O
IntNodeIDList
ie
Fire
ie
Expired
IntExpire
ie
rmProd(ie,iel)
frnidl
ChangeREGTable
iel
if listIsModified(ie,iel) then rmProd((#1(ie),1),frnidl) else frnidl
if listIsModified(ie,iel) then 1` #1(ie,1)
else empty
ExpREG
Out
ToNodeID

Figure 9: The *RegistrarN1* subpage of *RegistrarAndPresenceServerN1* page.

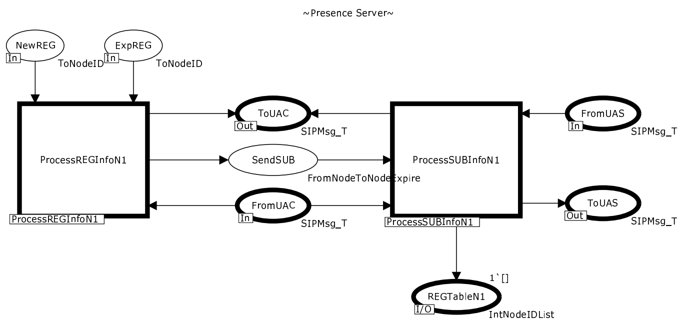


Figure 10: The *PresenceServerN1* subpage of *RegistrarAndPresenceServerN1* page.

The *LastEnteredREG* table is a list which keeps the information about consumers, providers and their registration expiration times. The incoming arc expression puts the latest information about entities and their expiration times in the *LastEnteredREG* table.

When there is a token in the place *Timer* with the time stamp less than or equal to the current time, the transition *Fire* becomes enabled, which, when fired, puts all the expired tokens in the place *Expired*. Firing the transition $ChangeREGTable$ checks whether there is an entity in the $LastEnteredREG$ table with the same ID number and expiration time as the one in the place *Expired*. If such an entity is found, then it is deleted from the *LastEnteredREG* table. The same entity is also deleted from the $REGTableN1$ and a token with the expired entity's ID number is put in the place *ExpREG* and passed to the *Presence Server*.

3.3.5 The *PresenceServerN1* subpage of *RegistrarAndPresenceServerN1* page

The page $PresenceServerN1$ (Figure 10) consists of two main components — $ProcessREGInfoN1$ and $ProcessSUBInfoN1$. $ProcessREGInfoN1$ is the page where new and expired registrations coming from the *Registrar* are handled. $ProcessSUBInfoN1$ is activated when subscription messages from $User\ Agents$ and $MCSOA\ nodes$ are received. This page is also responsible for setting timers for subscriptions and sending notification messages back to the consumers.

Observe that there is a communication between $ProcessREGInfoN1$ and $ProcessSUBInfoN1$ via the $SendSUB$ place. According to the $Presence\ Model$ requirements [3, 4], when a consumer unregisters, it must also unsubscribe. Hence, when the registration of a consumer expires, $ProcessREGInfoN1$ sends unsubscription requests (subscription request with expiration time equal to 0) for that consumer. As a result, all (consumer, provider) pairs, whose first components is the unregistered consumer, are removed from the $SUBList$ and $PendingSUB$ tables.

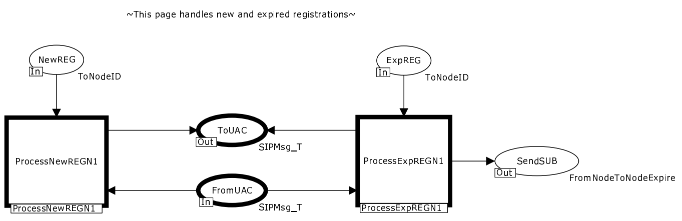


Figure 11: The *ProcessREGInfoN1* subpage of *PresenceServerN1* page.

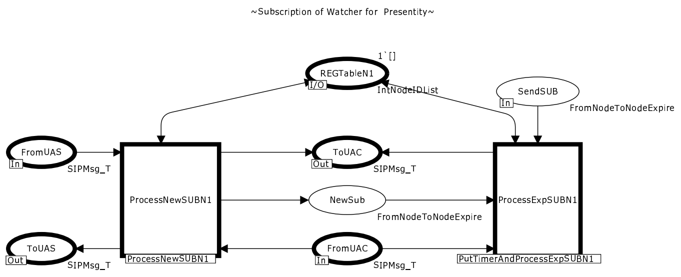


Figure 12: The *ProcessSUBInfoN1* subpage of *PresenceServerN1* page.

3.3.6 The *ProcessREGInfoN1* subpage of *PresenceServerN1* page

The page $ProcessREGInfoN1$ (Figure 11) consists of two subpages — *ProcessNewREGN1* and *ProcessExpREGN1*. The first subpage updates the *SUBList* and *PendingSUB* tables when a new registration of a provider arrives from *Registrar*. The second subpage updates those tables when an entity's registration expires. Both of these subpages are connected to the *Client Transaction* via the *ToUAC* and *FromUAC* places. When some changes occur in *SUBList* and *PendingSUB* tables, notification messages are sent to the affected consumers using those places.

3.3.7 The *ProcessSUBInfoN1* subpage of *PresenceServerN1* page

As we can see from Figure 12, the task of the $ProcessSUBInfoN1$ page, which was responsible for processing the subscription messages, is divided into two parts — processing of new subscriptions ($ProcessNewSUBN1$) and processing of expired subscriptions ($ProcessExpSUBN1$). When a new subscription message comes from the *Server Transaction* (via the place $FromUAS$), it is sent to the $ProcessNewSUBN1$ page, where the response for that message is generated and sent back to the *Server Transaction* through the place $ToUAS$. The $ProcessNewSUBN1$ page also sends notification messages about the current status of the provider.[4] The notification messages are sent to the *Client Transaction* residing on the *Presence Server* through the place $ToUAC$, and the responses for these notifications are received from *Server Transaction* through the place $FromUAC$. Furthermore, the SUB messages are passed to the $ProcessExpSUBN1$ page where a timer is created for them. As described earlier in the description of *Presence Server* page, when a consumer unregisters, it must also unsubscribe. The tokens corresponding to the unsubscription requests are generated in the $ProcessSUBInfoN1$ page and put into the $SendSUB$ place. They will get processed in the $ProcessExpSUBN1$ subpage.

3.3.8 The *Client and Server Transactions* Component

Non Invite Client and *Server Transactions*, depicted on Figures 13 and 14, are the most essential parts of SIP compliant transactions. Each request in an SIP compliant model should pass through the *Client Transaction* of one side and be received by the *Server Transaction* of the other side, where the response for the request should be generated and forwarded back along the path through which the request came.

For a message to travel from its source to its destination, it must start at a *User Agent* also known as *Transaction User* (TU), which corresponds to the source of the message. Then it must be forwarded to the *Client Transaction* on the source side, from *Client Transaction* to *Transport*, then to the *Server Transaction* on the destination side, which in its turn finally forwards the message to the destination TU. The *Client* and

[4]Currently this message informs whether the provider is in the *REGTable* or not and also contains the remaining expiration time for that subscription as per SIP specification.

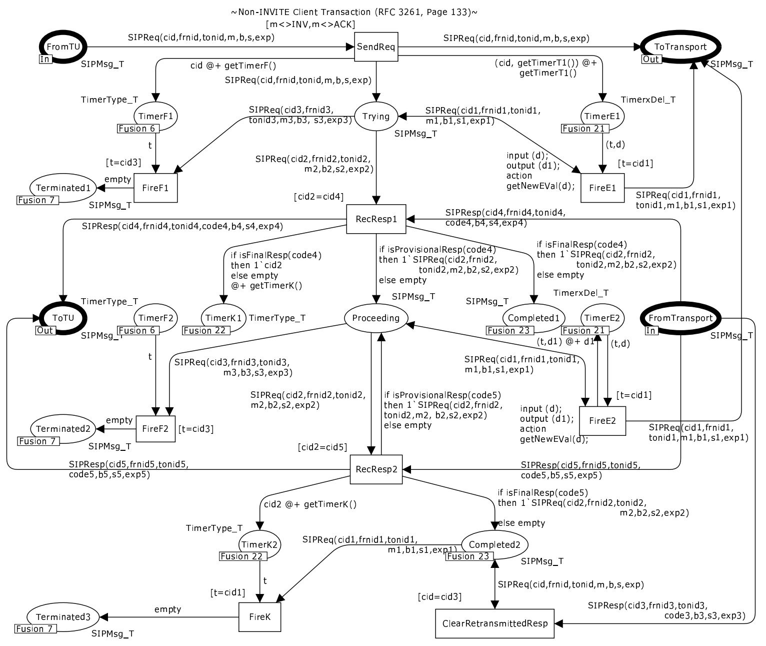


Figure 13: The *ClientTransactionN*1 page.

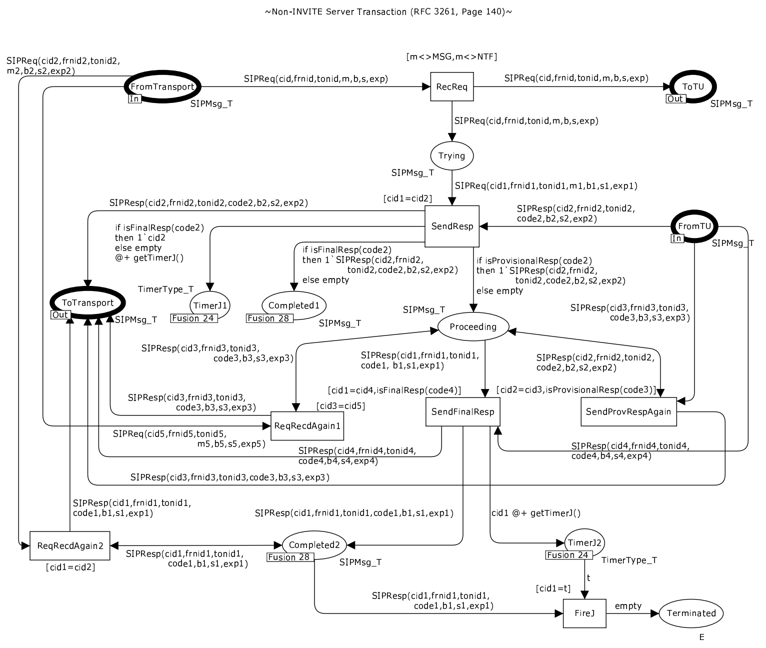


Figure 14: The *ServerTransactionN*1 page.

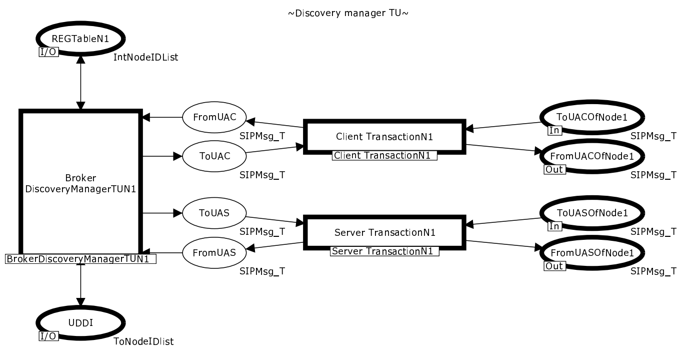


Figure 15: The *BrokerDiscoveryManagerN1* subpage of *MCSOANode1* page.

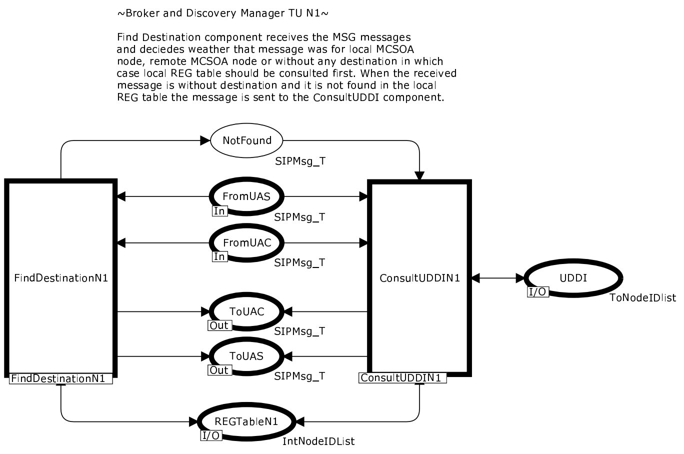
Server Transactions accomplish their task of delivering requests and responses using state machines, whose requirements can be found in [17]. The *Client* and *Server Transaction* pages are essentially the result of converting those state machines into their corresponding CPN models. The states of the machine get mapped to places in the model with identical names. Note that the *Client* and *Server Transactions* pages are used several times throughout the *Discovery* model. The *port* places *FromTransport*, *ToTransport*, *FromTU* and *ToTU* get populated when messages are passed or received to or from *Transport* or *TU*. Port-socket connection determines the flow in each case when the *Client* and *Server Transactions* are used and the connections can be different from the orientation used in Figures 13 and 14. For example, in Figure 13 the port *FromTU* is connected to the socket *ToUAC* place in Figure 7, the port *ToTU* is connected to the socket *FromUAC*, the port *ToTransport* is connected to the socket *FromUACOfNode1* and finally the port *FromTransport* is connected to the socket *ToUACOfNode1*. Although *Client* and *Server Transactions* used throughout the model have completely identical structures, we could not simply use the currently available pure cloning because when cloning a page that contains local fusion sets, new fusion sets are not created for the cloned page. This makes pages dependent on each other, which is not desirable in our case.

3.3.9 The *BrokerDiscoveryManagerN1* subpage of *MCSOANode1* page

The *BrokerDiscoveryManagerN*1 page (Figure 15) is activated any time a *MSG* request or a response for a *MSG* request is received. The page *BrokerDiscoveryManagerTUN*1 is responsible for identifying the type (with or without a specific service end point) of the message and handling it accordingly. *DM* also contains a *Client Transaction* in order to send or forward requests, and a *Server Transaction* in order to send or forward responses for those requests. The *ToUACOfNode*1, *FromUACOfNode*1, *ToUASOfNode*1, *FromUASOfNode*1 places connect the *MCSOA node* 1 with the *Network and Routing* component of the *Discovery* model. *FromUAC*, *ToUAC*, *ToUAS*, *FromUAS* places are considered local and are designated to connect the *BrokerDiscoveryManagerTUN*1 subpage with the *Client* and *Server Transactions*.

3.3.10 The *BrokerDiscoveryManagerTUN1* subpage of *BrokerDiscoveryManagerN1* page

The *BrokerDiscoveryManagerTUN*1 page (Figure 16) is responsible for handling two types of tasks. The first one is to find the destination of the *MSG* messages, i.e. discover whether the received *MSG* message was intended for the local *MCSOA node* 1, or for the remote *MCSOA node* 2, or has unspecified destination. This is implemented in the *FindDestinationN*1 subpage. The second task is to pass an *MSG* message without a specific service end point through the discovery process if its provider was not found in the local *REGTable*. This part is handled by the *ConsultUDDIN*1 subpage. Observe, that the place *UDDI* is connected only with the *ConsultUDDIN*1 transition. This reveals the fact that the *UDDI* should be consulted only in case of the *Discovery* process. The place *NotFound* ensures the connection between the two described subpages. When a *MSG* message without a specific service end point is received and the provider is not found in the *REGTableN*1, a token is put in the *NotFound* place and passed to the *ConsultUDDIN*1 subpage. The places *FromUAC*, *ToUAC*, *ToUAS*, *FromUAS* are designed to pass the tokens for *MSG* requests and their responses to *FindDestinationN*1 and *ConsultUDDIN*1 subpages.


~Broker and Discovery Manager TU N1~

Find Destination component receives the MSG messages and deciedes weather that message was for local MCSOA node, remote MCSOA node or without any destination in which case local REG table should be consulted first. When the received message is without destination and it is not found in the local REG table the message is sent to the ConsultUDDI component.

Figure 16: The *BrokerDiscoveryManagerTUN1* subpage of *BrokerDiscoveryManagerN1* page.

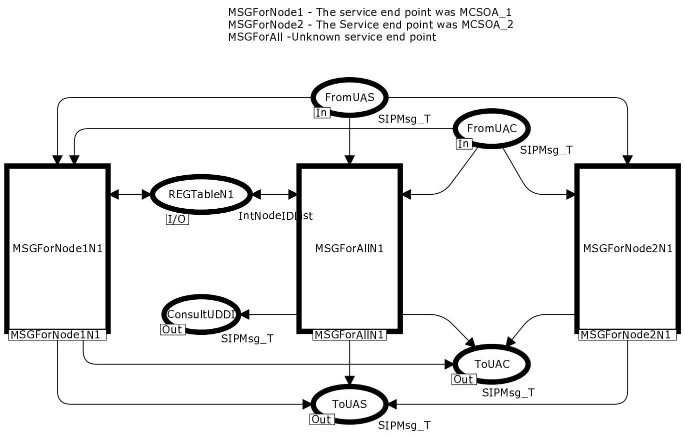

MSGForNode1 - The service end point was MCSOA_1
MSGForNode2 - The Service end point was MCSOA_2
MSGForAll -Unknown service end point

Figure 17: The *FindDestinationN1* subpage of *BrokerDiscoveryManagerTUN1* page.

3.3.11 The *FindDestinationN1* subpage of *BrokerDiscoveryManagerTUN1* page

The $FindDestinationN1$ page (Figure 17) describes how the destination of the MSG message is determined when it is received by $MCSOA$ *node* 1. The destination parameter of the MSG request is a pair whose first component is the ID of the provider for which the MSG message is intended and second component represents the end point of the MSG message and can be either 0, 1 or 2. According to the convention used in the *Discovery* model, if the value of the end point is 1, then the MSG message came for $MCSOA$ *node* 1. In this case, the local $REGTable$ is consulted to find whether the provider exists. If the provider is not found, a response with *code* of 480 is generated. Otherwise the message is forwarded to the $User\ Agent$ and the $MCSOA$ *node* waits for a response from that $User\ Agent$. This functionality is handled in the subpage $MSGForNode1N1$. If the value of the flag is 2, $MCSOA$ *node* 1 forwards the message to $MCSOA$ *node* 2 — the intended recipient — without consulting either the local $REGTable$ or the $UDDI$. $MCSOA$ *node* 1 then waits for a response from $MCSOA$ *node* 2. This case is handled by the $MSGForNode2N1$ subpage. Finally, when the value of the end point is 0, the provider is unspecified and must be discovered. According to the *Discovery* model requirement document [14], in this case the local $REGTable$ is consulted first. If the search is unsuccessful, the $UDDI$ is consulted next. If the $UDDI$ finds that the provider is available on $MCSOA$ *node* 2, the message is forwarded to $MCSOA$ *node* 2. Otherwise, a response with the *code* 480 is generated and sent back to $User\ Agent$ 1. The $MSGForAllN1$ subpage is responsible for handling this last case.

3.3.12 The *ConsultUDDIN1* subpage of *BrokerDiscoveryManagerTUN1* page

The main task of the $ConsultUDDIN1$ page (Figure 18) is to consult the $UDDI$ when a token corresponding to an MSG message appears in the place $NotFound$, implying that the provider was not found in the local $REGTable$ and the message was forwarded to the $ConsultUDDIN1$ page. Firing the transition

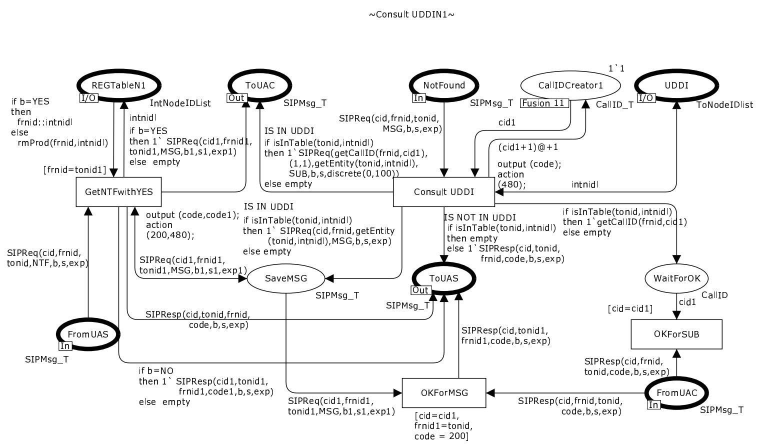


Figure 18: The *ConsultUDDIN1* subpage of *BrokerDiscoveryManagerTUN1* page.

$ConsultUDDIN1$ tries to find a pair in the $UDDI$ whose first component is the provider appearing in the MSG message. If such a pair is not found then a response with *code* 480 is generated and a token corresponding to the response is put in the place $ToUAS$. Otherwise, the second component of the pair from the $UDDI$ will indicate the $MCSOA\ node$, on which, according to the $UDDI$, the provider might be registered. In this case, a SUB message is sent in order to subscribe $MCSOA\ node$ 1 for the provider on the other $MCSOA\ node$. When a response for SUB requests is received, the place $FromUAC$ gets populated and the transition $OKForSUB$ becomes enabled. Firing it will remove the response token from the place $FromUAC$ as well as the token with the same value as the *cid* parameter of the response message from the place $WaitForOK$.

When a notification message is received from the other $MCSOA\ node$ by $MCSOA\ Node$ 1, the transition $GetNTFwithYES$ becomes enabled. Firing this transition will check the value of the body (b) parameter of the notification message. If the value is NO, it means that the provider was not registered on the other $MCSOA\ node$ and the information derived from the $UDDI$ was incorrect. If the body parameter is YES, firing the $GetNTFwithYES$ transition will forward the MSG message to the other $MCSOA\ node$. In both cases firing the transition $GetNTFwithYES$ will generate and send a response for the notification message to the other $MCSOA\ node$.

3.4 The Network And Routing Component

The $Network\ and\ Routing$ component is the second main component of the $Discovery$ CPN model, and is responsible for handling all the communication between the $User\ Agents$ and the $MCSOA\ nodes$ forming the $MCSOA$ fabric (see Section 2). In this section we will go into the details of the design of this component. As before, the first subsection will give an overview of the component, while each of the remaining subsections will describe a particular module of the component. Due to space limitations, our description will be high level, and many details will be omitted.

3.4.1 The *NetworkAndRouting* subpage of *Top* page

The $NetworkAndRouting$ page (Figure 19) is responsible for the communication between each $User\ Agent$ and its corresponding $MCSOA\ node$, as well as that between the two $MCSOA\ nodes$. In our model, the subpage $Node1UA$ ensures the communication between $User\ Agent$ 1 and $MCSOA\ node$ 1, the subpage $Node1Node2$ handles the communication between the two $MCSOA\ nodes$, and finally the subpage $Node2UA$ enables the communication between $User\ Agent$ 2 and $MCSOA\ node$ 2. The places around the substitution transitions in Figure 19 are port-socket places and get populated from the $Client$ and $Server\ Transactions$ of the corresponding $Discovery$ model component/subpages.

3.4.2 The *Node1UA* subpage of *NetworkAndRouting* page

The $Node1UA$ page (Figure 20) models how SIP requests and SIP responses are passed between outgoing places of $User\ Agent$ 1 and the incoming places of $MCSOA\ node$ 1. Notice that the direction of the messages are determined by the fr and *to* parameters of the request message. Also, the conditions imposed on the transitions ensure that the messages are forwarded along the proper direction.

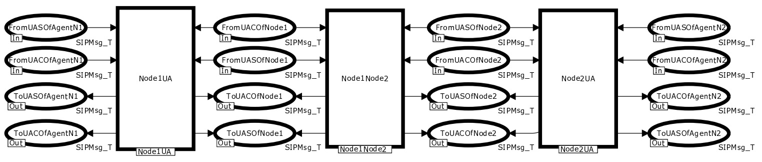


Figure 19: The *NetworkAndRouting* subpage of *Top* page.

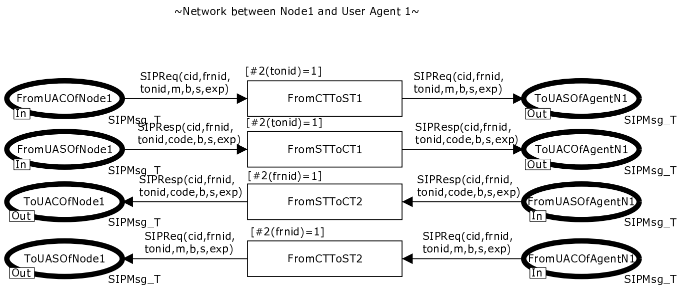


Figure 20: The *Node1UA* subpage of *NetworkAndRouting* page.

3.5 The User Agent Components

The final main components of the *Discovery* model are the *User Agents*. In the following subsections we will give a detailed description of the design of the *User Agent* 1 component. The *User Agent* 2 is analogous.

3.5.1 The *SIPUserAgentsN1* subpage of *Top* page

The page *SIPUserAgentsN*1 shows the high-level functionality of the *User Agent* 1 component. As can be seen from Figure 21, *User Agent* 1 consists of the two main types of entities — consumers and provides. The four places on the page are used to connect *User Agent* 1 to the *Network and Routing* component. Notice, that while consumers can send *REG*, *SUB* and *MSG* messages and receive *NTF* messages, providers can only send *REG* messages. The arcs in the figure that do not have auxiliary text on them are designated for receiving responses for *REG*, *SUB* and *MSG* messages and sending responses for *NTF* messages.

3.5.2 The *ConsumerN1* subpage of *SIPUserAgentsN1* page

Figure 22 depicts the structure of the page *ConsumerN*1. Its three subpages — *ClientTransactionN*1, *ServerTransactionN*1 and *ConsumerTransactionUserN*1 — are responsible for sending *REG*, *SUB*, and *MSG* messages to *MCSOA Node* 1. As was discussed above, the *ClientTransactionN*1 and *ServerTransactionN1* pages ensure *SIP* compliant communication. The page *ConsumerTransactionUserN*1 is the place where all the messages are generated and is also responsible for matching the responses for these messages to their sources. The *FromUACOfAgentN*1, *ToUACOfAgentN*1, *ToUASOfAgentN*1 and *FromUASOfAgentN*1 places get populated when a message request moves between the *Client* and *Server Transactions* and the *Network and Routing* component. On the other hand, the *FromTUToCT*, *FromCTToTU*, *FromSTToTU*, *FromTUToTST* places store tokens that move between the *Client* and *Server Transactions* and the *ConsumerTransactionUserN1* transition.

3.5.3 The *ConsumerTransactionUserN1* subpage of *ConsumerN1* page

The main task of the *ConsumerTransactionUserN*1 page (Figure 23) is to send *REG*, *SUB* and *MSG* messages. These functionalities are implemented in the *SendREGN*1, *SendSUBN*1, *SendMSGN*1 pages respectively. When a consumer sends a registration message and gets the response for it, a token with the value equal to the *ID* of the newly registered consumer is put in the *RegisteredCons* place. This is done to ensure that only registered consumers can send *SUB* and *MSG* messages. The thick black ovals correspond to the port-socket places connecting the *ConsumerTransactionUserN*1 page with the *Client* and *Server Transactions* of the consumer side.

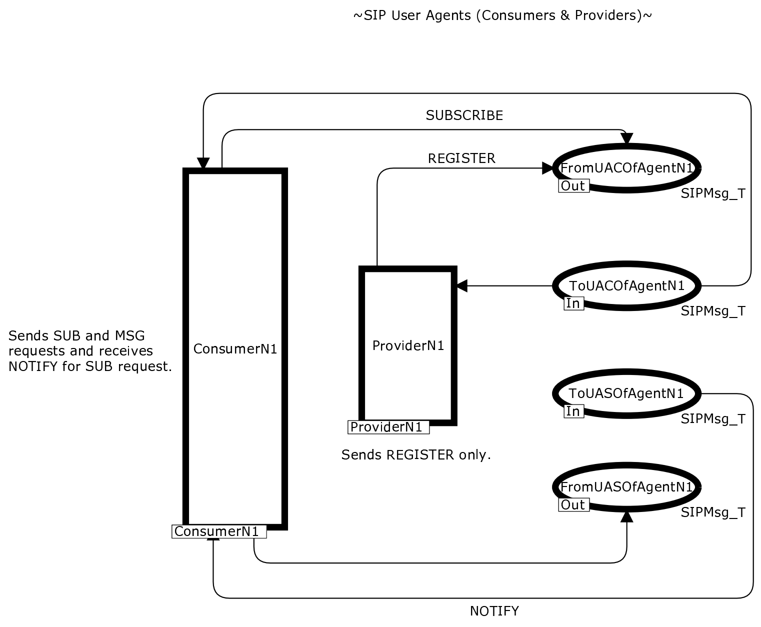


Figure 21: The *SIPUserAgentsN1* subpage of *Top* page.

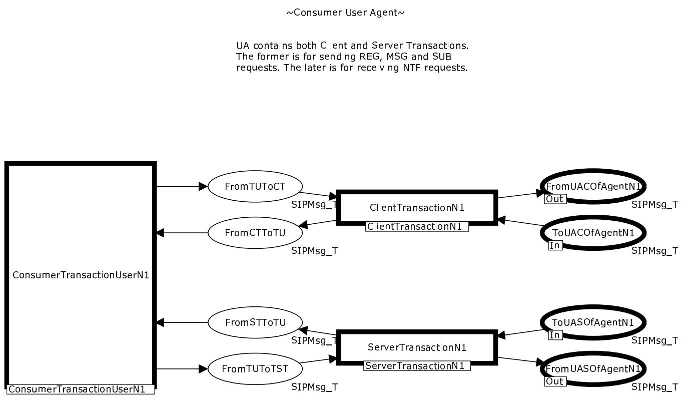


Figure 22: The *ConsumerN1* subpage of *SIPUserAgentsN1* page.

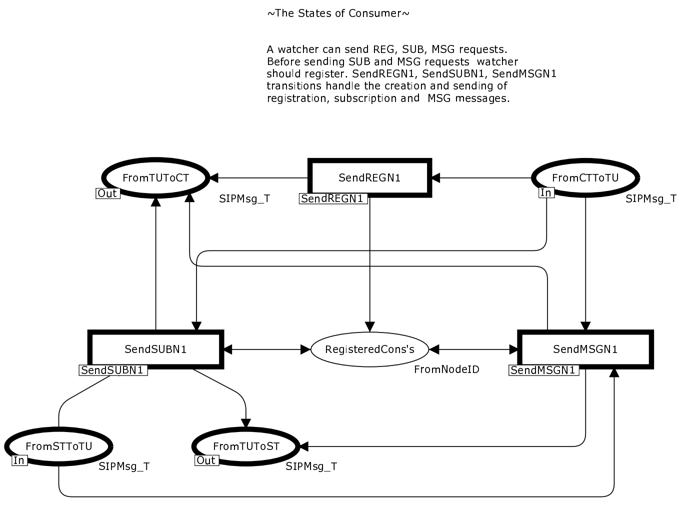
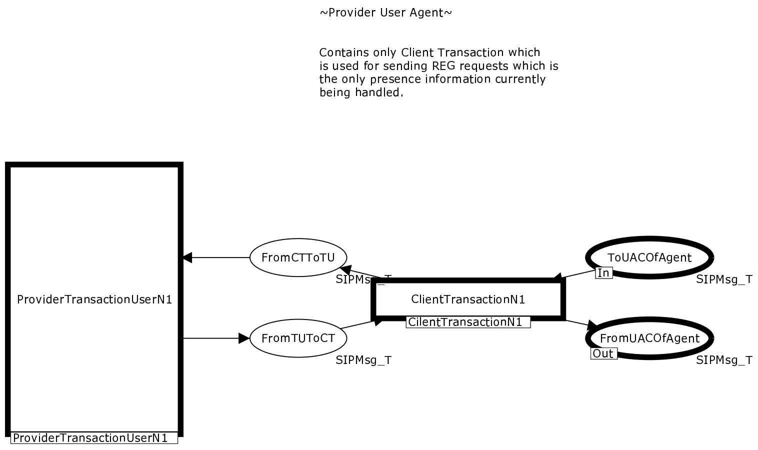

~The States of Consumer~
A watcher can send REG, SUB, MSG requests. Before sending SUB and MSG requests watcher should register. SendREGN1, SendSUBN1, SendMSGN1 transitions handle the creation and sending of registration, subscription and MSG messages.
FromTUToCT
Out
SIPMsg_T
SendREGN1
SendREGN1
FromCTToTU
In
SIPMsg_T
SendSUBN1
SendSUBN1
RegisteredCons's
FromNodeID
SendMSGN1
SendMSGN1
FromSTToTU
In
SIPMsg_T
FromTUToST
Out
SIPMsg_T

Figure 23: The *ConsumerTransactionUserN1* subpage of *ConsumerN1* page.


~Provider User Agent~

Contains only Client Transaction which is used for sending REG requests which is the only presence information currently being handled.

Figure 24: The *ProviderN1* subpage of *SIPUserAgentsN1* page.

3.5.4 The *ProviderN1* subpage of *SIPUserAgentsN1* page

The *ProviderN*1 page (Figure 24) contains two subpages — *ProviderTransactionUserN1* and *ClientTransactionN1* — which together provide the functionality of the page. The *ClientTransactionN*1 page is responsible for transporting registration messages and receiving responses for them. The *ProviderTransactionUserN*1 page is responsible for generating the *REG* messages for providers. The two places *ToUACOfAgent* and *FromUACOfAgent* connect the *ClientTransactionN*1 page with the *Network and Routing* component.

This concludes our description of the *Discovery* model. The initial validation of the model was done by executing several *Discovery* related *MCSOA* use cases. These use cases cover different scenarios that may trigger *Discovery process*. For example one of the use cases covers the scenario of service re-discovery prevention if the service has been previously discovered. Use cases were validating by analyzing the output generated by the various monitoring facilities of the *CPN Tools*.

4 Conclusions and Future Work

We presented details of a CPN model of a SIP-based Discovery process that underlies a multi-channel service oriented architecture geared towards defense needs. The discovery process is a crucial aspect of this architecture since it brings in service guarantees, availability awareness, and dynamic service discovery as well as a level of fault-tolerance.

The basic components of a SIP-based architecture are *Client Transactions* and *Server Transactions* that are executed by the associated SIP state machines [17]. These state machines translate to a CPN model of moderate size and complexity [7]. Any model of a process or protocol that employs SIP needs to duplicate these components. Furthermore, a model that is built on the top of such models may itself need to duplicate these higher-level components. Although CPN Tools has facility for deep cloning, we realize what is needed, especially from a practical and user convenience point of view, is a parameterization and module building facility. For example, moving from Presence to Discovery, we found that the types of the underlying transactions needed to be enhanced/extended. Support for parameterization facility in the CPN Tools along the lines of [12] would certainly simplify this task. In addition, inclusion of standard modules based on [16] would be an added plus for the users of the CPN Tools.

Our initial verification and validation of the model was done using a simulation based approach outlined in [18]. CPN Tools already contains many features and facilities that makes this task easier to carry out, especially for a third-party independent validation team that may not be very familiar with CPN. In fact, we have already made extensive use of the monitoring and performance analysis facilities provided by the CPN Tools [10, 11]. Some issues that were identified have been communicated to the CPN Support Team. Our future plans include making use of state-space facility as well as Temporal Logic for state spaces to establish certain desirable properties of the *Discovery protocol*. In fact, thanks to a well-defined semantics and formalized notions in CPN, we have been able to verify formally, for example, certain SIP specific properties using induction on number of transition firings. However, there were some problems that we came cross during the creation of the *Discovery Model*. Particularly, we were not able to fully exploit the “page instance” creation facility of the *CPN*

Tools because, in our experience, page instances did not mix well with (local) *fusion sets*. In addition, all page instances shared the same initial marking which were undesirable in some situations.

Our present Discovery model gives details of the process in a two-node fabric scenario. Our future plans include scaling the fabric. However, simply duplicating the nodes and interconnecting them would lead to a net that may become unwieldy in various ways. Our current thought is to get enough understanding of the discovery process using the current model so that we can abstract away the essential features of it and can then define a node as a token with a suitable CPN compound color set. This will allow us to capture fabric-level interactions in a suitable CPN built on top.

Acknowledgements

We would like to thank the Villanova ARCES Research team members Robert Beck, Peter DePasquale, Daniel Joyce, Anany Levitin, and Thomas Way as well as the Gestalt ARCES Research team members Elzbieta Dziembowski and James Solderitsch for many fruitful discussions and inputs on this undertaking. Our thanks are also due to Brian O'Neill, Fred Smith, and Eric Wiseblatt of the MCSOA Development team for helping us sort through MCSOA details. Finally, we would like to extend our thanks to Jim Stogdill of Gestalt LLC and Kevin Miller of the MITRE Corporation for their advisory role on the project.

This research was supported in part by the Air Force Materiel Command (AFMC), Electronic Systems Group (ESG) under contract number FA8726-05-C-0008 . The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of USAF, AFMC, ESC, or the U.S. Government.

We are also very grateful to the anonymous reviewers for providing many helpful comments and suggestions for improving the contents of this paper.

References

[1] B.Han and J.Billington. Experience with Modelling TCP's Connection Managment Procedures with CPNs. In Kurt Jensen, editor, *Proceedings of the Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 57–76. Department of Computer Science, University of Aarhus, 2004.

[2] G. Camarillo. *SIP Demystified.* McGraw-Hill, 2002.

[3] M. Day, S. Aggarwal, G. Mohr, and J. Vincent. *Instant Messaging/Presence Protocol Requirements, RFC 2779.* IETF, Feb 2000.

[4] M. Day, J. Rosenberg, and H. Sugano. *A Model for Presence and Instant Messaging, RFC 2778.* IETF, Feb 2000.

[5] J.C.A. de Figueiredo and L.M. Kristensen. Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols. In Kurt Jensen, editor, *Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN*, pages 21–40. Department of Computer Science, University of Aarhus, 1999.

[6] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005.

[7] A. Hayrapetyan. CPN Presence Model. In ARCES Software Design Document, Release 3, Villanova University and Gestalt LLC, August 2006.

[8] K. Jensen. *Coloured Petri Nets—Basic Concepts, Analysis Methods and Practical Use*, volume 1 of *Monographs in Theoretical Computer Science, An EATCS Series.* Springer-Verlag, 2nd edition, 1996.

[9] Lars Michael Kristensen, Jens Bæk Jørgensen, and Kurt Jensen. Application of Coloured Petri Nets in System Development. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 626–685. Springer, 2004.

[10] B. Lindstrøm and L. Wells. Performance Analysis Using Coloured Petri Nets. Master's thesis, University of Aarhus, May 1999.

[11] B. Lindstrøm and L. Wells. Towards a monitoring framework for discrete event system simulations. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, 2002.

[12] T. Mailund. Parameterised Coloured Petri Nets. In Kurt Jensen, editor, *Proceedings of the Workshop on Practical Use of Colored Petri Nets and Design/CPN*. Department of Computer Science, University of Aarhus, October 1999.

[13] CPN Project. Examples of industrial use of CPN-nets. Available online at `http://www.daimi.au.dk/CPnets/intro/example_indu.html`, last accessed 2006/08/08, Jan 2006.

[14] MCSOA Project. Multi-Channel Service Oriented Architecture. Internal Report, Gestalt LLC, April 2005.

[15] NESI Project. Net-Centric Implementation, part 1: Overview (version 1.3.0). Available online at `http://nesipublic.spawar.navy.mil/docs/part1`, last accessed 2006/08/08, June 2006.

[16] N. Mulyar and W.M.P. van der Aalst. Patterns in Colored Petri Nets. BETA Working Paper Series, WP 139, Eindhoven University of Technology, Eindhoven, 2005.

[17] J. Rosenberg, H. Shulzrine, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. *SIP: Session Initiation Protocol, RFC 3261*. IETF, June 2002.

[18] R. D. Sargent. Verification and validation of simulation models. In S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, editors, *Proceedings of the 2003 Winter Simulation Conference*, 2003.

A Table of Acronyms and Abbreviations

Term	***Description***
AFMC	Air Force Materiel Command
ARCES	Applied Research for Computing Enterprise Services
CPN	Coloured Petri Nets
DM	Discovery Manager
DoD	US Department of Defense
ESC	Electronic Systems Group
GIG	Global Information Grid
HTTP	Hypertext Transfer Protocol
MCSOA	Multi-Channel Service Oriented Architecture
MSG	Message
NESI	Net-Centric Enterprise Solutions for Interoperability
NTF	Notification
PendingSUB	Pending Subscription Table
PM	Perspective Manager
REG	Registration
REGTable	Registration Table
SAP	Service Access Point
SIP	Session Initiation Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SUB	Subscription
SUBList	Subscription List
TU	Transaction User
UDDI	Universal Description, Discovery and Integration

Design of Clearing and settlement operations: A case study in business process modelling and evaluation with Petri nets[1]

P.M. Kwantes, October 2006

Abstract

This paper describes the practical application of Petri nets to business process modeling in a sector of financial industry concerned with the "Clearing and settlement of capital market transactions" by means of a case study. The Clearing and settlement process, allows two competing design alternatives. Two Petri net models are described representing these two alternative process designs. Petri nets have an intuitive graphical representation and allow animation which is used for validating the two models by business experts familiar with the Clearing and settlement process. After validation of the two models they are executed giving two sets of measurements which are compared to demonstrate the relative performance characteristics of the two design alternatives. In the final section the choice for Hierarchical Timed Coloured Petri nets and CPN-Tools is discussed in the light of the experience and findings of the case study.

1 Introduction

1.1 Motivation

The main purpose of the case study described in this paper is to show that Petri net models can be applied in the financial services industry to select the best process design from competing alternatives, before starting implementation. This might help to reduce the risks and costs associated with process innovation in an area that is changing rapidly. The process examined in the case study is the "Clearing and settlement" process. The purpose of this process, in a nutshell, is to transfer financial assets between two parties that have closed a deal on an exchange or on a so called "over the counter" market. The parties involved can be many miles apart, and many intermediate institutions might be involved. As it turns out the problems involved are to some extent similar to the problems involved when transferring goods between different locations. So Clearing and settlement of securities can be seen as a kind of "capital market logistics". The application of Petri nets to the modelling of the logistics of physical goods is described in [1]. The question can be raised whether the perceived analogy merits the use of a similar approach. More specifically, the case study will be used to find answers to the following questions:

[1]This paper is based on [10]. The description of the case study is for a large part based on experience of the author with the subject matter while working as a consultant for the securities wholesale department at ING Bank.

1. is the language of Hierarchical Timed Coloured Petri nets (HTCPN) sufficiently expressive to model the critical properties of the Clearing and settlement process ?
2. are the possibilities for graphical representation and running simulations of HTCPNs helpful in making the models of the Clearing and settlement process accessible and understandable for a non-technical audience and thus in validating the model ?
3. is it feasible to decide which of two alternative designs of the Clearing and settlement process, described as HTCPNs, perform the best in terms of their critical properties?

1.2 Outline of the text

In section 2 the Clearing and settlement process, is introduced. As it turns out, there are two different ways to design this process. In section 2.2 it is explained what the differences between these two process designs are and why they might cause differences in performance. Section 3 describes a dataflow diagram representing the first design alternative and the subsequent translation of this dataflow diagram into a HTCPN. The Petri net representing the second design alternative is developed in a similar way and is described in [10]. In section 4 the two Petri net models are executed to demonstrate the relative performance characteristics of the design alternatives. Before this can be done, the relevant performance criteria must be defined and also the measurements that are required to calculate these performance criteria. These are described in section 4.1. The results of executing the models are presented in section 4.2. Finally in section 5 concluding remarks are made regarding the research questions posed in the introduction and the use of CPN Tools.

2 Designing the Clearing and settlement process

2.1 Introduction

The Clearing and settlement process is concerned with the processing of transactions on secondary capital markets. While primary capital markets are involved in the creation or issuing of financial assets, secondary capital markets are markets where already existing financial assets are traded. The exchange of financial assets in secondary markets is a process that is composed of a number of clearly defined stages. The first stage is the "trading stage", where market participants try to close a deal. The next stage is the "clearing stage", in which the accountability for the exchange of funds and financial assets are determined. This might, for instance, involve the confirmation between the trading parties of the conditions of a transaction, or, for efficiency reasons, the netting of several transaction over a longer period, to reduce the actual exchange of funds and assets. A third stage is the "settlement stage", which involves the actual exchange of funds and assets. After the settlement stage, if all goes well, the financial asset involved is in the possession of the rightful owner. In most cases the safe keeping of the asset is left to a specialized financial institutions called a **Custodian**. A number of things can go wrong after the completion of the first stage and before the completion of the third stage. A worst case scenario would be for instance that a deal is closed, and that the buyer of a financial asset already transferred the promised funds but never receives the assets he bought. Another problem would arise if the deal is closed, but one of the parties backs out altogether, so nothing is exchanged at all. When these kind of problems occur, individual parties might incur losses. Furthermore the loss of one party might cause a domino effect and spread throughout the entire system.

To reduce the risk that this happens, a number of institutional arrangements have been created. One is the regulation and formalization of capital markets in the form of **Exchanges**. Related to these Exchanges there often is an institution which performs the role of a **Central Counterparty** (CCP) to all other market participants. This means that when two parties close a deal, the deal is directly "split up" (the official term is "novation") in two deals: each of the two parties in the original deal ends up with a deal with the CCP. The CCP guarantees the finishing of phases two and three, clearing and settlement, of the exchange of assets. However, this introduces another potential risk: that of default of the CCP. To reduce this risk another arrangement is introduced: only a few specially selected, financially strong, firms are allowed to trade with the CCP. These firms, called **Clearing Member Firms** (CMF), are an intermediary between the regular market participants, called **Trading Member Firms** (TMF), and the CCP. A firm can only become a trading member on the Exchange if it is sponsored by a CMF. Sponsoring means that the CMF guarantees all deals made by the TMF vis a vis the CCP.

Figure 1 gives an impression of the relationships between the institutions that are related to the Exchange as described above. The upper part of the figure shows that in the trading phase TMF 1 sells an amount of 100 securities to TMF O (short for "other TMF"). The lower part of the figure shows the clearing and settlement phase for this trade. In this clearing and settlement phase the trade undergoes a number of transformations. The first transformation, as already mentioned above, involves a split up of the trade into two trade legs: one between TMF 1 and the CCP and another between TMF O and the CCP. The first tradeleg implies a promise of TMF 1 to deliver [2] an amount of 100 securities to the CCP. The second implies a promise of the CCP to deliver a 100 securities to TMF O. The second transformation involves a split up of each tradeleg in an obligation between the TMF and the CMF on the one hand and an obligation between the CMF and the CCP. In this example this means that CMF S (short for "sponsoring CMF") and CMF O are placed between the CCP

[2] "Delivery" of securities not so long ago involved the transport and delivery of "physical documents". Nowadays delivery usually means creating the correct book-entries in the system of the Custodian by sending an electronic message

and the TMFs. This final situation is shown in the lower part of figure 1 where TMF 1 must deliver 100 to Clearing Member Firm S which in turn must deliver 100 to the CCP. Subsequently the CCP must deliver these securities to CMF O which in turn must deliver to TMF O. We will call the delivery of securities of TMF 1 to CMF S “phase D1”, and the delivery of securities of CMF S to the CCP “phase D2” of the Clearing and settlement process.

There is also trading on secondary capital markets done outside the Exchanges. This means that there is subdivision in the secondary capital market between the Exchanges mentioned above, where the arrangements, like a CCP, are present, and an “over-the-counter secondary capital market” or **OTC-market** for short, where they are not or only in some rudimentary form.
Two more institutions need to be mentioned to complete this short introduction: the **Lending Firm** and the **Central Securities Depository** (CSD). The Lending Firm is involved in the borrowing of securities from share owners in order to lend these securities to others that for some reason temporarily want to own these shares. One reason might be that a trader sold shares that he did not possess, a practice called “short selling”. The Lending Firm lends securities to the short selling trader, to enable delivery of the securities to the buyer of the securities in time. Not delivering in time, especially if the securities are sold on an Exchange, can cause the counterparty to claim damages or impose a penalty. The CSD is like a Central Bank for securities. As a rule, each country has its own CSD, and all securities issued in that country are kept in the vaults

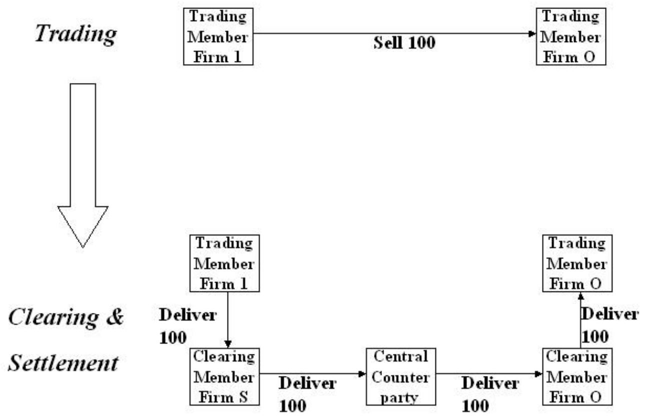


Figure 1: Relations between CCP, CMF and TMF

of the CSD. A Custodian that keeps securities for its clients, the actual share owners, will in turn keep those securities on an account with a CSD. There is an extensive amount of literature on the institutional arrangements underlying Clearing and settlement and it's basic mechanism see e.g. [11] or [6]. More detailed specifications of the process, as it is implemented by LCH Clearnet, a large European Central Counterparty, can be found in [2], [3] and [4]. Specification of CSD operations, as implemented by Euroclear Nederland, the dutch CSD, is described in [5]. The design of the Clearing and settlement process has undergone a lot of changes in the course of time, and this redesign is still an ongoing process. Economic growth, market forces, technological development and intervention by government and regulatory authorities are the main driving forces behind this process. (e.g. p. 541 in [9]).[3]

2.2 Two design alternatives for the Clearing and settlement process

It turns out that there are different ways to design the Clearing and settlement process. Processing that takes place within the market infrastructure (i.e. the Exchange, OTC-Market,CCP and CSD) is however assumed to be fixed and it provides the boundary of the design space within which we can choose alternatives. One of the dimensions of this design space is the degree of freedom for market intermediaries (i.e. CMF, Custodian and Lending Firm) to **use** the market infrastructure. The main differences between the two design alternatives examined in this paper are:

1. differences in the account structures used by the market intermediaries
2. differences in internal processing within market intermediaries

As argued in section 1.1, the Clearing and settlement process can be considered as a kind of capital market logistics involved in the transfer of assets. Like a warehouse can be used for storing goods an account can be considered a facility for storing assets. The number of warehouses and their location can influence the efficiency and effectiveness of the logistical process. Similarly the number and location of accounts in the Clearing and settlement process can influence the efficiency and effectiveness with which assets are transferred.

The main difference in **account structure** between the two design alternatives we will consider is that in the first design alternative **one account** will be held at the CSD. This account will be owned by the CMF and the Custodian. This does not imply that there are two account owners, because we assume that the account owner is one legal entity that operates both the CMF and the

[3]The description of the Clearing and settlement process just given is certainly a simplification of reality and leaves out aspects like the exchange of money that is associated with the exchange of securities, the handling of corporate actions, the pledge of collateral. This simplification is however not expected to have an important influence on the results of the analysis.

Custodian. In the second design alternative **two accounts** will be held at the CSD, one by the CMF and one by the Custodian. This extra account allows better *separation* between the process of the Lending Firm and the process of the Clearing Member Firm[4] but will add *an extra step* in the process of transferring assets. In both design alternatives each TMF will hold one account at the Custodian.

The differences in account structures between the two design alternatives go hand in hand with different **processing** within market intermediaries. The basic difference in processing is that if assets can be kept on two different accounts it might be the case that they are not kept on the account on which they are needed. So the process must monitor how much securities are needed on which account and transfer securities from accounts with an excess balance to accounts on which the balance is too low. The relationship between account structure and internal processing in the two design alternatives is illustrated by figure 2 below. The upper part of the figure represents the account structure and processing in design alternative one and the lower part the account structure and processing in design alternative two. Each account structure consists in

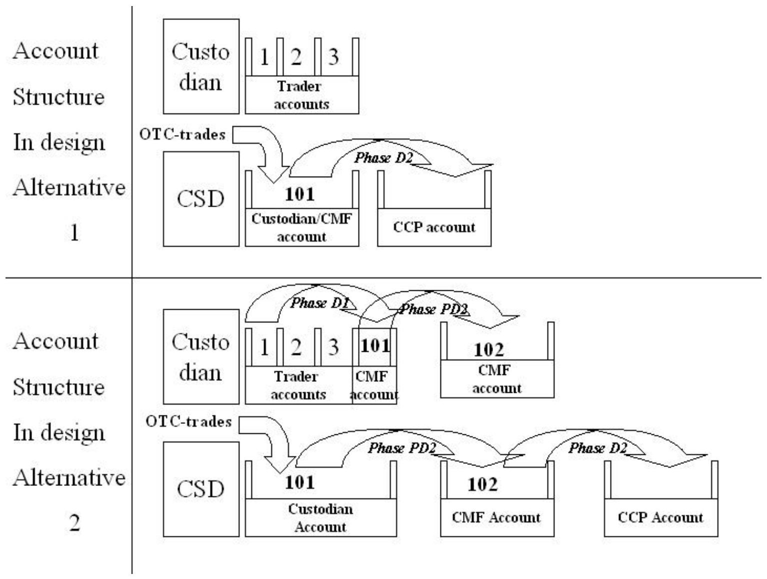


Figure 2: Relationship between account structure and internal processing

[4]If all securities are pooled on the one account that is also used for delivery to the CCP, it is not possible for the CCP to know which part of the pool belongs to which trader. Collecting securities from a short selling trader it might use securities from another trader, without knowing it. This is only legal if a lending agreement is created for these securities. If two accounts are available, securities of traders that don't want to enter into lending might be transferred to the other account, that can not be accessed by the CCP.

both design alternative of two levels: the upper level representing the accounts serviced by the Custodian and the lower level serviced by the CSD. An account in figure 2 is represented as a “container” which might contain an amount of securities and is identified by an integer.

The account(s) serviced by the Custodian that are positioned just above a corresponding account of the CSD are related by an invariance property. For the account structure of the first design alternative this property entails that the total balance of securities that is held at the first three accounts (1,2 and 3) serviced by the Custodian must be equal to the account 101 that is owned by the Custodian and the CMF and is serviced by the CSD. The reason for this is that the mentioned sum of balances on the accounts 1,2 and 3 serviced by the Custodian and the account 101 serviced by the CSD in fact represent **the same** securities. However, because these balances are registered by separate institutions (the Custodian and the CSD) and in separate systems, (temporary) differences might (and in fact will) occur. The internal processing of the Custodian must ensure that corrections are made to eliminate these differences when they occur. In the account structure in the second design alternative there is an invariance relation between account 102 serviced by the CSD and account 102 serviced by the Custodian. Also there is an invariance relation between account 101 serviced by the CSD and the sum of the accounts 1,2,3 and 101 serviced by the Custodian.

The processing in both design alternatives in figure 2 is represented by the arrows. The arrow, marked “phase D2”, corresponds to the same phase D2 mentioned on page 4 in the explanation given with figure 1. The timing of phase D2, represented by these arrows in the respective design alternatives, is determined by the CCP and is therefore the same in both design alternatives. In both the lower and upper part of figure 2 an arrow marked “OTC-Trades” is shown. It is an incoming arrow to represent the settlement, i.e. the receipt, of securities bought on the OTC-Market. These securities are received in both design alternatives on account 101 serviced by the CSD. The upper part of the figure clearly shows that in design alternative 1 these securities are available immediately for delivery to the CMF and CCP, because delivery is accomplished by debiting this same account 101. In design alternative 2, in the lower part of the figure, two additional steps, D1 and PD2, are necessary to transfer these securities to account 102 serviced by the CSD before they are available for delivery to the CMF and CCP in phase D2.[5]

3 Modelling the first design alternative

Figure 3 below shows a dataflow diagram representing the first design alternative of the Clearing and settlement process. Each of the institutions described in section 2 is represented in the figure by a square.

[5]The CCP needs access to the account used by the CMF for delivery. Therefore the CMF must grant the CCP a power of attorney over its account.

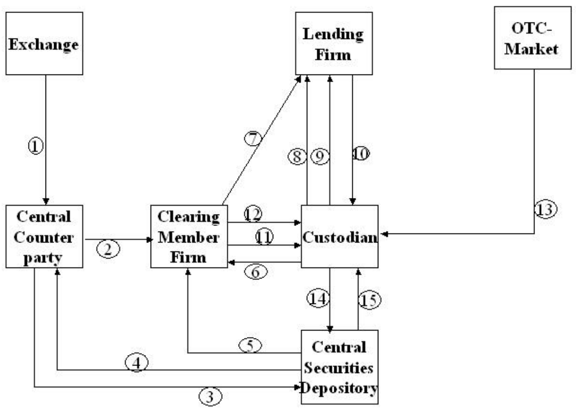


Figure 3: Illustration of the first design alternative of the Clearing and settlement process

The arrows in the figure represent data- and/or control flows between the institutions and are explained in the table below.

Nr	Name	Explanation
1	Trade	Traders close deals on the **Exchange** which are transmitted as *Trade* to the **CCP**.
2	Tradeleg	A *Trade* is split-up into two *Tradeleg*s. Of each a copy is sent by the **CCP** to the **CMF**s that sponsor the traders involved.
3	CCP Instruction	*Tradeleg*s are aggregated by the CCP per day per sponsoring CMF. The **CSD** is instructed to move the securities between the **CCP** and **CMF**.
4	CCP Confirmation	The **CSD** confirms any movement of securities (caused by *3*) to the **CCP**. Unsuccesful movements are recycled in the next *CCP-instruction*.
5	CMF Confirmation	The **CSD** confirms any movement of securities also to the **CMF**.
6	Balance	The **CMF** is informed by the **Custodian** of the *Balance* of securities of each trader to determine if it is sufficient for settlement of that traders *Tradeleg*s.
7	Pending Confirmation	The **CMF** informs the **Lending Firm** of pending delivery of exchange-traded securities so it can cover for an insufficient balance.
8	Balance	The **Custodian** informs the **Lending Firm** of the balance of securities of each trader to determine whether shortages might occur.

Nr	Name	Explanation
9	Pending Instruction	The **Custodian** informs the **Lending Firm** of pending delivery of OTC-traded securities so it can cover for an insufficient balance.
10	LF Instruction	The **Lending Firm** instructs the **Custodian** to move securities from participants with excess balance to participants with a shortage.
11	Client Receive Confirmation	The **CMF** allocates the securities received (given in *5*), to each trader it sponsors (using *2*) and instructs the **Custodian** to credit his balance.
12	Client Deliver Confirmation	The **CMF** allocates the securities delivered (given in *5*), to each trader it sponsors (using *2* and *6*) and instructs the **Custodian** to dedit his balance.
13	Trader Instruction	Traders close deals on the **OTC Market** and send *Trader Instruction* to their **Custodian** instructing it to move the securities as agreed.
14	Custodian Instruction	The **Custodian** forwards *Trader Instructions* to the **CSD** to execute the instructed movement of securities, if the available balance is sufficient.
15	Custodian Confirmation	The **CSD** confirms the **Custodian** that the movement of securities instructed by *(14)* has been executed.

The dataflow diagram described above is translated into the HTCPN presented in figure 4. The HTCPN is constructed by mapping each of the squares in figure 3 onto one substitution transition and each numbered arrow in figure 3 onto one place. This leaves one place named *delay* still unaccounted for. This place is the socket node associated with the port node *delay* in the Petri net fragment in figure 6 and will be explained later. The sub net associated with the substitution transition **CCP** is presented in figure 5. The places *Trades1* in figure 5 and *(1)* in figure 4 are associated as port- and socket node [7]. *Trades1* contains the transactions received from the **Exchange**, and triggers the operations of the CCP. The transition **Trade capture** gathers the transactions in place *Trades2* which is associated with a colourset "TRADES" defined as "list of TRADE". The transition **Generate delay** is enabled if a token is available in its input place *Settlement cycles*. The "settlement cycle" is an institutional parameter designating the time interval in which the **CSD** will execute settlement operations. The creation of a new token in this place triggers the start of a new settlement cycle. In this case study it is assumed there is just one settlement cycle after the middle of each business cycle of 48 time units (representing 24 hours). The time value of the token produced in the place *delay* in figure 5 is 5 time units later, causing the time value of the token produced by the CCP sub net in place *CCP-Instructions1* to be 5 or 6 time units after the middle of each business cycle. This time value represents the start of "phase D2" as explained in section 2.1.

The transition **Notify** selects a sublist (using the function "Select") from

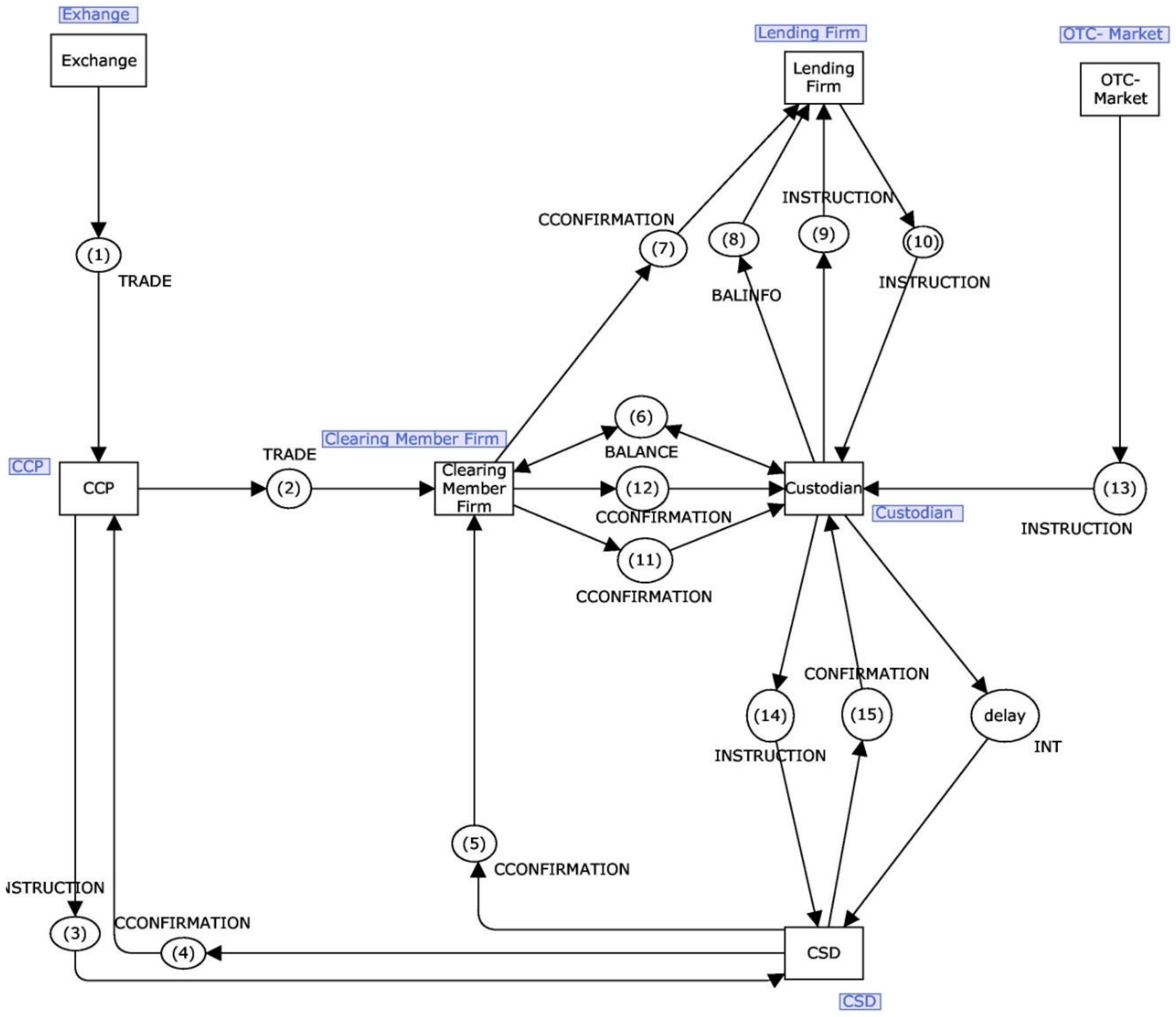


Figure 4: A Petri net model of Clearing and settlement

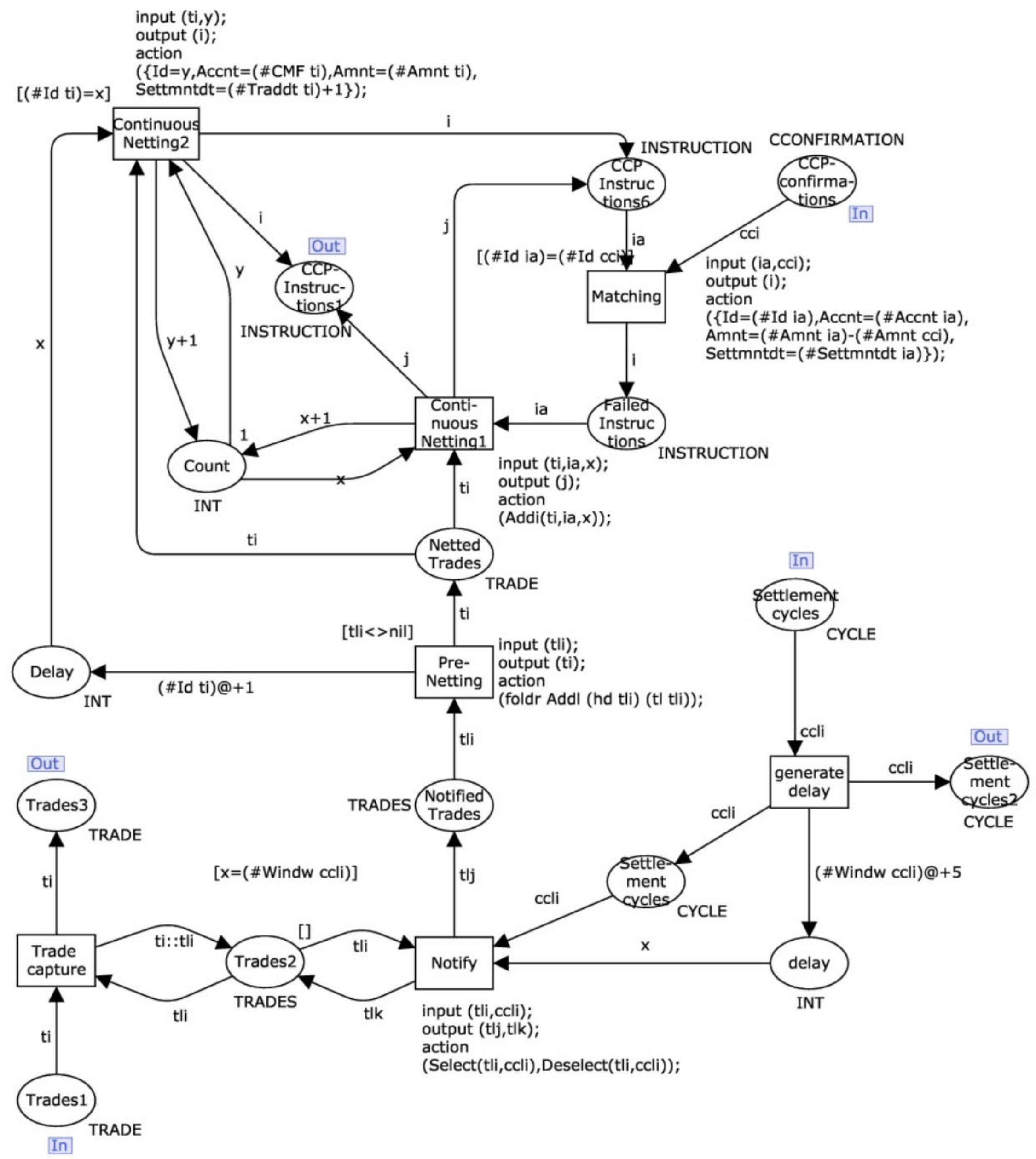


Figure 5: A Petri net model of the Central Counterparty

the list in *Trades2*. The sublist consists of all transactions from the list with a trade date one cycle before the current business cycle [6]. This sublist is put into the place *Notified trades*. The transition **Prenetting** aggregates the amount of traded securities in the sublist and generates one trade in the place *Netted Trades*. Assuming that there is no token in the place *Failed instructions*, the transition **Continuous Netting2** will create a settlement instruction in *CCP Instructions1*, which is a port node associated with socket node *(3)* in figure 4. The **CSD** will respond to the settlement instruction in *CCP Instructions1* by putting a settlement confirmation in place *(4)* in figure 4, which is the socket node that corresponds to the port node *CCP Confirmations* in figure 5. The settlement confirmation contains the amount of securities that has actually been settled. If this is less than the amount in the settlement instruction this will be recycled by the transition **Continuous netting1** in figure 5.[7] Figure 6 shows a fragment of the sub net associated with the substitution transition **Custodian** in figure 4.

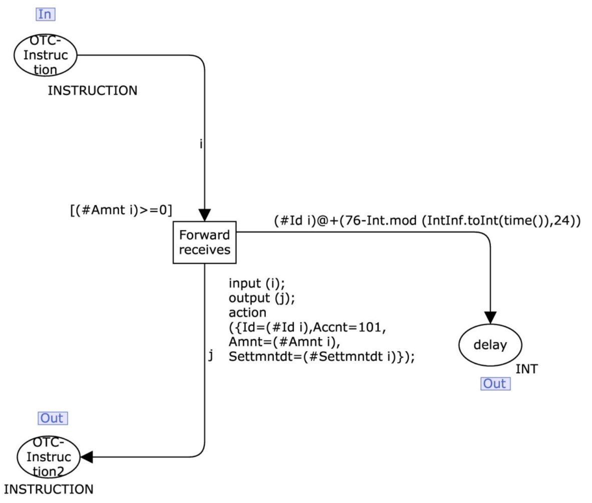


Figure 6: A fragment of the Petri net model of the custodian

It shows how the token in the place *delay* on the primary page in figure 4 is generated. The place *OTC-Instruction* is a port node that corresponds to socket node *(13)* on the page in figure 4 and contains the settlement instructions received from the **OTC-Market**. The transition **Forward Receives** filters instructions corresponding to “buy-transactions” on the OTC-Market (causing securities to be received) and forwards these instructions to the **CSD** with a

[6]Normally there is a time interval of 3 days between trade date and settlement date, for convenience set here to 1 day

[7]Transition **Continuous netting2** must have a lower priority than **Continuous netting1** which is enforced by its input place *Delay*

delay. The delay is calculated in such a way that it will result in a time value of 4 time units after the beginning of the settlement cycle (i.e. the middle of each business cycle of 48 time units) which is just before the start of phase D2 (i.e. 5 or 6 units after the beginning of each settlement cycle). This delay, which will be given the same value in both design alternatives, is chosen in such a way that the differences in performance between the two design alternatives can be demonstrated clearly. This demonstration is given in section 4.2.

4 Analysis of the design alternatives

Important motives for building a model of a process design are firstly to validate it against the intentions of the process owner and secondly to determine its properties before starting implementation. Petri nets have a solid mathematical foundation allowing model execution and, at least in principal, formal verification of properties. Model execution can be used as a means to explain a process design and its performance characteristics to the process owner and thus establish the validity of the model. Formal verification of the properties of a Petri net might not always be feasible. Estimating these properties by model simulation often is a good alternative. Simulation of real world behaviour of a business process will unavoidably involve the introduction of stochastic elements. In the case study described here, for instance, the time of arrival of securities bought on the OTC-Market, represented in figure 4 as a fixed time delay, would be modelled more realistically by a stochastic time delay.[8] Extension of the Petri nets described here with stochastic elements is straightforward, but has not been attempted due to lack of resources. Section 4.2 describes the use of model execution as a tool to demonstrate and explain the relative performance characteristics of the design alternatives of the Clearing and settlement process. For this the term *model evaluation* might be appropriate. [9] But first the performance criteria that are relevant here are introduced in section 4.1.

4.1 Performance criteria

The institutional arrangements shaping the Clearing and settlement process, as explained in section 2.1, have been designed to reduce the risks associated with trading in secondary capital markets [11]. Relevant performance criteria would seem to be how well the Clearing and settlement process succeeds in reducing risk. The term "how well" might include "how efficient". The performance criteria *risk* and *efficiency* are defined next.

Risk The purpose of the Clearing and settlement process as explained in section 2.1 is the timely settlement of traded securities. This timely settlement is

[8]Because, firstly, this depends on the delivery of securities by a counter party which has been left outside the model and, secondly, this is highly unpredictable anyway

[9]Model evaluation can be seen as a logical precursor to *model simulation* which is a means to acquire reliable estimates of these performance characteristics if the models were to be implemented in the real world.

unfortunately not certain. The performance criterion risk is intended to capture this aspect of uncertainty. Risk is defined here as the probability of a "failed settlement". Any transaction for which settlement has not occurred on the contractually laid down intended settlement date constitutes a failed settlement. The consequence of failed settlement of a transaction on the Exchange might be a penalty imposed by the CCP. If the transaction is done on the OTC-Market the consequence can be a claim for damages by the counterparty to the transaction. We give the following definitions:

1. $S_{trad}(t)$ the total amount of securities traded with intended settlement date $\leq t$

2. $S_{fail}(t)$ the total amount of failed settlements on a particular day, i.e. the amount of securities traded with intended settlement date $\leq t$ that have not been settled on date t

The risk R associated with the Clearing and settlement process can now be defined as the total amount of failed settlements in a period between t_0 and t_1 divided by the total amount of securities that should have been settled in that period:

$$R = \int_{t_0}^{t_1} [S_{fail}(t)/S_{trad}(t)]dt$$

Because the term $S_{trad}(t)$ will be the same in both design alternatives during model execution, we only need to measure $S_{fail}(t)$ to compare their performance.

Efficiency Efficiency is here defined as the costs associated with the Clearing and settlement of a transaction excluding the costs associated with settlement failure as explained in the previous section. Two sources for these transaction costs are assumed: firstly the *settlement costs*, i.e. fee that has to be paid to the CSD for the settlement of each transaction, and secondly, the *borrowing costs*, i.e. the fee that has to be paid to the Lending Firm for borrowing securities.

To determine relative performance of the two design alternatives in terms of settlement costs we need to measure the number of Settlement Instructions that are actually settled by the CSD. To determine relative performance of the two design alternatives in terms of borrowing costs we need to measure the amount of securities that is borrowed. So, if the same amount of risk is attained by a design alternative with less settlement instructions and less borrowing, then that design alternative is more efficient.

4.2 Model evaluation

In this section (a subset of) the measurements gathered from executing one run of each of the Petri nets representing the two design alternatives are presented.

Each design alternative is executed for 384 units of model time. We will call this one "run" of the Petri net model. One unit of model time represents half an hour in the real world, so the running time of each model execution represents 8 business cycles of 24 hours in the real world. The Petri nets are described in detail in [10].

4.2.1 Performance of design alternative 1

In the table at the end of this section measurements are presented that are gathered from one run of the Petri net representing the first design alternative. Each sequence of three columns gives the measurements taken from the Petri net during one cycle of 48 time units. This cycle represent one business day on the capital market. The table shows measurements from the last three, of eight, consecutive cycles in the run. During each cycle three traders, numbered 1, 2 and 3, are active on the capital market. The measurements in one column corresponds to the behaviour of one trader. The rows numbered 1 through 4 represent the buying and selling of securities on the exchange and the OTC market respectively. During each cycle the Petri net fragments of the Exchange and the OTC-Market will generate the set of values given in these four rows and the three columns corresponding to that cycle. This set of values is influenced by the initial markings of these Petri net fragments. For any cycle t this set corresponds to the term $S_{trad}(t)$ discussed in section 4.1.

Row 5 *starting position*, represents the amount of securities in possession of each trader at the start of the cycle, and is also determined by the initial marking of the Petri net. Row 13 *end position*, represents the amount of securities in possession of each trader at the end of the cycle. A change in position during a particular cycle is caused by the buying and selling of securities in the *previous cycle*. The reason for this is simply that there is a time delay of one cycle between a transaction on the exchange and the settlement of the transaction.[10] If securities are bought in cycle 1, then these securities are received in cycle 2. If securities are sold in cycle 1, then these securities are delivered in cycle 2, i.e. *provided* that the counterparty to the transaction has the securities in his possession. If not then this will cause a *failed delivery*. If they are not received for this same reason then this will also cause a *failed receive*. The failed deliveries and receives are shown in the three rows at the bottom of the table. The values in these three rows for any cycle t correspond to the term $S_{fail}(t)$ discussed in sections 4.1. These values can thus be used to calculate the performance criterion risk associated with this design alternative.

Failure can be prevented if the required securities can be borrowed. This is shown in row 8 *Lending received*, representing the amount of securities that are actually borrowed (i.e. received from the Lending Firm). Row 9 *Lending deliv-*

[10]This delay is in fact often 3 days. A delay of 1 cycle suffices however to show how this type of delay affects the process. For a more real life scenario this can be easily changed in to a delay of 3 cycles.

ered, represents the amount of securities that is lent out (i.e. delivered to the Lending Firm). These two figures must add up to zero because borrowing and lending does not change the amount of securities in the system. The delivery of securities to the CCP is represented by row 10 *Exch. deliv.*. It might occur that securities are bought on the OTC market and sold on the exchange in the same cycle. If the securities that are bought on the OTC market are received after the moment that delivery of the securities sold on the exchange to the CCP should occur, then this delivery will fail. If they are received before that moment then the delivery will succeed. This issue plays a critical role in the difference between the design alternatives discussed in section 2.2. Row 6 is reserved for securities that have been bought on the OTC-Market and have been received in time to be used for delivery to the CMF and CCP. Row 11 is for those that have been received too late.

As explained in section 2.2 the point in time that securities bought on the OTC market are received is crucial in the relative performance of the two designs alternatives. The measurements are taken under the assumption that the securities bought on the OTC market are received just one time unit before delivery of securities to the CCP occurs, referred to in sections 2 and 2.2 phase D2, but after phase D1. The timing of phase D2 is fixed by the CCP and is the same in both design alternatives. The measurements show that, under these assumptions, in the first design alternative the delivery of these securities will succeed, they show up in row 6. In the next section it is shown that the delivery of these securities in the second design alternative will not succeed, they will then show up in row 11.

Cycle	T=6			T=7			T=8		
Trader	1	2	3	1	2	3	1	2	3
1 Exch. bought	-	-	15	-	-	15	-	-	15
2 Exch. sold	27	72	-	27	72	-	27	72	-
3 OTC bought	-	30	-	-	30	-	-	30	-
4 OTC sold	9	-	22	18	-	11	9	-	22
5 **Start position**	64	0	0	0	0	10	0	0	0
6 OTC received	-	-	-	-	30	-	-	30	-
7 Exch. received	-	-	15	-	-	15	-	-	15
8 Lending received	-	42	-	18	7	-	5	10	-
9 Lending deliv.	37	-	5	-	-	25	-	-	15
10 Exch. deliv.	27	72	-	18	37	-	5	40	-
11 OTC received	-	-	-	-	-	-	-	-	-
12 OTC delivered	-	-	-	-	-	-	-	-	-
13 **End position**	0	0	10	0	0	0	0	0	0
14 OTC failed deliv.	18	-	11	9	-	22	18	-	11
15 Exch. failed deliv.	-	-	-	9	35	-	31	67	-
16 Exch. failed receives	-	-	-	-	-	-	-	-	-

4.2.2 Performance of design alternative 2

The table below shows measurements taken from one run of the Petri net representing the second design alternative. It shows clearly that the performance is worse. First of all the number of failed settlements, shown in the 3 rows at the bottom of each table, is higher. This means that the level of *risk* in the second design alternative is higher. Secondly more borrowing is necessary in the second design alternative compared to the first. This can be seen from the measurements shown in rows 8 and 9. This means that the *borrowing costs* in the second design alternative are higher than in the first and thus the efficiency of the Clearing and settlement process in second design alternative is lower than in the first. The level of *settlement cost* in the second design alternative is also higher. This can not be seen directly from this table, because it doesn't contain a count of the number of settlement instructions. It can be inferred however by looking at the values in row ten, "Exch. deliv.". The sum of the entries in this row corresponds to the total number of securities that have been delivered by the CMF to the CCP. In the table with measurements of the second design alternative this sum is lower than in the table with measurements of the first design alternative. In each cycle there is only one instruction sent from the CMF to the CCP. So this means that per instruction less securities are delivered in the second design alternative, and the cost per security delivered is thus higher. This is of course a logical consequence of the fact that there are more settlement failures in the second design alternative.

Cycle	T=6			T=7			T=8		
Trader	1	2	3	1	2	3	1	2	3
Exchange bought	-	-	15	-	-	15	-	-	15
Exchange sold	27	72	-	27	72	-	27	72	-
OTC bought	-	30	-	-	30	-	-	30	-
OTC sold	9	-	22	18	-	11	9	-	22
Start position	9	30	25	0	30	7	0	30	0
OTC received	-	-	-	-	-	-	-	-	-
Exchange received	-	-	0	-	-	0	-	-	0
Lending received	-	18	-	-	7	-	-	-	-
Lending deliv.	-	-	18	-	-	7	-	-	-
Exchange deliv.	9	48	-	0	37	-	0	30	-
OTC received	-	30	-	-	30	-	-	30	-
OTC deliv.	-	-	-	-	-	-	-	-	-
End position	0	30	7	0	30	0	0	30	0
OTC failed deliv.	18	-	11	9	-	22	18	-	11
Exchange failed deliv.	18	24	-	45	59	-	72	101	-
Exchange failed receives	-	-	15	-	-	30	-	-	45

5 Concluding remarks

In this final section the research questions posed in section 1 will be discussed in the light of the experience gained by the case study. Also some issues will be raised concerning the use of CPN-Tools.

Are HTCPNs suitable for modelling the Clearing and settlement process?
The case study illustrates that HTCPNs have sufficient expressiveness to model the relevant characteristics of the Clearing and settlement process. These characteristics can be categorized as follows:

1. Decentralized control
2. High complexity
3. Time critical

Decentralized control The Clearing and settlement process involves a large number of different participants, like the traders, Clearing Member Firms, Custodians, Lending Firms, the CCP, and the CSD, that have been described in this paper. Each of these participants in general only controls a part of the process and independently decides about the course of action to be taken. The consequence however is that the performance of the Clearing and settlement process as a whole, measured by the criteria described in section 4.1, depends very much on efficient cooperation between the participants. Petri nets are well suited to model both independence and cooperation.

High complexity It requires several hundreds of places and transitions to model the Clearing and settlement process [10]. The primary page located at the top of the hierarchy of the HTCPNs, shown in section 3, represent the Clearing and settlement process with only 7 transitions and several tenths of places. Without a hierarchy concept it would be difficult to develop these models in a controlled way. It allowed to translate high level concepts like "CSD", Custodian, "CMF" etc. familiar to business experts directly into substitution transitions. The use of coloured tokens also reduced the complexity of the Petri net models. For instance, concepts such as trades, settlement instructions or accounts are translated into structured coloursets which would be hard to model in a PT-net.

Time critical The performance criteria described in section 4.1 imply that timing is an important determinant of the performance of the clearing and settlement process. The HTCPN allowed to model the timing aspect of the Clearing and settlement process satisfactorily by attaching time-stamps to tokens and delay statements to the arcs or transitions.

The following issues concerning the use of CPN Tools can be raised. *Firstly* priorities between transitions is modelled in figure 5 by using time stamps. Perhaps other ways of doing this might be more appropriate. *Secondly*, moving around of highly structured data, typical for this kind of applications, is sometimes a bit inconvenient. To name a small example: if a highly structured output token is created from an input token which is the same except for one small item in the token, all items must be copied explicitly resulting is relatively long lines of code. Furthermore, applications like the one explored in this case study might benefit from more tool support for the modelling of data.

Are HTCPNs helpful in validating process models?
Because the transitions and most of the places on the primary pages of the Petri nets can be mapped to familiar business concepts, the graphical representation of these primary pages could be easily explained to business experts. The validation of the dynamics of the Petri net was performed by showing an animation of the Petri net. This appeared to be a bit more tricky. The following issues can be raised concerning the animation of behaviour with CPN-Tools:
Firstly, animation by CPN-Tools shows a snap shot of the entire net and state space after each step (or sequence of steps). No distinction is made between the part of the state space that has changed and the part that hasn't change. If the net is large enough it is difficult to see for an observer what has happened between two snapshots. *Secondly* it is difficult for a human observer to get a picture of the entire occurrence sequence. To do that he has to remember all the snapshots he has seen. *Thirdly*, dynamics of transitions in a HTCPN is defined by the Petri nets on the underlying subpages. But validation of the entire Petri net model by a process owner or business expert is preferably done by animating the net on the primary pages, because that is the level he understands the best. *Fourthly* the squares and circles used by CPN-Tools to represent activities and communication are slightly alien to business experts. More domain specific symbols might do better. *Finally* the tables with performance measurements gathered during the animation described in section 4.2 also proved helpful in explaining the behaviour of the net. This type of reporting is often quite familiar to process owners and business experts. Creating these reports was somewhat time consuming. Creating support in CPN-Tools to address the issues just mentioned might ease the process of model validation.

Can HTCPNs effectively be used to decide between design alternatives?
Performance analysis of the design alternatives for the Clearing and settlement process by executing the Petri net models representing these design alternatives showed better performance for the first design alternative relative to the second, as was expected. This might seem a bit trivial, because this result just confirms the informal explanation of the difference between the two design alternatives given in section 2.2. In reality this difference became only clear after the models were built and compared by means of model execution. The informal explanation is historically of later date than the results of model execution. In fact the development of the Petri net models was an iterative process, going back and forth between model building and model execution on the one hand and informal conception of what was going on on the other. This shows that formal model building and model execution are useful tools to create insight and understanding of a process design and its performance. Some additional tool support

for creating model variants might however be convenient. Large fragments of both design alternatives (i.c. what is designated as “market infrastructure”, i.e. CSD, CCP, Exchange and OTC-market) are the same. In the case study presented here, the Petri net model of the market infrastructure representing the first design alternative was copied into the Petri net model representing the second design alternative. This has obvious drawbacks. Having just one copy would be better.

References

[1] W. van der Aalst; *Timed coloured Petri nets and their application to logistics*, Phd thesis Tue, 1992

[2] Clearnet; *Introduction of Clearing 21 for the clearing of cash products (phase 2) in the Amsterdam branch of Clearnet*, Aug. 2002

[3] Clearnet; *Main specifications for the migration from Cash Dutch applications towards Clearing21 Version 1.2*, 16 May. 2002

[4] LCH Clearnet; *Settlement Connect - Detailed specifications of Clearing 21 Version 5*, Feb. 2004

[5] Euroclear Nederland; *Euroclear Nederland Guidelines*, Amsterdam, 27 october 2005

[6] The Giovannini Group; *Cross-Border Clearing and Settlement Arrangements in the European Union*, Brussels, 2001

[7] K. Jensen; *Coloured Petri Nets; Basic Concepts, Analysis Methods and Practical Use; Vol. 1*, Springer-Verlag, 1992

[8] K. Jensen; *Coloured Petri Nets; Basic Concepts, Analysis Methods and Practical Use; Vol. 2*, Springer-Verlag, 1995

[9] M. Kohn; *Financial Institutions and Markets*, Oxford University Press, 2004

[10] P. M. Kwantes; *Design of Clearing and settlement operations: a case study in Business Process Modelling and Analysis with Petri nets*, Masters Thesis, Leiden Unniversity, 2006

[11] T. H. Mclnish; *Capital Markets, A Global Perspective*, Blackwell Publishers Inc., 2000

Some Rules to Transform Sequence Diagrams into Coloured Petri Nets [⋆]

Óscar R. Ribeiro and João M. Fernandes

{oscar.rafael, jmf}@di.uminho.pt
Dept. of Informatics, University of Minho, Portugal

Abstract. This paper presents a set of rules that allows software engineers to transform the behavior described by a UML 2.0 Sequence Diagram (SD) into a Colored Petri Net (CPN). SDs in UML 2.0 are much richer than in UML 1.x, namely by allowing several traces to be combined in a unique diagram, using high-level operators over interactions. The main purpose of the transformation is to allow the development team to construct animations based on the CPN that can be shown to the users or the clients in order to reproduce the expected scenarios and thus validate them. Thus, non-technical stakeholders are able to discuss and validate the captured requirements. The usage of animation is an important topic in this context, since it permits the user to discuss the system behavior using the problem domain language. A small control application from industry is used to show the applicability of the suggested rules.

1 Introduction

Although complex systems are, by their nature, hard to build, the problem can be ameliorated if the user requirements are rigorously and completely captured. This task is usually very difficult to complete, since clients and developers do not use the same vocabulary to discuss. For behavior-intensive applications, this implies that the dynamic behavior is the most critical aspect to take into account. This contrasts with database systems, for example, where the relation among data types is the most important concern to consider. A scenario is a specific sequence of actions that illustrates behaviors, starting from a well defined system configuration and in response to external stimulus. Petri nets are used to formalize the behavior of some component, system or application, namely those that have a complex behavior. Since Petri nets are a formal model, they do not carry any ambiguity and are thus able to be validated.

This paper proposes a set of rules that allow software engineers to transform requirements, expressed as a UML 2.0 SD into an behaviorally equivalent CPN. The main purpose of the transformation is to generate a CPN [1] that is akin to be animated (with the mechanism available in CPN-Tools) and thus understood by the users. The synthesized CPN that can be shown (i.e. animated) to

[⋆] This work has been supported by the grant with reference SFRH/BD/19718/2004 from *"Fundação para a Ciência e Tecnologia"*

the users or the clients in order to reproduce the expected scenarios and thus validate them. Thus, non-technical stakeholders are able to discuss and validate the captured requirements. The usage of animation is an important topic in this context, since it permits the user to discuss the system behavior using the problem domain language, which they are supposedly familiar with.

The paper is organized as follows. In section 2, the SDs of UML 2.0 are introduced. Section 3 presents some rules of how to translate a SD into a behaviorally equivalent CPN. In section 4, the result of applying the rules presented in previous section to the case study of an industrial reactor system. In Section 5 is presented a discussion of the related work. Section 6 presents the conclusions of this work and some possible directions for the future work.

2 UML Diagrams for interaction

The introduction of UML 2.0 standard changed almost every sort of things in previous versions designated by UML 1.0.

The dynamic part of the system can be specified in UML 2.0 through various behavioral diagrams, such as: activity diagrams, sequence diagrams and state machines diagrams. These diagrams use behavioral constructs, namely activities, interactions, and state machines.

Interactions are a mechanism for describing systems, which can be understood and produced, at varying level of detail. Usually, interactions do not tell the complete story, when they are produced by designers or by computer systems, because normally some other legal and possible traces are not contained within the described interactions. There are some exceptions where the project request that all possible traces of a system shall be documented through interactions.

An interaction is formed by lifelines and messages between them, that sequence is important to understand the situation. Although data may be also important, its manipulation is not the focus of interactions. Data is carried by the messages, and stored in the lifelines, and can be used to decorate the diagrams.

SDs are the most common interaction diagram defined by the UML [2], that focus on the message interchange between a number of lifelines. Communication diagrams show interactions through an architectural view where the arcs between the communicating lifelines are decorated with description of the passed messages and their sequencing. Interaction overview diagrams are a variant of activity diagrams that define interactions in a way that promotes overview of the control flow, these diagrams can be seen as a high-level structuring mechanism that is used to compose scenarios through sequence, iteration, concurrency or choice. There are also optional diagram notations such as timing diagrams and interaction tables. In this work, we concentrate on SDs.

In the UML 2.0 new notions for SDs are introduced to treat iterative, conditional and various other control of behavior. The old iteration markers and guards on messages have been dropped from SDs.

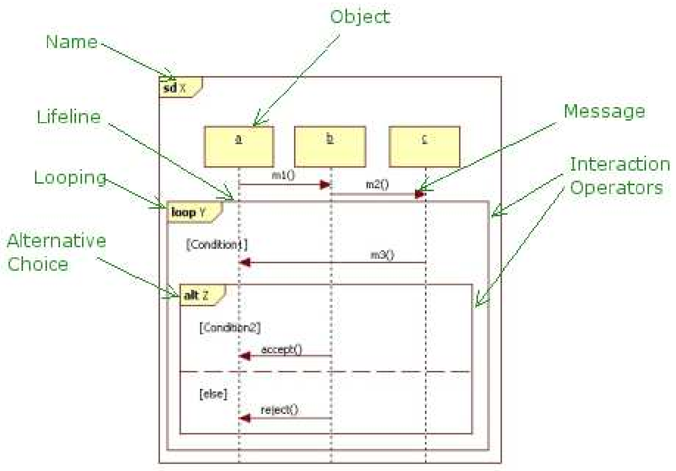


Fig. 1: An example of a UML 2.0 SD

Fig. 1 shows an example of a SD. A SD is enclosed in a frame and includes a pentagon in the upper left handed corner with the keyword `sd` followed by a label identifying the SD.

There are several possible operators, whose meaning is described informally in the UML 2.0 Superstructure specifications [2]:

- **sd:** Indicates the principal frame of the sequence diagram;
- **ref:** references another fragment of interaction;
- **seq:** indicates the weak sequencing of the operands in the fragment, which is select by default. The weak sequencing maintain the order inside each operand, and the events on different operands and different lifelines may occur in any order.
- **strict:** specifies that messages in the fragment are fully ordered;
- **alt:** specifies that the fragment represents a choice between two possible behaviors. There is a guard associated with the fragment, its evaluation define which of choices is executed;
- **par:** indicates that the fragment represents a parallel merge between the behaviors of the operands;
- **loop:** indicates an interaction fragment that shall be repeated some number of times. This may be indicated using a guard condition, and it is executed until the guard evaluates to false.

UML 2.0 provides two kinds of conditions in SDs, namely interaction *constraints* and *state invariants*. An interaction *constraint* is a boolean expression shown in square brackets covering the lifeline where the first event will occur, positioned above that event inside an interaction operand. A *state invariant* is a constraint on the state of an instance, and is assumed to be evaluated during run time immediately prior to the execution of the next event occurrence. Notationally, state invariants are shown as a constraint inside a state symbol or in curly brackets, and are placed on a lifeline.

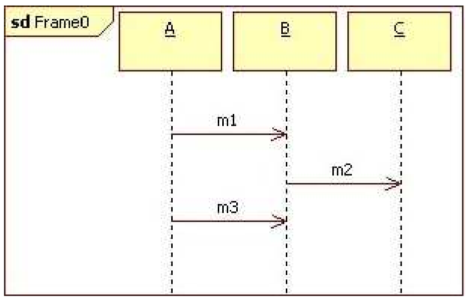
In the previous versions of UML it was not possible to express that, at any time, a specific scenario should not occur. In the UML 2.0 negative behavior (i. e. invalid traces) can be specified using the new operator **neg**. Currently, this operator is not considered in this work.

3 Transforming SD Operators into CPNs

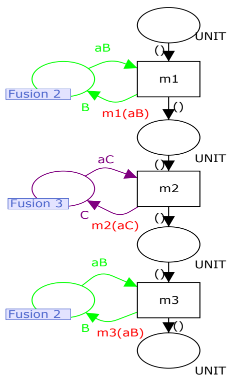
In this section we show how to translate some of the high-level operators available in the UML 2.0 SDs, into a behaviorally equivalent CPN. To accomplish this, we explain the semantics of the operator, we describe in an informal way how the transformation is achieved, and additionally we show the result of applying these ideas to some illustrative examples. We restrict our study to the following high-level operators: `strict`, `seq`, `par`, `loop` and `alt`. Operators like `neg`, `assert`, `critical` are not considered by now.

First of all we look to *InteractionFragments* without any of the high-level operators. An *InteractionFragment* is a set of *Lifelines*, each of which has a sequence of *EventOccurences* associated with it.

We consider a semantic for SD with a order relation between messages such that the emission requires the reception of the preceding message.



(a) A UML 2.0 SD without high-level operators



(b) The obtained CPN

Fig. 2: Example of transform a SD without high-level operators

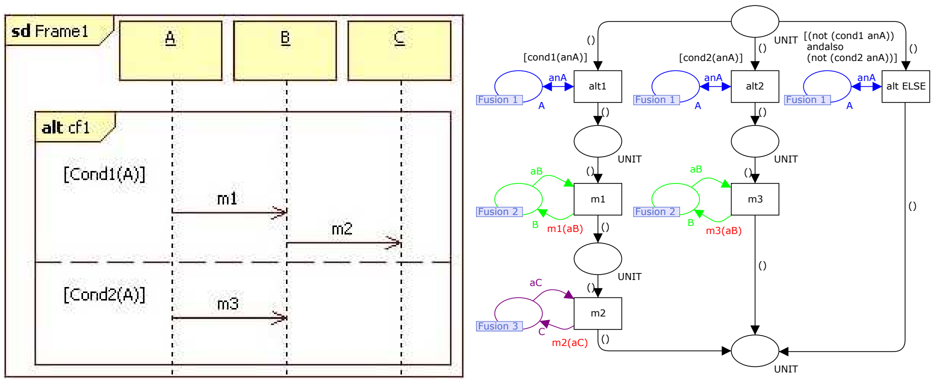
The SD presented in Fig. 2a represents an interaction without high-level operators. There are three *Lifelines* and three messages between them. The obtained CPN (see Fig. 2b) associates a transition for each message in the SD. In this way an execution of a message is represented by the firing of its corresponding transition in the CPN. There are places to guarantee the order between the firring of transitions, and other places to represent the object which the message changes when executing. When firing a transition, a function is applied to the object in the place. This function is a representation of the changes made in the object of the lifeline in the message's destination. To represent the guards in the SD we use a transition guarded by a conditions over the object representation. When there are more than one message in the same point of a lifeline we consider a unique transition which includes all the messages.

Let us consider *CombinedFragments* with high-level operators.

3.1 Alternative Choice

The choice of behavior is represented by a *CombinedFragment* with the *interactionOperator* `alt`. Each operand of `alt` has an associated guard, which is evaluated when choosing the operand to be executed. No more than one operand will be chosen and in this work we assume that the guards must be disjoint. When one of the operands has its guard evaluated to true, the interaction associated with this operand is considered. The empty guard is by default evaluated to true. The operand guarded by `else` means that the guard is evaluated to true when none of the guards in the other operands is evaluated to true. In the case that none of the operands' guards are evaluated to true (this means that there are no `else` and empty guards) none of the operands are executed.

The SD in Fig. 3a is transformed into the CPN in Fig. 3b. Each operand in SD is transformed in a sequential branch. All sequential branchs begin in a common input place and end a common output place.



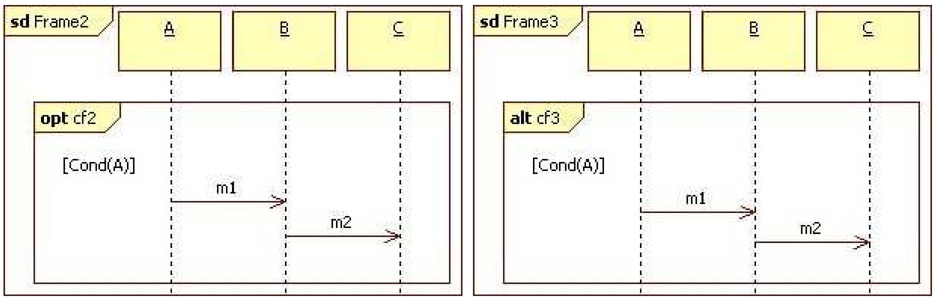
(a) A UML 2.0 SD (b) The obtained CPN

Fig. 3: Example with the alternative choice operator (`alt`)

Please notice that in this case there is no `else` guard, and thus when none of the guards is evaluated as true, no operand is executed. In terms of CPNs this is represented by the rightmost part of the CPN where the "`alt ELSE`" transition condition is the negated disjunction of all other guards. The other branches are guarded by the same condition as in SD and describe the same sequence.

3.2 Optional

The optional operator, represented by *InteractionOperator* `opt`, can be seen as an alternative choice with only one *Operand*, whose guard is not the `else` (see Fig. 4). With this similarity, we can apply to the optional operator the same general translation scheme used for alternative choice.



(a) A UML 2.0 SD (b) The corresponding UML 2.0 SD with `alt`

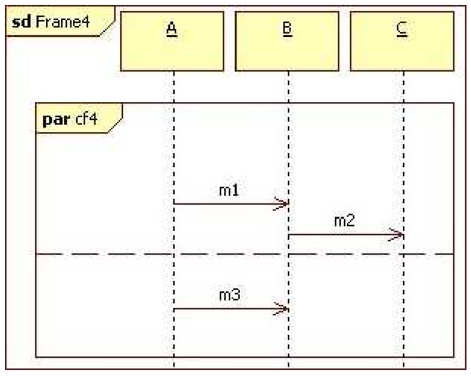
Fig. 4: The option operator (`opt`) expressed by an alternative choice

3.3 Parallel Composition

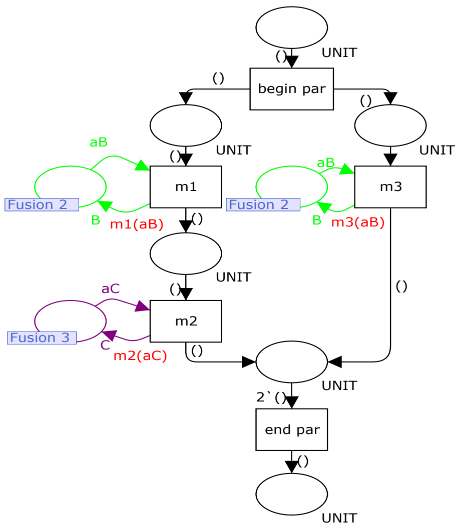
The parallel merge between two or more behaviours is represented by a *CombinedFragment* with the *interactionOperator* `par`. Keeping the order imposed within operands, *EventOccurrences* from different operands can be interleaved in any way. The SD in Fig. 5a is transformed into the CPN in Fig. 5b. The obtained CPN has two additional transitions to control the interleaving of behaviors. The transition "`begin par`" creates two branches (one for each operand) introducing a token into the two output places, in this way we obtain the interleaving between the transitions of each branch. The transition "`end par`" wait for the execution of all created branches, because it is enabled only when its input place has a number of token equal to the number of created branches.

3.4 Weak Sequencing

When using the *InteractionOperator* `seq` the corresponding *CombinedFragment* represents a weak sequencing between the behaviors of the operands. The or-



(a) A UML 2.0 SD



(b) The obtained CPN

Fig. 5: Example with the parallel composition operator (*par*)

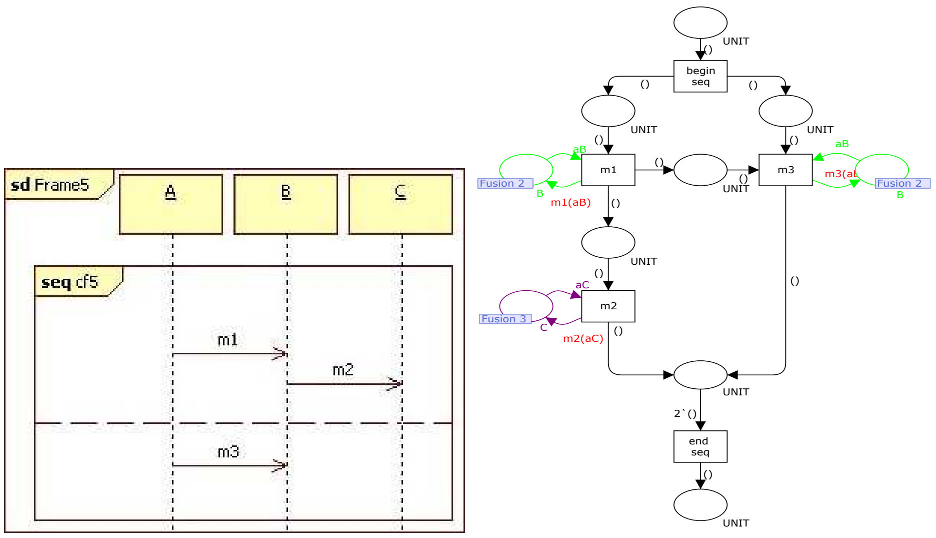
dering of *EventOccurrences* within each of the operands are maintained in the result. *OccurrenceSpecifications* on different lifelines from different operands may come in any order. *OccurrenceSpecifications* on the same lifeline from different operands are ordered such that an *EventOccurrence* of the first operand comes before that in the second operand.

In Fig. 6a we have an example of a SD with `seq` operator. The messages `m1` and `m3` have the *EventOccurence* in the same *Lifeline*, and in the first operand, after the message `m1` we have the message `m2`. Thus, message `m1` must occur before messages `m3` and `m2`.

To construct a corresponding CPN to a SD with the `seq` operator, we first consider the CPN for the parallel composition between the operands, and after that we impose some more order between transitions in different branches. The CPN in Fig. 6b is obtained from the CPN in Fig 5b changing the name of transitions "`begin par`" and "`end par`") to "begin seq" and "end seq", adding the place between transitions "`m1`" and "`m3`" and the corresponding arcs to complete the connection.

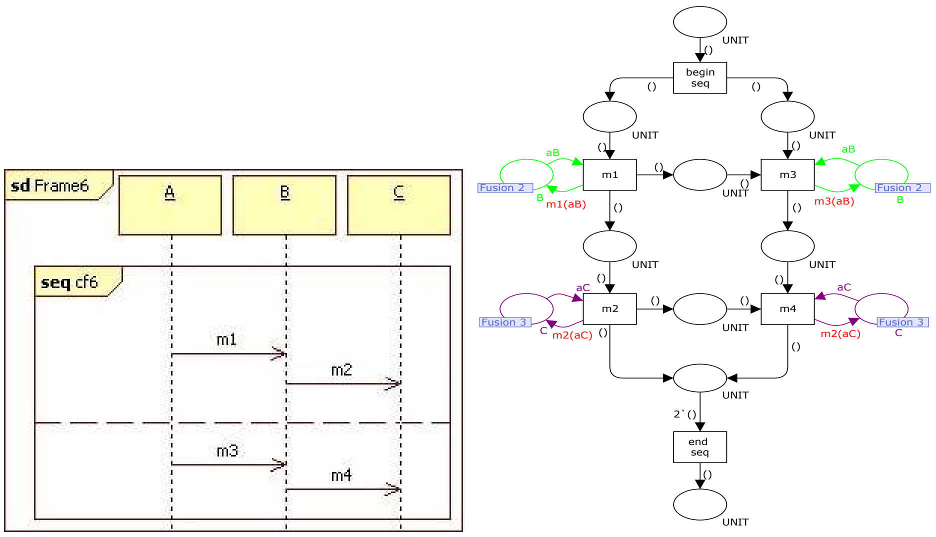
The SD in Fig. 7a is another example using the operator `seq`. The corresponding CPN is presented in Fig. 7b.

There are some particular cases using this operator. If the *EventOccurrence* of the last message from the first operand is in the same *Lifeline* as the first message of the second operand, we have a sequential order between all the messages in the operands. If none *EventOccurrence* of messages is in the same lifeline we have a parallel composition between the operands.



(a) A UML 2.0 SD (b) The obtained CPN

Fig. 6: Example with the weak sequencing operator (*seq*)



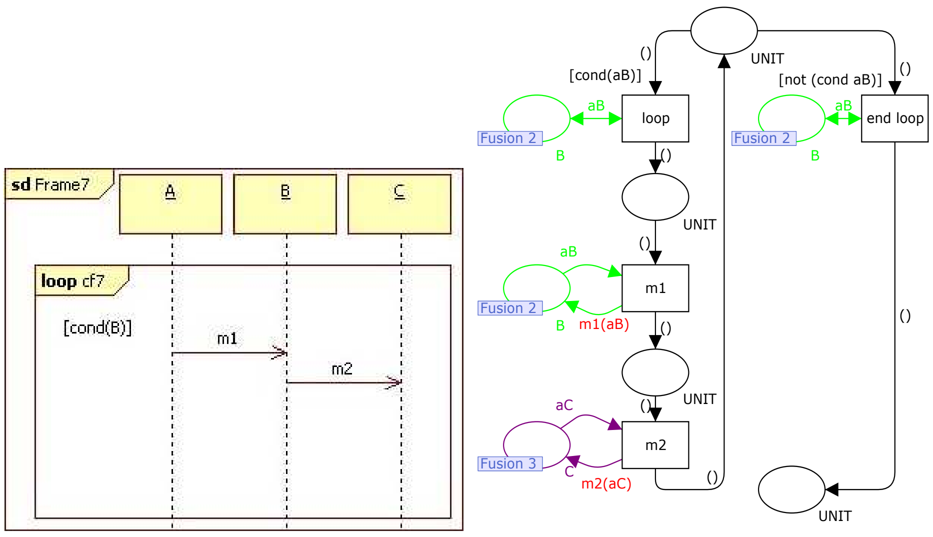
(a) A UML 2.0 SD (b) The obtained CPN

Fig. 7: Another example with the weak sequencing operator (*seq*)

3.5 Looping

The `loop` *InteractionOperator* represents the iterative application of the operand in the *CombinedFragment*. This iterative application can be controlled by a guard or by a minimum and maximum number of iterations.

Given the CPN for the operand inside the `loop`, we add two transitions: “`loop`” and “`end loop`”. These two transitions have the same input place. Transition “`loop`” is enabled if the condition (guard for `loop` operator) evaluates to true, and its output place is the input place for the operand’s CPN. The transition “`end loop`” is enabled when the condition evaluates to false and its output place is used as connection to the end of the `loop` operator. In Fig. 8 we have an example with the `loop` operator.



(a) A UML 2.0 SD (b) The obtained CPN

Fig. 8: Example with the looping operator (*loop*)

4 Validation of the Rules

To validate the proposed transformation rules we plan to apply them to several case studies, so that we can also evaluate their practical usefulness. Currently we are using an industrial reactor as a case study. In this chapter we show two CPNs for the reactor: one obtained directly from the requirements (or more precisely adapted from a PN-based specification) and another one obtained from a SD using the proposed translation rules.

4.1 A Case Study: Industrial Reactor

The industrial reactor system consists in a reactor that controls the filling of a tank. It was used in previous works [3–5].

A plant of the reactor system is presented in Fig. 9. The system has two storage vessels, called `SV1` and `SV2`, each of them has a valve (`openSV1` and `openSV2`) to control the exit of liquid. Downside of each storage vessel there is a measuring vessel (`MV1` and `MV2`). A measuring vessel has the same structure as the storage vessel plus two sensors, one indicating when it is full and another when it is empty.

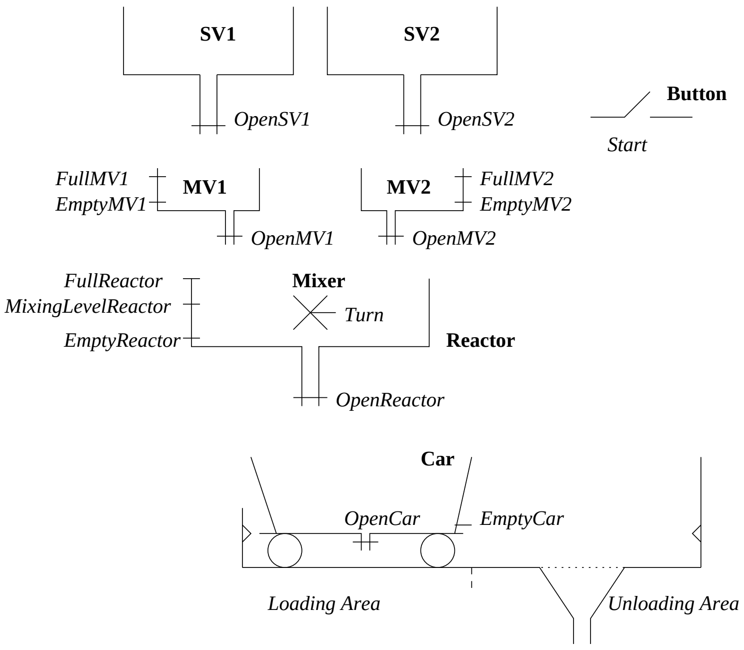


Fig. 9: The environment of industrial reactor system

The `Reactor` is fed with two kinds of liquids from measuring vessels `MV1` and `MV2` which draw from storage vessels `SV1` and `SV2`. After the reaction between the liquids is complete, the reactor is discharged into catch vessel named `Car`. When the `Reactor` is empty the process product is transported using carriage `Car`. To ensure complete reaction the process liquid in the reactor is agitated by stirrer `Mixer`.

When the push button `Start` is pressed the valves `OpenSV1` and `OpenSV2` are opened and measuring vessels `MV1` and `MV2` are refilled until a high-level condition `FullMV1` (`FullMV2`) is sensed. After that, `OpenSV1` (`OpenSV2`) is closed.

The reactor is filled with a liquid input control valves `OpenMV1`, `OpenMV2` and a product discharge valve `OpenReactor`. At the start of a reaction cycle charges of process, liquids are delivered into the reactor from the measuring vessels `MV1` and `MV2`. The valves `OpenMV1`, `OpenMV2` are opened while this proceeding the reactor stirrer may start (`Turn`), when the level in the reactor is higher than `MixingLevelReactor`. When a low level (`EmptyMV1` in `MV1`, `EmptyMV2` in `MV2`) is sensed the valves `OpenMV1` an `OpenMV2` must be closed and the reactor is emptied (`OpenReactor`). After discharging the reactor (`EmptyReactor`) product is transported by using carriage which may move right (`GoUnloadingArea`) or left (`GoLoadingArea`).

4.2 A Manual CPN model

A model of the industrial reactor using High-Level Petri Nets was presented in [4], where a shobi-PN (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net) model of an industrial reactor control system is considered as a case study to illustrate the model's applicability and capabilities. The shobi-PN model is an extension to SIPNs (Synchronous and Interpreted Petri Nets) [5]. The model of shobi-PN includes the same characteristics as the SIPN model, in what concerns to synchronism and interpretation, and adds to functionalities by supporting object-oriented modeling approaches and new hierarchical mechanism, in both the control unit and the plant. We have done a translation of SIPN models into PROMELA code to improve the analysis methods for SIPNs [6]. We used the SIPN model of industrial reactor as a case study to validate our approach.

Based on the shobi-PN model of reactor system presented in [4] we created the CPN model in Fig. 10.

The objects are represented by record colors, and the methods are represented by functions on the the object's color, e.g. the storage vessel object and the method to open a storage vessel is defined by the following CPN-ml code:

```
colset StorageVessel =
          record  id       : INT *
                  isOpen   : BOOL*
                  capacity : INT ;

fun openStorageVessel (sv:StorageVessel)
               = StorageVessel.set_isOpen sv true;
```

In this model we have some more pipelining between tasks of the system, than in the shobi-PN model. For example, it is possible to be emptying the storage vessels while the the car is going to the unloading area.

Notice that the tokens associated to each instance of an object is used to control the behavior of the CPN model. The place `anti-place` is only used to restrict the firing of transition `t1`. To simulate the system behavior we create a CPN to represent the environment of reactor.

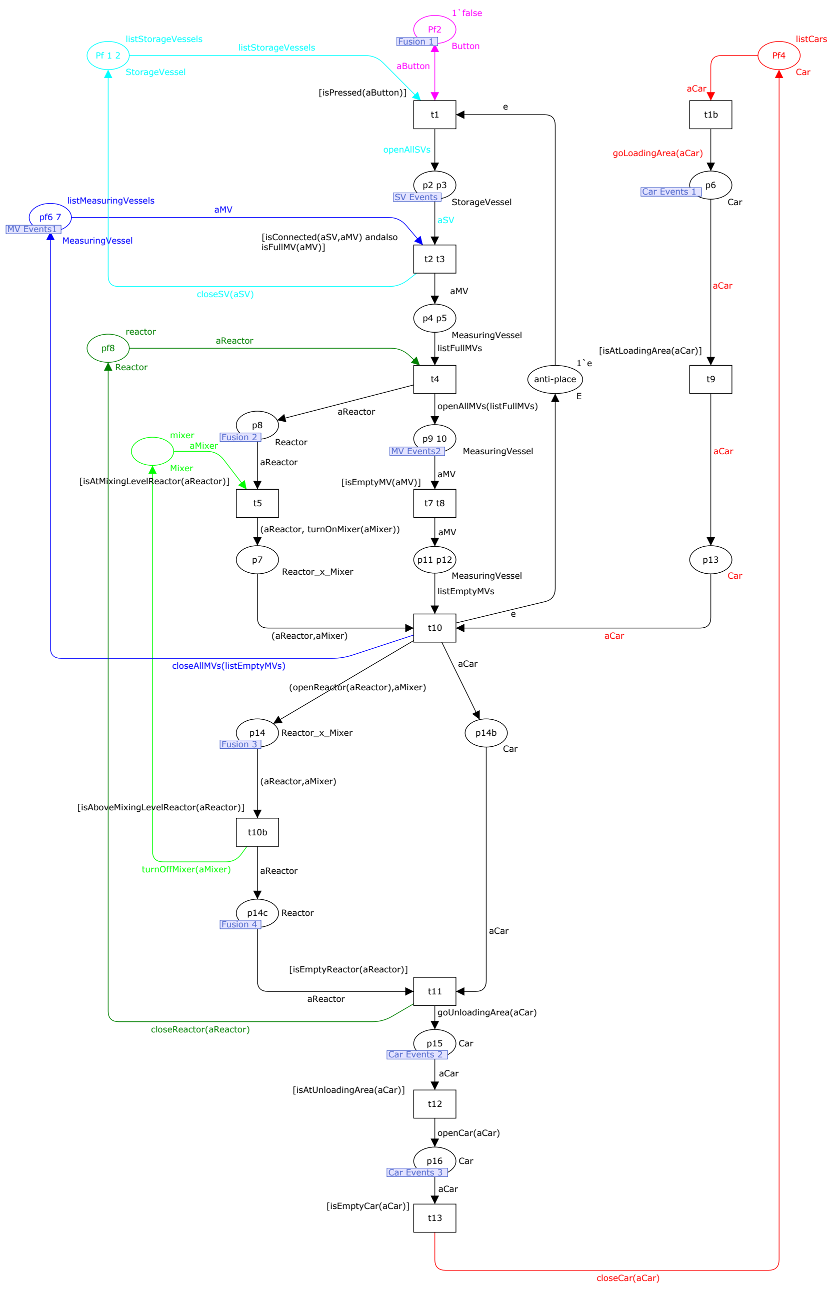


Fig. 10: A CPN model for reactor system

4.3 A CPN model from a SD

This subsection presents a SD for some scenarios of reactor system's usage, and a CPN model obtained from the SD through the rules presented in section 3.

The reactor system is textually described in subsection 4.1, and in subsection 4.2 there is a CPN model which defines its behavior. Taking into account these two exercises of analyzing the reactor system we construct the SD in Fig. 11 to represent the handled scenarios. This SD uses high-level operators, namely the `ref` to point to another two SDs: "Preparing Car" (see Fig. 12) and "Vessels Behavior" (see Fig. 13).

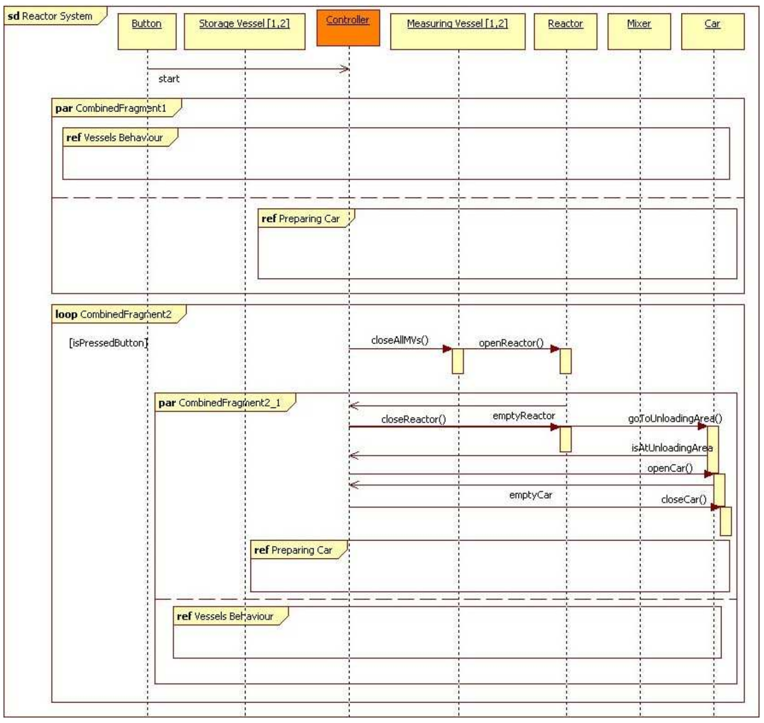


Fig. 11: A SD describing some scenarios of using the reactor system

To transform this SD we firstly apply the rules to the fragments with one high-level operator. After that we compose the obtained CPNs into a hierarchical CPN. We put each SD pointed by `ref` into a subpage. Fig. 14 shows the CPN obtained from the SD in Fig. 12, where we can find transitions which are links to a CPN in a subpage. The subpage `Vessels Behavior` is presented in Fig. 14, which corresponds to the SD presented in Fig. 12.

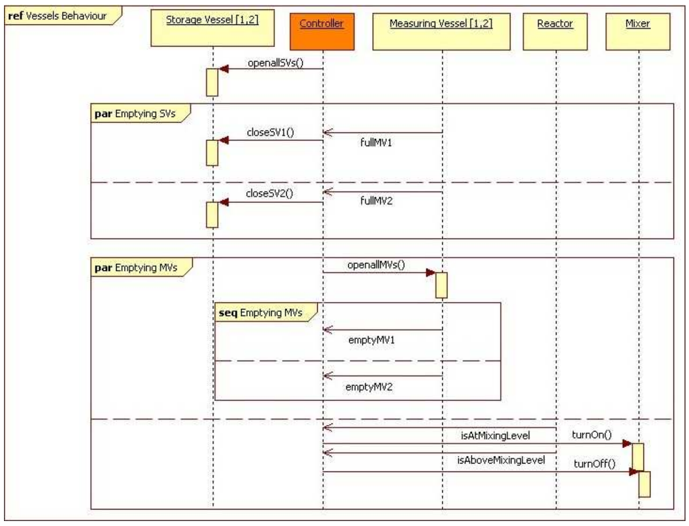


Fig. 12: A SD describing the behavior of vessels

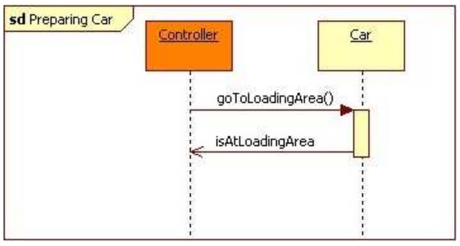


Fig. 13: A SD describing the preparation of car

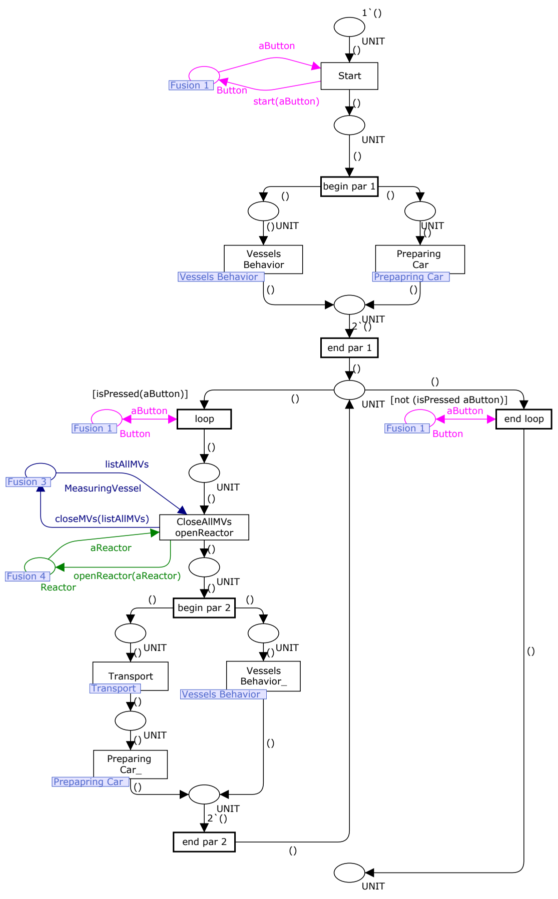


Fig. 14: CPN from the SD presented in Fig. 11

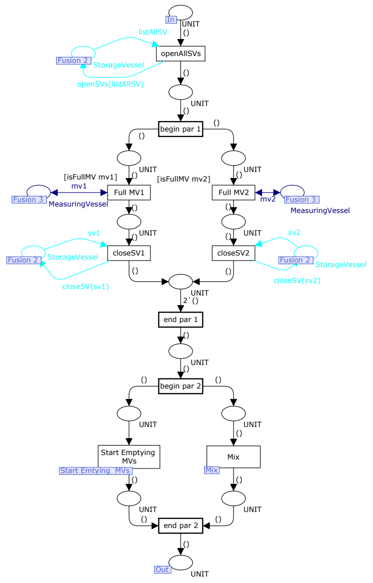


Fig. 15: CPN to represent the behavior of vessels (see Fig. 12)

4.4 Animation

We have developed a *SceneBeans* [7] animation to be associated with the created CPNs, through the *BRITNeY* animation tool [8]. A screen shot of the animation of reactor system is shown in Fig. 16.

In this animation we can find the elements of the problem domain, which is the reactor domain. We obtain this animation using the Scenebeans animator. On the left top side of the image we have commands accepted by the animation. On the left button side, we have the events produced by animation. These set of commands and events are use to do the interaction between the animation and the CPN models. For example, to animate the message "`openAllSVs`" in SD of Fig. 12, we invoke, in the corresponding transition, two commands of the animation: "`openSV1`" and "`openSV2`". The user can interact with the animation using the start button. The vessels on the top are the storage vessels. Each storage vessel has a corresponding measuring vessel. In the center we have the reactor vessel, with a mixer inside of it. On the button we have which transports the liquid to the unloading area.

This animation is intended to help us in validating the obtained CPNs in terms of their appropriateness to express in the user's domain language the requirements of a given system. Additionally we plan to use the animation to compare the two CPNs for the reactor system (Figs. 10 and 14) and to evaluate the performance of the rules to generate "good" CPNs. We would like to

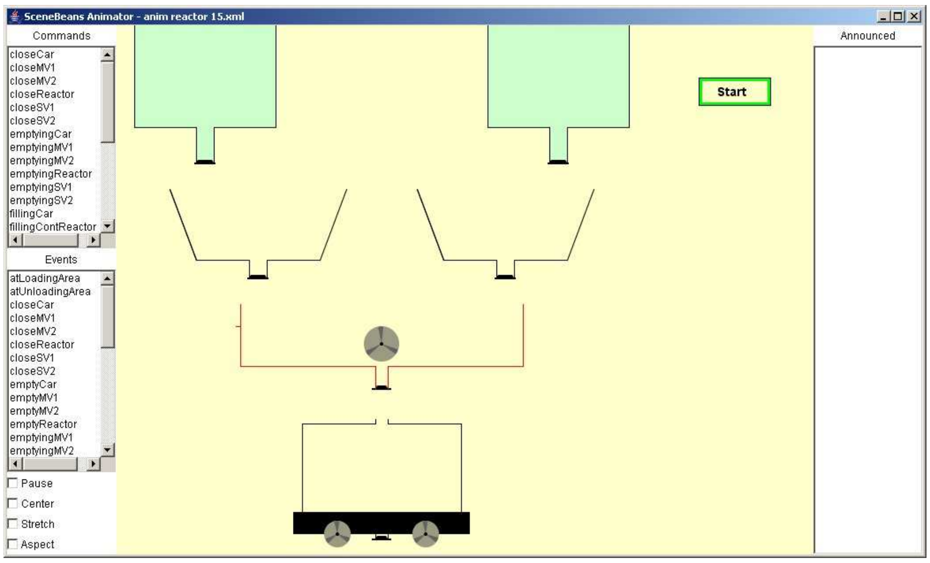


Fig. 16: The animation of the reactor system

observe if the animations controlled by both CPNs produce the same externally observable behavior, independently of their internal structures.

5 Related work

Transforming scenarios into state-based models (namely, sequential finite-state machines) has been the subject of many researchers. In fact, several approaches were already proposed to combine the usage of both scenarios and state-based models and, in this section, some of them are discussed. A major problem for obtaining state-based models from scenarios is the big computational complexity of the synthesis algorithms that does not allow the technique to scale up. Some additional obstacles include methodological issues, the definition of the level of detail in the scenarios to allow effective synthesis, to the problem of guaranteeing that the scenarios are representative of the users' intentions.

The majority of the approaches propose the usage of FSM (finite state machine). Krüger et al. suggest the usage of Message Sequence Charts (MSCs) for scenario based specifications of component behavior, especially during the requirements capture phase of the software process [9]. They discuss how to schematically derive statecharts from MSCs, in order to have a seamless development process. Harel proposes the usage of scenario based programming, through UML (Unified Modeling Language) [2] use cases and play in scenarios [10]. Harel's play in scenarios make it possible to go from a high level user friendly requirements capture method, via a rich language for describing message sequencing, to a full model of the system, and from there to the final implementation.

Whittle and Schumann propose an algorithm to automatically generate UML statecharts from a set of UML sequence diagrams [11]. The usage of this algorithm for a real application is also presented [12, 13], and the main conclusion is that it is possible to generate code mostly in an automatic way from scenario based specifications.

Hinchey et al. propose a round trip engineering approach, called R2D2C (Requirements to Design to Code), where designers write requirements as scenarios in constrained (domain specific) natural language [14]. Other notations are however also possible, including UML use cases. Based on the requirements, an equivalent formal model, using CSP, is derived, which is then used as a basis for code generation.

Uchitel and Kramer present an MSC language with semantics in terms of labeled transition systems and parallel composition [15]. The language integrates other languages based on the usage of high level MSCs and on the identification of component states. With their language, scenario specifications can be broken up into manageable parts using high level MCSs. These authors also present an algorithm that translates scenarios into a specification in the form of Finite Sequential Processes, which can be used for model checking and animation purposes.

The synthesis of Petri nets from scenario-based specification is less popular than the one that generates FSMs, because Petri nets represent a model of computation, where parallelism and concurrency of activities are "natural" characteristics. We next describe some of the approaches proposed to obtain Petri nets from a set of scenarios. In [16], the authors present a polynomial algorithm to decide if a scenario, specified as a Labelled Partial Order, is executable in a given place/transition Petri net. The algorithm preserves the given amount of concurrency and does not add causality. In case the scenario is indeed executable in the Petri net, the algorithm computes a process net that respects the concurrency expressed by the scenario. Although quite useful, this technique is not yet available for high-level Petri nets, such as Object-oriented Petri nets, Colored Petri nets (CPNs), or Reference nets.

In [17] an informal methodology to map Live Sequence Charts (LSCs) into CPNs is presented, for allowing properties of the system to be verified and analyzed.

The formal translation of Interaction Overview Diagrams (IODs) into PEPA nets is described in [18]. PEPA nets constitute a performance modeling language that consists of a restriction of Petri nets, where tokens are terms of a stochastic process algebra [19]. The translation is based on the idea that the structure given by the IOD corresponds to the high level net structure of the PEPA net, and the behavior described in the IOD nodes (sequence diagrams) can be translated onto PEPA terms. The translation allows a designer to formally analyze UML 2.0 models, using the tools for PEPA nets.

A formal semantics by means of Petri nets is presented in [20] for the majority of the concepts of sequence diagrams. This semantics allows the concurrent behavior of the diagrams to be modeled and subsequently analyzed. Moreover, the usage of CPNs permits an efficient structure for data types and control elements. In their approach they use places to represent the messages, instead transitions as we do.

This work is based on the preliminary results presented in [21], where the authors show how the behavior of animation prototypes results from the translation of SDs into CPNs. We extend their results by showing how to translate more types of operators in UML 2.0 SDs, namely by considering parallel constructors which result in CPNs with true concurrency (i.e. CPNs that are not just sequential machines).

6 Conclusions

In this paper we show a set of rules to transform SD into equivalent CPNs for animation proposes. In UML 2.0, SDs are quite expressive and this work explores the new constructors (in relation to UML 1.x) that allow several plain sequences to be combined in a unique SD. Thus the rules allow the generation of a CPN that covers several sequences of behaviors. This work is in progress so we plan to develop it further. First we plan to investigate all SD operators, namely the *neg* operator and evaluate if it can be useful for the software engineer. Second, we need to better tune the rules, to realize if they can be automated. In the future we would like to use a UML-based tool to draw the SD diagrams and apply automatically the rules to obtain a CPN for animation. Probably this automation requires a second set of rules that “optimizes” the CPN by eliminating redundant parts. In this work we only have a validation of transformations though the implementation, we plan to study the soundness and completeness of the approach.

Finally the usage of the rules in real-world projects is planned, since we believe that methods and tools for software engineers need to be evaluated by them in complex industrial projects.

References

1. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Brauer, W. and Gozenberg, G. and Salomaa edn. Volume Volume 1, Basic Concepts of Monographs in Theoretical Computer Science. Springer-Verlag (1997) ISBN: 3-540-60943-1.
2. Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modelling Language. Addisson-Wesley (2003)
3. Adamski, M.: Direct Implementation of Petri Net Specification. In: 7th International Conference on Control Systems and Computer Science. (1987) 74–85
4. Machado, R.J., Fernandes, J.M., Proença, A.J.: Specification of Industrial Digital Controllers with Object-Oriented Petri Nets. In: IEEE International Symposium on Industrial Electronics (ISIE’97). Volume 1. (1997) 78–83

5. Fernandes, J.M., Pina, A.M., Proença, A.J.: Concurrent Execution of Petri Nets based on Agents. In: Encontro Nacional do Colégio de Engenharia Electrotécnica (ENCEE95), Lisbon Portugal, Ordem dos Engenheiros (1995) 83–9
6. Ribeiro, O.R., Fernandes, J.M., Pinto, L.F.: Model Checking Embedded Systems with PROMELA. In: 12th IEEE International Conference on the Engineering of Computer Based Systems (ECBS 2005), Greenbelt, MD, E.U.A., IEEE Computer Society Press (2005) 378–85
7. Pryce, N., Magee, J.: SceneBeans: A Component-Based Animation Framework for Java. (Online) http://www-dse.doc.ic.ac.uk/Software/SceneBeans/.
8. Westergaard, M., Lassen, K.B.: Building and Deploying Visualizations of Coloured Petri Net Models Using BRITNeY Animation and CPN Tools. In: Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. (2005)
9. Krüger, I., Grosu, R., Scholz, P., Broy, M.: From MSCs to Statecharts. In Rammig, F.J., ed.: Distributed and Parallel Embedded Systems, Kluwer Academic Publishers (1999) 61–71
10. Harel, D.: From play-in scenarios to code: An achievable dream. IEEE Computer **34**(1) (2001) 53–60 (Also, Proc. Fundamental Approaches to Software Engineering (FASE; invited paper), Lecture Notes in Computer Science, Vol. (Tom Maibaum, ed.), Springer-Verlag, March 2000, pp. 22-34.).
11. Whittle, J., Schumann, J.: Generating statechart designs from scenarios. In: 22nd International Conf. on Software (ICSE), Limerick, Ireland (2000) 314–323
12. Whittle, J., Saboo, J., Kwan, R.: From scenarios to code: An air traffic control case study. In: ICSE'03. (2003) 490–497
13. Whittle, J., Kwan, R., Saboo, J.: From scenarios to code: An air traffic control case study. Software and Systems Modeling **4**(1) (2005) 71 – 93
14. Hinchey, M.G., Rash, J.L., Rouff, C.A.: A formal approach to requirements-based programming. ecbs **00** (2005) 339–345
15. Uchitel, S., Kramer, J.: A workbench for synthesising behaviour models from scenarios. In: Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada, IEEE Computer Society (2001) 188–197
16. Juhs, G., Lorenz, R., Desel, J.: Can I Execute My Scenario in Your Net? In Ciardo, G., Darondeau, P., eds.: Applications and Theory of Petri Nets 2005: 26th International Conference (ICATPN 2005). Volume 3536 of LNCS., Miami, USA, Springer (2005) 289
17. Amorim, L., Maciel, P., Nogueira, M., Barreto, R., Tavares, E.: A methodology for mapping live sequence chart to coloured petri net. In: IEEE International Conference on Systems, Man and Cybernetics. Volume 4. (2005) 2999–3004
18. Kloul, J., Kuster-Filipe, J.: From interaction overview diagrams to pepa nets. In: Proceedings of the 4th Workshop on Process Algebras and Timed Activities (PASTA'05), Edinburgh (2005)
19. Gilmore, S., Hillston, J., Kloul, L., Ribaudo, M.: Pepa nets: a structured performance modelling formalism. Performance Evaluation **54**(2) (2003) 79–104
20. Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., Stehno, C.: Compositional semantics for UML 2.0 sequence diagrams using Petri Nets. In Lecture Notes in Computer Science, Volume, J.., ed.: SDL 2005: Model Driven Systems Design: 12th International SDL Forum. Volume 3530., Grimstad, Norway (2005) 133–148
21. Machado, R.J., Lassen, K.B., Oliveira, S., Couto, M., Pinto, P.: Execution of UML Models with CPN Tools for Workflow Requirements Validation. In: Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. (2005)

Inside LoLA - Experiences from building a state space tool for place transition nets

Karsten Wolf
Universität Rostock
18051 Rostock, Germany

1 History of LoLA

The development of LoLA started in the second half of the 1990s when research results on symmetry reduction and partial order reduction needed to be validated with the help of a prototype implementation. At that time, the author worked with Peter Starke who developed the Petri net analysis tool INA. It turned out that, due to unfortunate design decisions in the core part of INA, algorithms run in unacceptable time. Consequently, a new core part of a verification tool was designed, based on the analysis of time consumption in INA. This core, step by step complemented with state space based verification techniques, was later on called LoLA and first presented at the Petri net conference in 2000.

Meanwhile, LoLA is integrated in several tool platforms including CPN-AMI of Université Paris VI, the Model Checking Kit of Universität Stuttgart, and the Petri Net Kernel of Humboldt-Universität zu Berlin. It was successfully used in several case studies, including the detection of hazards of an asynchronous circuit that is part of a globally asynchronous - locally synchronous chip design for coding and decoding signal of a 802.11 WLAN protocol, the validation of a Petri net semantics for the Business Process Execution Language for Web Services (BPEL), and for the verification of block-level distributed programs. LoLA was able to solve a challenge example posted some years ago by Hubert Garavel. The example stemmed from a LOTOS specification. The verification problem was quasi-liveness.

2 Features of LoLA

LoLA can read place transition nets and high level nets in formats that are easy to read, easy to write, and easy to generate. It can not yet read PNML but translate its own format into PNML code. Once started, LoLA calculates a state space for the purpose of verifying a property on-the-fly. That is, LoLA stops execution as soon as the property to be verified is settled. The class of properties to be verified can be selected at compile-time, through editing a particular file `userconfig.H`. In the same file, the reduction techniques to be used can be specified. Consequently, one can generate different executable files, each specialized on one particular property and one particular combination of reduction techniques.

The classes of properties that can be verified using LoLA include

- Reachability of a marking or a state predicate,
- Liveness of a transition or a predicate,
- Reversibility and existence of home states of the net,
- Quasi-liveness (non-death) of a transition,
- Boundedness of a place or the net,
- Existence of deadlocks,
- Model checking for CTL,
- The linear time properties $F\phi$, $GF\phi$, and $FG\phi$.

The particular formula, predicate, place, or transition to be checked can be read from a separate file. For state space reduction, LoLA offers

- Partial order reduction, with optimized versions for the various properties,
- The symmetry method, including an automatic exploration of the symmetries,
- The coverability graph technique,
- The sweep-line method, including an automatic generation of a progress measure,
- Cycle coverage reduction based on transition invariants, i.e. the attempt to store only one state per cycle of the state graph,
- State compression based on place invariants, i.e. the removal of places that are linearly dependent on other places,
- A memory-less goal-attracted simulation.

Most of the techniques can be applied in combination. LoLA takes care during its compilation process that combined techniques fit to each other. In case of incompatibility, one of the conflicting techniques is switched off.

Triggered by command line options, LoLA generates several pieces of output that can be saved to separate files. Output includes

- a witness or counterexample path, if possible
- a witness or counterexample state, if possible

- the generated state space itself
- the generating set of computed symmetries
- the generated progress measure for the sweep-line method.

3 Implementation

In comparison with other tools, LoLA typically shows excellent run time behavior and competitive reduction results. The reason for running fast is due to a careful design of the core procedures of LoLA including state space management and the firing process.

A state space of LoLA is stored as a binary decision tree. Thus, search and insert of states can be implemented in linear time. We invented a dense structure for storing a decision tree which requires a small record and a compressed bit vector for each state. The state compression based on place invariants saves memory and run time, since it decreases the depth of the decision tree to typically 40-70% of the original size. On top of the decision trees, there is a hash table that further speeds up access to states.

In the process of firing a transition in LoLA, most information is processed incrementally. Calculation of the next state, the new list of enabled transitions, and information needed for the reduction techniques, is realized by just locally updating the corresponding old state, or lists. This way, generation of the next state can be executed in a time that only depends on the fan-in and fan-out of the fired transition, independently from the size of the net. This way, the idea of *locality* is exploited for efficiency of the state space generation.

LoLA explores a state space by depth-first or breadth-first search. Breadth-first search is simulated by a depth-first search with stepwise enlarged depth limit. It is useful for generating shortest paths, otherwise depth-first search tends to be more efficient. The depth-first stack of LoLA consists of just transitions. Backtracking from a marking is organized by firing transitions backwards. This way, backtracking is as well an incremental procedure that is much more efficient than copying states.

Several reduction techniques can be implemented as a mutation of elementary actions during depth-first search. Partial order reduction influences the generation of the list of transitions to be fired, symmetry reduction affects the search and insert procedures with freshly generated states. The sweep-line method is an exception - it requires its own management of states since states must be sorted according to their progress value. In the implementation of the sweep-line method, LoLA exploits the nature of the automatically generated progress measure. In this measure, progress values of a transition and its immediate successor transition differ only by a value that is globally bounded by a constant. Thus, the sweep-line execution engine organizes states as bundles (one bundle for each progress value), and keeps always those bundles efficiently available that are within reach from the currently processed state. Other bundles are organized in slower data structures such as lists.

4 Lessons learned

Our implementation was driven by the assumption that a net is small, compared to a huge state space. We optimized those steps that are executed once for each state. To this end, we accepted memory overheads due to explicitly storing preprocessed information for each net element. This approach is one of the key factors of the speed of LoLA.

Another key factor for run time is the fact that LoLA uses reduction techniques that are tailored to specific properties. LoLA shows its best results if the verification problem can be reduced to properties such as reachability, deadlocks, or dead transitions. In those cases, LoLA can substantially outperform tools like SPIN which feature general purpose (LTL preserving) reduction techniques.

In case studies, it was always useful to separate the verification problem into many small properties tat can be verified individually. For instance, the quasi-liveness of the net in the Garavel challenge mentioned earlier could be reduced to quasi-liveness queries for each individual transitions. This way, we obtained a few hundred state spaces that actually fit into memory, compared to a global state space that would not fit.

Using Coloured Petri Nets to Simulate DoS-resistant protocols

Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto

Information Security Institute
Queensland University of Technology
GPO Box 2434, Brisbane, QLD 4001, Australia
s.tritilanunt@student.qut.edu.au,
{c.boyd, e.foo, j.gonzaleznieto}@qut.edu.au

Abstract. In this work, we examine unbalanced computation between an initiator and a responder that leads to resource exhaustion attacks in key exchange protocols. We construct models for two cryptographic protocols; one is the well-known Internet protocol named Secure Socket Layer (SSL) protocol, and the other one is the Host Identity Protocol (HIP) which has built-in DoS-resistant mechanisms. To examine such protocols, we develop a formal framework based on Timed Coloured Petri Nets (Timed CPNs) and use a simulation approach provided in CPN Tools to achieve a formal analysis. By adopting the key idea of Meadows' cost-based framework and refining the definition of operational costs during the protocol execution, our simulation provides an accurate cost estimate of protocol execution comparing among principals, as well as the percentage of successful connections from legitimate users, under four different strategies of DoS attack.

1 Introduction

Denial-of-service (DoS) attacks continue to be one of the most troublesome security threats to communication networks. DoS attacks can be classified roughly into two types: *flooding* attacks and *logical* attacks. During a flooding attack the adversary simply keeps sending messages to the victim so that the victim is unable to process any genuine requests for service. Logical attacks try to be more clever and aim to exhaust either the computational or memory resources of the victim by exploiting some feature of the communications protocol.

A general method to defend against logical DoS attacks is to authenticate connections before committing significant resources to servicing the connection. In practice, though, secure authentication is a computationally expensive process and so the effort expended in authenticating has the potential to be turned into a DoS attack in itself. Recognising this dilemma, protocol designers in the 1990s advocated a simplified form of authentication to be used before full-fledged cryptographic authentication takes place. A canonical example of this is the use of *cookies* first suggested by Karn and Simpson [18]. This mechanism can be recognised as the principle of *gradual authentication* [25] in which the server uses multiple authentication mechanisms, each successive one being more computationally expensive. Such methods have been incorporated into several protocols for authentication and key exchange, most notably into the widely deployed IPSec protocols [1].

Design of key exchange protocols has long been considered a delicate problem, but the analysis becomes even harder when DoS prevention is an additional requirement. Meadows [19] introduced a systematic framework to analyse DoS resistance by computing and comparing the cost incurred by both parties at each step in a (key exchange) protocol. Meadows analysed the STS protocol (a protocol without special DoS resistance properties) and later Smith et al. [24] used Meadows' framework to analyse the JFK protocol [1] in order to demonstrate its DoS prevention capabilities.

Surprisingly, there has been little interest in the research community in applying Meadows' framework to different protocols. Moreover, the limited application so far has suffered from two significant shortcomings which make the results of restricted value.

1. The cost analysis has only taken into account *honest* runs of the protocol. In principle, the adversary (typically the client in a client-server protocol) can deviate arbitrarily from the protocol in order to achieve an attack. By only taking into account honest behaviour it is quite likely the logical attacks will be missed. While Meadows certainly recognised this fact, no research has yet examined the effectiveness of the framework in detecting such potential attacks.

2. Meadows used only a coarse measure of computational cost, with three levels denoted as cheap, medium or expensive. In practice it can be quite difficult to classify and compare operations in such a limited way. For example, in Meadows' classification digital signature generation and verification are counted as of equal cost, yet in practice an RSA signature generation may take 2 or 3 order of magnitude more effort than RSA signature verification.

Motivated by the above two limitations, this paper provides a refinement of Meadows' cost-based framework. For our sample protocols we use the Host Identity Protocol (HIP) [21], which has built-in DoS resistance, and compare it with the well-known Secure Socket Layer (SSL) protocol. To develop a formal framework of such protocols, we use CPN Tools [27] which is a general-purpose verification tool for modeling and analysing Coloured Petri Nets. Using CPNs as our formalism, we provide a formal specification of two protocols to allow automatic searching of adversary and victim cost under different adversarial attack strategies. Moreover, we set up another experiment for examining the tolerance of HIP under such attacks.

Comparing to the previous work on the analysis of HIP by Beal and Shepard [6] that employs a mathematical approach, simulation approaches are also valued in the research community. They have been applied not only for exploring vulnerabilities in cryptographic protocols, but also guaranteeing security services of such protocols. Using simulation approaches has several benefits over mathematical analysis; for instance, simulation provides *flexibility* to the developer to adjust parameters for evaluating the system. Simulation also provides *visualization* to users who can see and learn what is happening during the simulation of cryptographic protocols to gain more understanding for evaluating the correctness of those protocols.

The main contributions of this paper are:

– a refinement of Meadows' cost-based framework to more accurately represent the cost of typical cryptographic algorithms;
– the first formal specification and automatic analysis of Meadows' framework;
– a cost-based model of SSL;
– a cost-based model of HIP protocol in Timed Coloured Petri Nets (Timed CP-Nets);
– simulation and analysis of HIP under normal conditions and under four scenarios of DoS attacks.

2 Background and Previous Work

The purposes of Section 2 are to provide the background on the Meadows's cost-based framework, SSL and HIP protocol, as well as the previous work on the analysis of security protocols using Coloured Petri Nets.

2.1 Meadows's Cost-Based Framework

Meadows framework [19] works by comparing cost to the attacker and cost to the defender, defined using a *cost set*. To model the protocol framework, we need to calculate the cost of the sequence of protocol actions, comparing between the attacker and the defender. Once the actions of each protocol principal are classified into the computational costs cheap, medium, or expensive, all actions of the protocol run can be compared. The protocol is secure against DoS attacks, if the final cost is great enough from the point of view of the attacker in comparison with the cost of engaging in the events up to an accepted action from the point of view of the defender. Otherwise, we conclude that the protocol is insecure against DoS attacks.

Considering the characteristic of DoS attacks, there are two possible ways mentioned by Meadows [19] to cause the defender to waste resources. First, the defender may process a bogus instance of a message inserted by the attacker into a protocol. The cost to an attacker is the cost of creating and inserting the bogus message, while the cost to the defender is the cost of processing the bogus message until an attack is detected. Second, the defender participates in a protocol execution with bogus instances of the attacker. The cost of this situation is equivalent to the cost of running the entire protocol until the defender can detect the attack or the attack stops.

At this stage, we limit the abilities of an attacker during the protocol execution to take one of a small number of possible actions when the protocol specifies that a message should be sent: the attacker either continues normally with the protocol or partially completes the protocol. Intuitively this is the most obvious way for an adversary to make the defender use unwanted resources. In our examples, the adversary sends messages at two points in the protocol; either attack the first message by flooding a large number of random messages to overwhelm the resources of the responder, or attack its second message by faking its packets to waste the responder resources for verifying it.

2.2 Protocol Notation

For the protocols presented in this section, we focus only on important elements for the simplification of the protocol description. Any data, such as header information, that are not relevant to the discussion of DoS resistance are omitted. For complete descriptions of the protocols, the reader is referred to the full specifications [12, 21]. The protocol notation used for the remainder of this section is presented in Table 1.

Table 1. Protocol Notation

Messages	**Notation**
I	The principal who initiates the request message known as Initiator or Client
R	The principal who responds to the request message known as Responder or Server
$H(M)$	Unkeyed cryptographic hash of the message M
$H_{K_{(\cdot)}}(M)$	Keyed cryptographic hash of the message M, with key $K_{(\cdot)}$
$E_{K_{(\cdot)}}\{M\}$	Symmetric encryption of message M with the secret key $K_{(\cdot)}$
$D_{K_{(\cdot)}}\{M\}$	Symmetric decryption of message M with the secret key $K_{(\cdot)}$
$PK_R[M]$	Asymmetric encryption of the message M by the public key PK_R belonging to R
$SK_R[M]$	Asymmetric decryption of the message M by the private key SK_R belonging to R
$Sig_I(\cdot)$	Digital signature signed by the private key SK_I belonging to the principal I
$Sig_R(\cdot)$	Digital signature signed by the private key SK_R belonging to the principal R
$LSB(t, k)$	Returns the k least significant bits of an output by taking a string t as input
0^k	A string consisting of k zero bits
p, q	Large prime numbers
i, r	A Diffie-Hellman secret parameter of I and R, respectively
g	Group generator of order q used in Diffie-Hellman key exchange and key agreement protocol
s	A periodically changing secret only known to the responder R
K_s	A session key generated by key establishment protocol used to secure ongoing communications
HIT_I, HIT_R	The host identity tag of I and R created by taking a hash over the host identity HI_I and HI_R
$Cert_R$	A certificate which contains a responder's identity and a public key used for authentication

2.3 Secure Socket Layer (SSL)

Secure Sockets Layer (SSL) is a well-known Internet protocol developed by Netscape [12] for establishing and transmitting secure data over the Internet. In order to establish a secure communication, an initiator and a responder have to negotiate the cryptographic algorithms and optionally authenticate each other by using public-key cryptosystems. SSL uses public-key encryption techniques not only for the mutual authentication, but also for the protection of a session key generated during the SSL protocol handshake. This session key is a short-term symmetrical key generated by an initiator and temporarily used during the secure communication. The description of the SSL handshake is illustrated in Figure 1.

I		R
1) create *ClientHello*	$\xrightarrow{\mathit{Client\ Hello}}$	create $Cert_R$
	$Cert_R, g^r,$	choose $r \in_R [1, 2, \ldots, q-2]$
2) verify $Cert_R$	$\xleftarrow{\mathit{ServerHelloDone}}$	compute g^i
choose $i \in_R [1, 2, \ldots, q-2]$		
compute g^i		
compute $K_s = H(g^{ir})$		
3) encrypt $PK_R[K_s]$	$\xrightarrow{PK_R[K_s]}$	decrypt $SK_R[K_s]$
4) verify $HMAC$	$\xleftarrow{HMAC}$	compute $HMAC = H_{K_s}(Data)$

Fig. 1. SSL Protocol: DH Key Agreement without Client Authentication Mode is used

To provide an explanation of the SSL handshake, we shall use the Diffie-Hellman key agreement without client authentication mode as an example because it is the simplest mode (most general users do not have a certificate). At the beginning of the SSL handshake, the initiator sends a *Client Hello* message to the responder for establishing a secure connection. Following, the responder responds with its certificate ($Cert_R$), and Diffie-Hellman key agreement parameter. Now the responder will send the *ServerHelloDone* message indicating that the responder completes the hello-message phase. The server will then wait for the initiator response.

In order to reply, the initiator has to send the *client key exchange* message which contains a session key (K_s). This session key is generated by using Diffie-Hellman key agreement. In order to send this key to the responder in a secure manner, the initiator uses the responder's public key (PK_R) provided in the responder's certificate to encrypt it; $PK_R[K_s]$.

To verify the initiator's message, the responder starts to decrypt the message; $SK_R[K_s]$, by using a private key SK_R to obtain a session key K_s. If this message is valid, the responder generates hashed-MAC ($HMAC$) by using K_s, and sends it to the initiator for performing a key confirmation. At this point, the SSL handshake is complete and the initiator and responder begin to exchange secure information.

2.4 Host Identity Protocol (HIP)

The host identity protocol (HIP) has been developed by Moskowitz [21]. Later, Aura et al. [3] found some vulnerabilities and proposed guidelines to strengthen the security of HIP. The base exchange of HIP is illustrated in Figure 2.

HIP adopts a proof-of-work scheme proposed by Jakobsson and Juels [15] for countering resource exhaustion attacks. In a proof-of-work, HIP extends the concept of a *client puzzle*, first proposed by Juels and Brainard [17], and later implemented by Aura et al. [5] for protecting the responder against DoS attacks in authentication protocols. Moreover, HIP allows the additional feature of the client puzzle that helps the responder to delay state creation [4] until the checking of the second incoming message and the authentication has been done in order to prevent the responder against resource exhaustion attack.

2.5 Coloured Petri Nets

Coloured Petri Nets (CPNs) [9,16] are one type of high-level nets based on the concept of Petri Nets developed back in 1962 by Petri [23]. CPNs is a state and action oriented model which consists of places, transitions, and arcs.

I		R
		Precomputed parameters
		choose $r, s \in_R [1, 2, \ldots, q-2]$
		sign $sig_{R1} = Sig_R(g^r, HIT_R)$
		....
1) create HIT_I, HIT_R	$\xrightarrow{HIT_I, HIT_R}$	check HIT_R
		compute $C = LSB(H(s, HIT_I, HIT_R), 64)$
2) verify sig_{R1}	$\xleftarrow[]{HIT_I, HIT_R, puzzle, g^r, sig_{R1}}$	choose $k \in [0, 1, \ldots, 40] \rightarrow puzzle = (C, k)$
Find J such that		
$LSB(H(C, HIT_I, HIT_R, J), k) = 0^k$		
choose $i \in_R [1, 2, \ldots, q-2]$		
compute $K_e = H(HIT_I, HIT_R, g^{ir}, 01)$		
encrypt $E1 = E_{K_e}\{HI_I\}$		
sign $sig_I = Sig_I(HIT_I, HIT_R, J, g^i, E1)$		
3)	$\xrightarrow{HIT_I, HIT_R, J, g^i, E1, sig_I}$	compute $C = LSB(H(s, HIT_I, HIT_R), 64)$
		check $LSB(H(C, HIT_I, HIT_R, J), k) \overset{?}{=} 0^k$
		compute $K_e = H(HIT_I, HIT_R, g^{ir}, 01)$
		decrypt $E1$
		verify sig_I
		compute $K_s = H(HIT_I, HIT_R, g^{ir}, 02)$
		compute $HMAC = H_{K_s}(HIT_I, HIT_R)$
4) verify sig_{R2}	$\xleftarrow[]{HIT_I, HIT_R, HMAC, sig_{R2}}$	sign $sig_{R2} = Sig_R(HIT_I, HIT_R, HMAC)$
compute $K_s = H(HIT_I, HIT_R, g^{ir}, 02)$		
check $H_{K_s}(HIT_I, HIT_R) \overset{?}{=} HMAC$		

Fig. 2. HIP Protocol [21]

Over many years, cryptographic and security protocols have been modeled and verified using Coloured Petri Nets [11, 14, 20, 22]. Neih and Tavares [22] implemented models of cryptographic protocols in the form of Petri Nets. In order to explore vulnerabilities of such protocols, they allowed an implicit adversary with limited abilities to launch attacks and then examined the protocol using exhaustive forward execution technique. Doyle [11] developed a model of three-pass mutual authentication and adopted the forward state-space searching technique from Neih and Tavares. Doyle allowed more sophisticated abilities of an adversary; multiple iteration and parallel session attacks. Han [14] adopted CPNs specification for constructing a reachability graph to insecure states and examining the final states in OAKLEY and the Open Network Computing Remote Procedure Call (ONC RPC) protocol. In 2004, Al-Azzoni [2] developed a model of the Needham-Schroeder public key authentication protocol and Tatebayashi-Matsuzaki-Neuman (TMN) key exchange protocol. In this work, Al-Azzoni introduced a new scheme to reduce the size of the occurrence graph for obtaining the practical result in the CPN programming called Design/CPN tool. By examining a reachability test to insecure states, Al-Azzoni found several flaws of such protocols. For instance, an adversary is able to impersonate a legitimate user during the protocol run without knowledge from such a legitimate user.

To the best of our knowledge, there is no implementation of CPNs focusing on an exploration of vulnerabilities based on unbalanced computation that might lead to DoS attacks in key exchange protocols.

3 Modeling Cryptographic Protocols with Coloured Petri Nets

We briefly explain the Coloured Petri Nets representation for constructing CPN models. The abilities of an individual adversary used in the simulation are also provided in this section.

3.1 CPNs Objects Description

Prior to demonstrating the model of cryptographic protocols, we shall describe the fundamental elements provided in the CPN Tools for constructing our cryptographic protocols. To implement a model with the concept of states and actions as illustrated in Figure 3, some important constructions of Timed CPNs are used including:

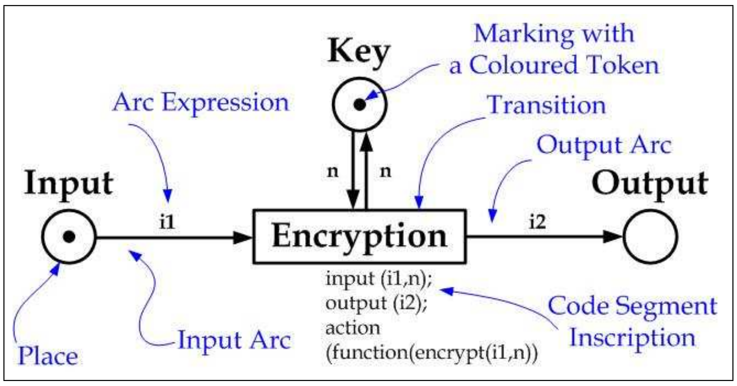


Fig. 3. An Example of CPNs Constructions

1. *Place*: is drawn as an ellipse or circle representing states of a system. Each place has a colour set which determines the type of data that the place can carry, e.g. type of users, type of messages.
2. *Type*: is used to specify the colour set of a token in each place. For example, a *User* colour set in our protocol consists of an honest client (`hc`), four types of adversary (`ad1-4`), and a responder/server (`sv`).
3. *Marking*: is a state of a Timed CPNs. It consists of a number of tokens positioned on the individual places. The marking of a place can be a multi-set of token values, for example, incoming requested packets from multiple honest clients and different types of adversaries arrive to the same server.
4. *Transition*: represents actions of a system, which is drawn as rectangles. A transition is connected with input places by incoming arcs and output places by outgoing arcs. Some examples of transition in our model are hash function and encryption algorithms.
5. *Arc*: is used to connect transitions and places. Each arc contains a weight value called an arc expression which represents the number of removed token from input places traveling to output places.
6. *Arc Expression*: determines the number of tokens which are removed from the input place and added to the output place during occurrences of transitions. This action represents a dynamic behaviour which can be seen as the traversing of messages in the cryptographic protocols.

3.2 Adversary's Ability

In the simulation, our goal is to explore unbalanced computational vulnerabilities in DoS-resistant protocols. As a result, we allow not only the honest client (`hc`) who initiates the legitimate traffic, and the responder who participates in the protocol execution, but also four types of adversary who have the same goal to deny the service of the responder by overwhelming the responder's resource. The major differences of the individual adversaries are:

Type 1 adversary (`ad1`) computes a valid first message (may be pre-computed in practice), and then takes no further action in the protocol. This type of adversary is used in the protocol simulation for both SSL and HIP.

Type 2 adversary (`ad2`) completes the protocol normally until the third message is sent and takes no further action after this. Type 2 adversary is used only in HIP in order to investigate the effect of the client puzzle. The computations of this adversary include searching a correct client puzzle solution J, generating a session key K_e and encrypting a public key PK_I, and finally computing a digital signature Sig_I.

Type 3 adversary (`ad3`) completes the protocol step one and two with the exception that the adversary does not verify the responder signature sig_{R1}. The adversary searches for a correct client puzzle solution J but randomly chooses the remaining message elements: an encrypted element $K_e\{HI_I\}$ and a digital signature sig_I. The adversary takes no further action in the protocol. This type of adversary is used only in HIP because SSL does not incorporate a client puzzle.

Type 4 adversary (`ad4`) attempts to flood bogus messages at the third step of the protocol by choosing the third message randomly. This type of adversary is used for both SSL and HIP.

To clarify the description of adversaries' ability, the major goal of an adversary type 1 is to overwhelm the server's storage by sending a large number of requested packets, for example, a denial-of-service attack via ping [7] and SYN flooding attack [8], while the major goal of an adversary type 2, 3, and 4 is to force the server to waste computational resources up to the final step of the digital signature verification and digital signature generation which are expensive operations.

During the protocol execution, each individual adversary has a specified number of requested tokens at the beginning. Moreover, our simulation allows all adversaries to re-generate new bogus messages as soon as they receive returned packets from the responder. That means adversaries have the power to constantly flood new bogus messages to deplete the connection queue. This kind of attack can be considered as ping flooding attacks [7], or TCP SYN flooding and IP spoofing attacks [8]. However, allowing this unlimited ability to adversaries might cause more advantages over honest clients and a responder because adversaries are able to launch such attacks until the responder gets congested and terminates. Therefore, those adversaries are able to deny any services to any websites by constantly flooding bogus messages with unlimited power. That might lead to a difficult task not only for the defender to protect the communication network from DoS attacks, but also the protocol engineering to develop efficient protocols to resist such attacks.

In order to model DoS adversaries effectively, we need to limit the power of the adversary in some ways. One possible way to limit adversaries is to specify the attack timing period. This approach has an obvious effect because the percentage of throughput will be much less during the attack, meanwhile the output becomes the normal level when there is no attack in the network. Another approach is to specify the resource in order to perform the attacks as well as to limit the capacity such as CPU, memory, or bandwidth of adversaries. This ensures that adversaries must have enough resources available for launching attacks. The latter approach seems more interesting and useful to implement than the former one because adversaries are able to perform attacks at any time as long as they have available resources.

In our model, adversaries are able to perform the number of attacks depending on the available resources specified at the initial state during the protocol simulation. Before launching the new attacks, adversaries have to wait for a return message (token/available resource) from the responder. In normal situation, the number of returned messages will be equivalent to the number of messages that the responder receives. That means adversaries still have the same level of capability to perform DoS attack as long as the responder can serve those packets. However, once the responder is in a full-loaded condition, the responder starts to reject next arriving messages from any principals that causes adversaries to lose their packets (tokens) from the system. Some may argue this is not fair to the adversary; however, if we do not limit the DoS attack ability, the responder is always in a full-load condition and unable to serve any legitimate users, so we are unable to measure the tolerance of any key exchange protocols for resisting DoS attacks (because this is the way to examine and evaluate protocols). Furthermore, the most important reason is that the major goal of the protocol designer who implements cryptographic DoS-resistant protocols is to prevent adversaries who attempt to exploit vulnerabilities of such protocol itself to attack against legitimate users on that protocols.

As a result, it might difficult or impossible to implement only cryptographic DoS-resistant protocols to deal with flooding attacks for degrading throughput of services without any helps from other protection mechanisms such as intrusion detection systems (IDS).

4 Cost-Based Framework

In this section, we provide a cryptographic benchmark of some specific algorithms. This could be one promising technique used to measure CPU usage which alternatively be used to represent more specific computational costs instead of an original representation. We can use the total computations for comparing a cost of operations between an initiator and a responder as stated by Meadows. Finally, we present examples of the SSL and HIP cost-based framework.

4.1 Refinement of Meadows's Framework

An obvious limitation of the original formulation of the framework is that the computational costs are not defined precisely, but consist instead of a small number of discrete values. Indeed Meadows herself called this a "crude and ad hoc cost function" [19]. In order to obtain a more useful cost comparison we need to obtain a more accurate estimate of the computational and/or storage costs required to complete the protocol steps. How to do this is not as obvious as it may seem at first.

When comparing efficiency of different cryptographic protocols is it customary to count the number of different types of cryptographic operations. For protocols that use public key operations it is common to ignore most operations and count only the most expensive ones, which typically are exponentiations in different groups (traditionally written as multiplications in elliptic curve computations). However, for the protocols that interest us this is definitely not acceptable. As mentioned above, one common technique in DoS prevention is to demand that clients solve *puzzles* which require the client to engage in some computational effort, such as to iterate a hash function a large number of times. Although one hash computation takes minimal time in comparison with a public key operation, ignoring a large number of hash computations could make the cost function ignore completely the DoS prevention mechanism when a puzzle is used. Therefore we need to be able to compare directly the cost of all different kinds of cryptographic operations.

Comparing operations like hashing and exponentiations directly seems very hard to do since they are based on completely different types of primitive instructions. Therefore we have resorted to an empirical comparison which compares benchmark implementation on common types of processors. While we acknowledge that the detailed results may differ considerably for different computing environments (CPU, compilers, memory, and so on) we believe that the obtained figures are indicative of the true cost in typical environments and allow reasonable comparisons to be made.

For our cost estimates, we use the cryptographic protocol benchmarks of Wei Dai [10]. These include tested speed benchmarks for some of the most commonly used cryptographic algorithms using Crypto++ library[1] version 5.2.1 on a Pentium 4 2.1 GHz processor under Windows XP SP 1. More cryptographic benchmarking has been done by Gupta et al. [13] and by Tan et al. [26] on specific processors; however, they did not test public-key encryption.

Table 2 presents the results for some specific cryptographic algorithms available for negotiating during the three-way handshake on the SSL protocol and the HIP based exchange defined in HIP specification. The units that we use in Table 2 are kilocycles per block (note that block size varies for different algorithms). This allows direct comparison of CPU usage and may be expected to be similar on processors with different clock speeds. This entails conversion from the original data which uses direct time measurements.

From the table, we are able to estimate the CPU usage in cycles per block for common hash functions and the symmetric key encryption, and cycles per operations for the 1024-bit key lengths of public-key encryption and Diffie-Hellman key exchange algorithm. Once we get a result, we scale it down by a factor

[1] available at `http://www.eskimo.com/~weidai/cryptlib.html` and `http://sourceforge.net/projects/cryptopp/`

Table 2. Computational Cost of CPU and Time Usage of Specific Algorithms

Hash	kCycle/Block	nsec/bit	Symmetrical Crypto	kCycle/Block	nsec/bit
SHA-1 (512bits/block)	1.89	1.84	**DES** (64bits/block)	0.75	5.86
MD5 (512bits/block)	0.59	0.58	**Blowfish** (64bits/block)	0.25	1.94
HMAC/MD5 (512bits/block)	0.59	0.58	**AES** (128bits/block)	0.53	2.05

Public-Key Crypto	kCycle/ops	nsec/bit	Key Exchange	kCycle/ops	nsec/bit
RSA Encryption/Verification	383.66	187.08	**Diffie-Hellman Key-Pair Generation**	4605.65	2245.80
RSA Decryption/Signature	9985.47	4869.11			
DSA Signature	4569.62	2228.23	**Diffie-Hellman Key Agreement**	8100.69	3950.05
DSA Verification	5239	2554.64			

of 1000 (kilo) and apply these costs in our formal specification and analysis. Before we can export these values into CPN Tools, we round them into an integer representation because CPN Tools uses integers in the simulation process.

4.2 Experiment 1: SSL Cost-based Model

Figure 4 shows the simulation of the SSL protocol. As SSL is modeled hierarchically for simplicity of the model and simulation, all nodes in the top page are related to individual subpages defined by the SSL specification. The top page consists of three network segmentations; the initiator (could be either the honest client who performs as a protocol specification or the adversary who does not play honestly), the responder of the protocol, and the communication network. At this state, we do not permit the adversary to reuse the previous messages to attack the responder, i.e. when the adversary attempts to flood new bogus messages, the adversary has to participate in the construction by computing individual messages.

Because SSL protocol is modeled in the cost-based framework, every single state has been attached with the computational cost-place for displaying the operational effort of that state during the protocol execution. During the protocol execution, we record all operational costs of individual transitions by adding CPU usage data from Table 2.

At the beginning, there are three types of user who can request for services; the honest client (`hc`), the adversary type one (`ad1`) who attempts to attack the protocol by flooding initial request messages, and the adversary type four (`ad4`) who attempts to attack the protocol at the third messages, indicated in the user token. Moreover, we are able to specify the number of messages sent simultaneously from the initiator, define the string of messages, as well as investigate the computational cost[2] when the message travels to each operation in this token.

To examine our cost-based model, the initiator starts sending a request message to the responder. At the initial phase, there is no operation to make a cost to the message. Once the responder receives a request

[2] It is important to note that the display cost at each state shows the total operation cost of that corresponding state only, not an accumulation cost of all state. The reason is that it is easy to compare the cost of message construction on the initiator's machine with the cost of protocol engagement on the responder's machine at every single step of the protocol as suggested by Meadows [19].

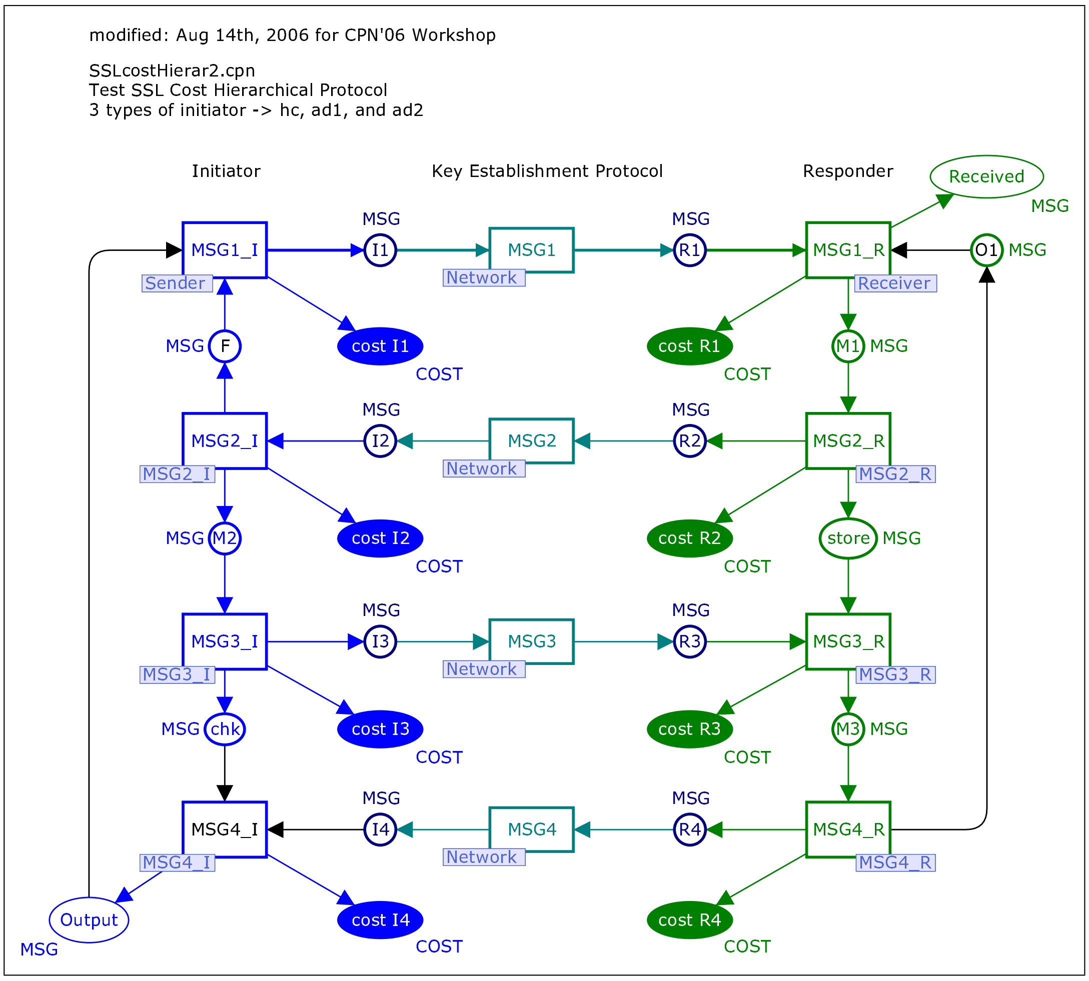


Fig. 4. SSL Cost-based Construction

message, the responder has to choose the Diffie-Hellman (DH) parameter used for generating a session key and returns the certificate in the second message. At this step, the responder has to store the received information and open the connection until the responder receives the replied message from the initiator because SSL session is a stateful protocol. This condition is subject to a flooding attack that presents a risk to the responder to exhaust its connection queues. The stored information during the protocol run is shown in the store-place under the MSG2_R position in the top page (figure 4).

Upon receipt of a reply message, the initiator has to select a DH value for calculating a session key used to protect the communication. The initiator is also required to verify the responder's certificate and subsequently to extract the public key for encrypting the DH value. At this phase, the adversary might send a large number of bogus messages to the responder to exhaust the responder's computational resources used for verifying the initiator's identity if the client authentication mode is selected. If no client authentication is required by the responder, Type 4 adversary is able to choose message 3 randomly for depleting the responder's resource more easily. That is because the responder has to waste its resources to decrypt message 3 which is encrypted using the public-key cryptosystem. Note that the RSA decryption algorithm takes far more CPU usage than RSA verification.

In the final step, the responder obtains the session key by using its private key to decrypt the initiator's message. If this process is successful, the responder uses this session key to produce a hashed-MAC (HMAC) and returns it to the initiator for a key confirmation. Once the initiator receives this message, the initiator decrypts it to check the correctness of the key.

Authentication Protocol	**Initiator**		**Responder**	
			with Client Authentication	without Client Authentication
SSL	***hc***	5376	14978	14594
	ad1	0	4606	4606
	ad4	0	4990	14592

Table 3. Comparison of the SSL Computational Cost with and without Client Authentication

As described above, SSL has two vulnerabilities to the DoS attacks at two states, following message 1 and message 3. The first vulnerability is demonstrated in Figure 4 at the store position (at the middle right under the transition MSG2_R). In the second vulnerability, the analyst can see the computational cost from Table 3 when comparing the cost of computation between the initiator and the responder. The total cost for the responder is greater than for the initiator because the responder has to participate with the RSA decryption, which is an expensive operation, in the third phase. Meanwhile the adversary does not spend resources to verify the second message in the second step and employs only cheap computations (because we have not defined the cost of the adversary to reuse, spoof, insert, or interrupt the message) to send the third message to deny service.

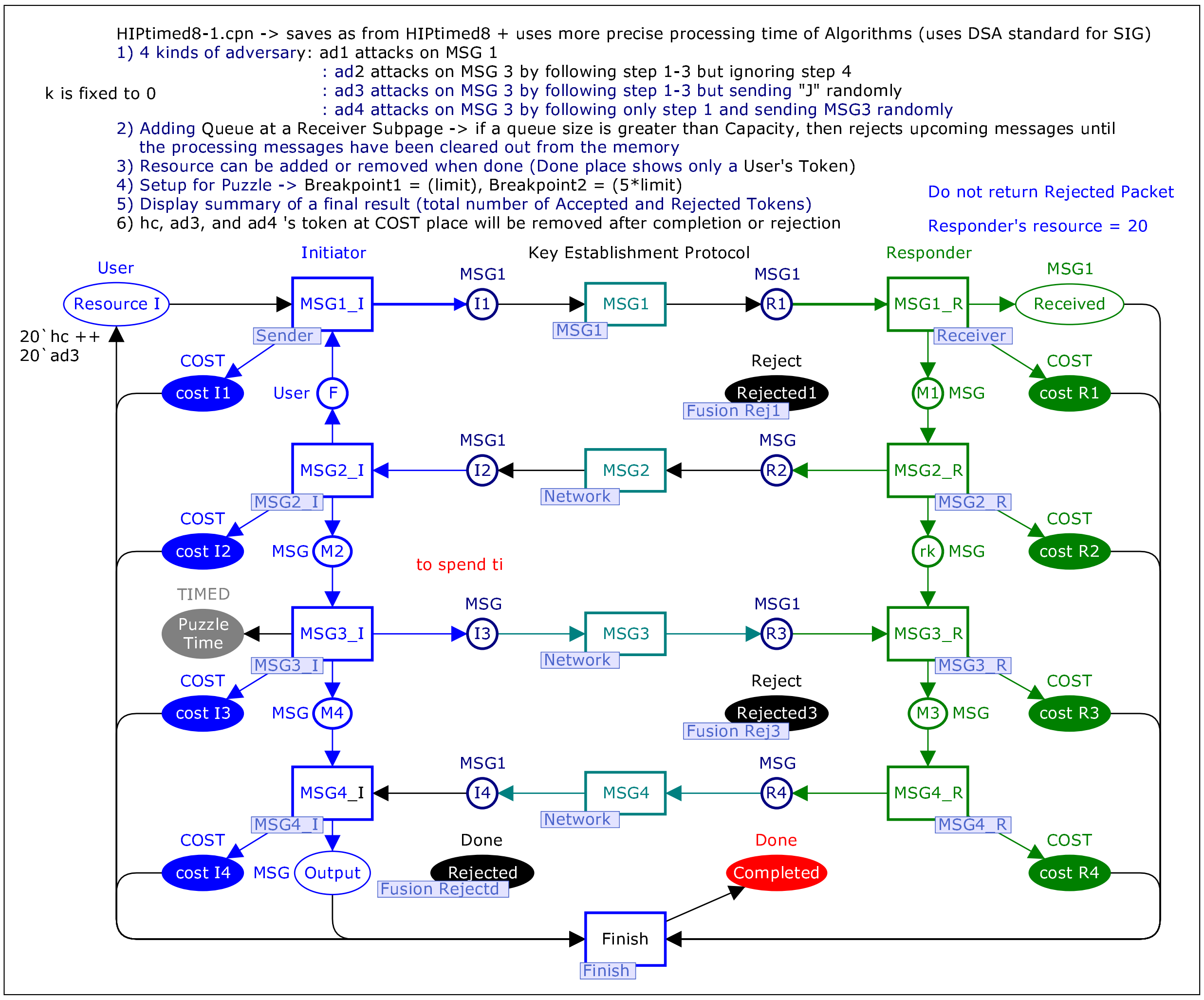
To sum up, as SSL is designed with message efficiency rather than resistance to DoS attacks in mind, these situations reveal the denial-of-service threats to the responder. Recently, there are several well-known techniques used as a denial-of-service tool to attack the communication running over the SSL protocol. One example is a SYN flooding attack [8] in which the adversary requires little computational effort for constructing bogus messages, while the responder requires greater magnitude compared to the adversary for handling these messages.

4.3 Experiment 2: HIP Cost-based Model

The purpose of this simulation is to compare computational cost of the protocol execution between all principals with some possible ranges of puzzle difficulty (k) including $k = 0$ (which means that no puzzle is required), easiest value $k = 1$ for contrasting the difference between `ad3` and `ad4`, intermediate values $k = 10$, $k = 20$, and $k = 40$ for a hardest value as instructed by the designer. Similar to the SSL-model, we insert a cost-place to individual transition for displaying a computational cost of every single step. A measurement of CPU usage has been used to indicate individual steps and compare the total cost among an initiator, individual adversary type, and a responder as a key concept of cost-based analysis specified by Meadows.

In this simulation, we allow individual initiator to initiate a request token only once, while the responder is able to flexibly adjust the puzzle difficulty within defined values. Once the simulation has arrived to the final step, we record a total computational cost of individual user comparing to the responder on specified ranges of k. The HIP cost-based model is demonstrated in Figure 5.

During the protocol execution, the initiator sends a request message to the responder using the host identity tag (HIT) which is a hash of the host identifier (HI) used in HIP payload and to index the corresponding state in the end hosts. Therefore, the initiator only employs cheap operations at the beginning step. We assume that the computation at this step can be precomputed, so, the cost at the first operation would be negligible. Once the responder receives the requested message, the responder requires a hash operation and some values from the precomputation for returning to the initiator in the second step. This operation is estimated as a cheap operation similar to the initiator.



HIPtimed8-1.cpn -> saves as from HIPtimed8 + uses more precise processing time of Algorithms (uses DSA standard for SIG)
1) 4 kinds of adversary: ad1 attacks on MSG 1
: ad2 attacks on MSG 3 by following step 1-3 but ignoring step 4
: ad3 attacks on MSG 3 by following step 1-3 but sending "J" randomly
: ad4 attacks on MSG 3 by following only step 1 and sending MSG3 randomly
k is fixed to 0
2) Adding Queue at a Receiver Subpage -> if a queue size is greater than Capacity, then rejects upcoming messages until the processing messages have been cleared out from the memory
3) Resource can be added or removed when done (Done place shows only a User's Token)
4) Setup for Puzzle -> Breakpoint1 = (limit), Breakpoint2 = (5*limit)
5) Display summary of a final result (total number of Accepted and Rejected Tokens)
6) hc, ad3, and ad4 's token at COST place will be removed after completion or rejection
Do not return Rejected Packet
Responder's resource = 20
User
Initiator
Key Establishment Protocol
Responder
Resource I
MSG1_I
MSG1
I1
MSG1
MSG1
R1
MSG1_R
MSG1
Received
20`hc ++
20`ad3
Sender
MSG1
Receiver
COST
cost I1
User
F
Reject
Rejected1
Fusion Rej1
M1
MSG
COST
cost R1
MSG1
MSG
MSG2_I
I2
MSG2
R2
MSG2_R
MSG2_I
Network
MSG2_R
COST
cost I2
MSG
M2
rk
MSG
COST
cost R2
to spend ti
TIMED
Puzzle Time
MSG3_I
MSG
I3
MSG3
MSG1
R3
MSG3_R
MSG3_I
Network
MSG3_R
COST
cost I3
MSG
M4
Reject
Rejected3
Fusion Rej3
M3
MSG
COST
cost R3
MSG1
MSG
MSG4_I
I4
MSG4
R4
MSG4_R
MSG4_I
Network
MSG4_R
COST
cost I4
MSG
Output
Done
Rejected
Fusion Rejectd
Done
Completed
COST
cost R4
Finish
Finish

Fig. 5. HIP Cost-based Construction

When the initiator receives the replied message in the MSG2_I subpage, the initiator first verifies the HIT and responder's signature. There are three possible outputs after verification depending on the user field; 1) if the user is hc, the token will traverse to accept-place and the cost is equal to the HIT verification plus signature verification, 2) if the user is ad2, ad3, and ad4, the token will also traverse to accept-place but the cost is zero because it performs nothing at this step, and 3) if the user is ad1, the token will traverse to fail-place because ad1 does not take any further actions after the first message has been sent, therefore, the computational cost of ad1 is zero for the second stage.

The operations in message three of the initiator include the brute-force search to find the puzzle solution, and the key generation. The cost of solving a puzzle depends on the value of k including $k = 0$, $k = 1$, $k = 10$, $k = 20$, and $k = 40$ in the puzzle message field. However, hc, ad2, and ad3 are required to solve the puzzle solution. Like ad1, the ad4 does not attempt to solve the puzzle; as a result, the puzzle difficulty does not affect to the computational cost on this type of adversary. Another important thing to note is that, the cost of the adversary to spoof, insert, or interrupt the message has not been defined in this phase. So, we set the cost of randomly chosen messages in the case of ad3 and ad4 to be zero.

Considering the responder's task, when the responder receives the third-step message from the initiator, the responder begins to validate the puzzle solution which is defined as a cheap authentication process because the responder performs only one hash calculation. If it is invalid, the process will stop and the responder will drop the corresponding packet from the connection queue (the system will return a resource to the responder). Otherwise, the responder performs the decryption to obtain an initiator's public key. The responder finally verifies the signature by using the initiator's public key obtained in the previous step. The result would be either valid or invalid. After the authentication has been completed, the responder and the initiator will perform a key confirmation and start to exchange information. Table 4 summarizes the computational cost when the puzzle difficulty is set to k=1 or k=10 comparing between every principal and the responder. The experimental result shows that the most effective adversary is ad3 (the greatest different threshold between ad3 and the responder) because ad3 can force the responder to engage in the expensive tasks, i.e. digital signature verification.

Table 4. Comparison of Computational Cost of HIP with k=1 and k=10

Authentication Protocol	**Initiator**			**Responder**		
		k=1	**k=10**	J,$E1$,sig_I valid	only J valid	Everything invalid
HIP	***hc***	19973	22017	19591	-	-
	ad1	0	0	-	-	2
	ad2	14982	17026	19591	-	-
	ad3	4	2048	-	4998	-
	ad4	0	0	-	-	6

Figures 6 illustrate the computational cost of the honest client, Type 2, and Type 3 adversary, respectively. In the comparison charts, we measure the cost of those users involved with solving the puzzle of the difficulty level $k = 0, 1, 10, 20, 40$.

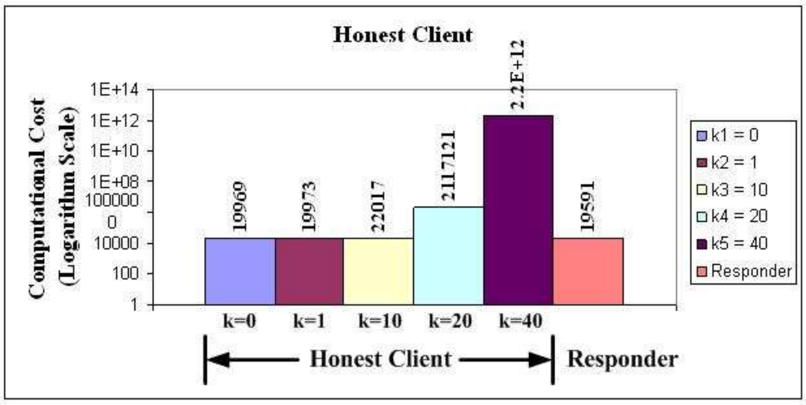
Comparison between Figures 6(a) and 6(b) shows that hc and ad2 incur similar computational costs for the same value of k chosen. This illustrates well the effectiveness of HIP in achieving its aims in resisting DoS attacks, at least against this type of adversary. On the other hand, ad3 and ad4 spend very small computational resources compared with the responder because both adversaries use some random message elements. This situation would bring the responder to the risk of DoS attacks if the value of k is not chosen properly. Figure 6(c) indicates that a value of k a little above 10 would be appropriate to ensure that ad3 uses as much computation as the responder.

5 Timed Coloured Petri Nets (Timed CPNs)

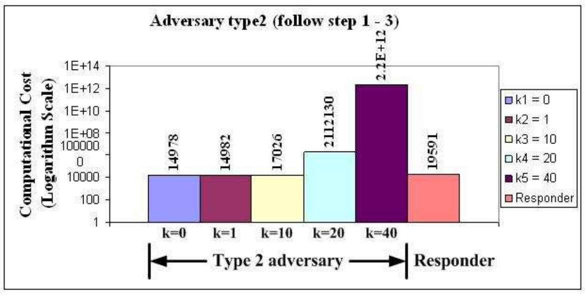
We attempt to design more realistic models of cryptographic protocols by adopting the concept of Timed Petri Nets into our implementation, i.e all cryptographic processes require some amount of time calculated by using cryptographic benchmark of Crypto++ library developed by Wei Dai [10]. In the Timed Petri Nets, the concept of *the simulated time* or *the model time*[3], which is represented by the system clock in the tool, has been introduced. Once we have attached the system time into tokens, we can see the sequence or action of states that tokens move to as a temporal analysis. That means only tokens which hold the current time as presented on the clock can be moved to the next step, while the others have to wait until the system clock reaches their value.

To develop a model on CPN Tools, HIP is constructed hierarchically for simplicity of the model and simulation. All nodes in the top page are related to individual subpages defined by the HIP specification [21].

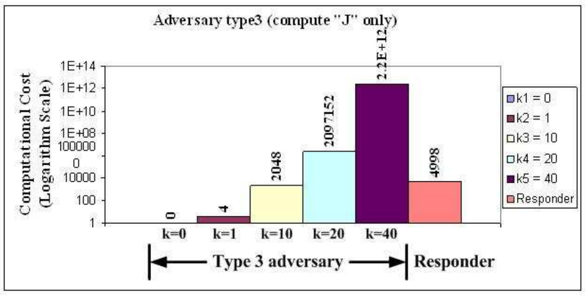
[3] More formal descriptions are available on the official website of CPN Tools, `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`



(a) Computational Cost between `hc` and `R`



(b) Computational Cost between `ad2` and `R`



(c) Computational Cost between `ad3` and `R`

Fig. 6. Comparison of Computational Cost on HIP with different ranges of k

HIP is modeled in the cost-based framework, so each state has the computational cost place to display the total cost of that state during the protocol simulation. Furthermore, the concept of the responder's resource is used in this evaluation. Once the responder has to deal with requests, the responder spends one resource for an individual request. If incoming packets exceed the responder capacity, the responder then rejects the further incoming packets until he has either processed the legitimate traffics or detected bogus messages and removed them from the storage.

In our model, we initially configure an individual message to contain four coloured sets; 1) *User* who initiates the token, 2) *NUM* which is the number of messages sent simultaneously from the initiator, 3) *DATA* which indicates the string of messages, and 4) *COST* which is used to display the computational cost when the message is traveling to each operation. Similar to other models, the display cost at each state shows the total operation cost of that corresponding state only, not an accumulation cost of all state. The top page of HIP Timed-CPNs is constructed as demonstrated in Figure 5.

From Figure 5, the top page consists of three major segments; 1) an initiator's network, 2) a communication channel, and 3) a responder's network. Each transition represents the stage of protocol execution, which consists of four stages in each principal because HIP is a four message protocol, corresponding to a specified subpage. Each stage consists of CPN elements constructed as specified in HIP protocol specification [21]. An example of responder's subpage at the first stage is demonstrated in Figure 7

This responder subpage consists of two main important transitions which are used for arranging the order of requested messages and verifying the validity of the responder's host identity tag (HIT_R). Considering the *Queue* and *Count* transition, the purpose of these transitions is to measure the number of arriving requested packets in order for the responder to flexibly and appropriately adjust the puzzle difficulty associating to the number of workloads. Moreover, in the situation when multiple requested messages arriving to this transition simultaneously, the process of arranging the order of such packets is random based on CPN Tools.

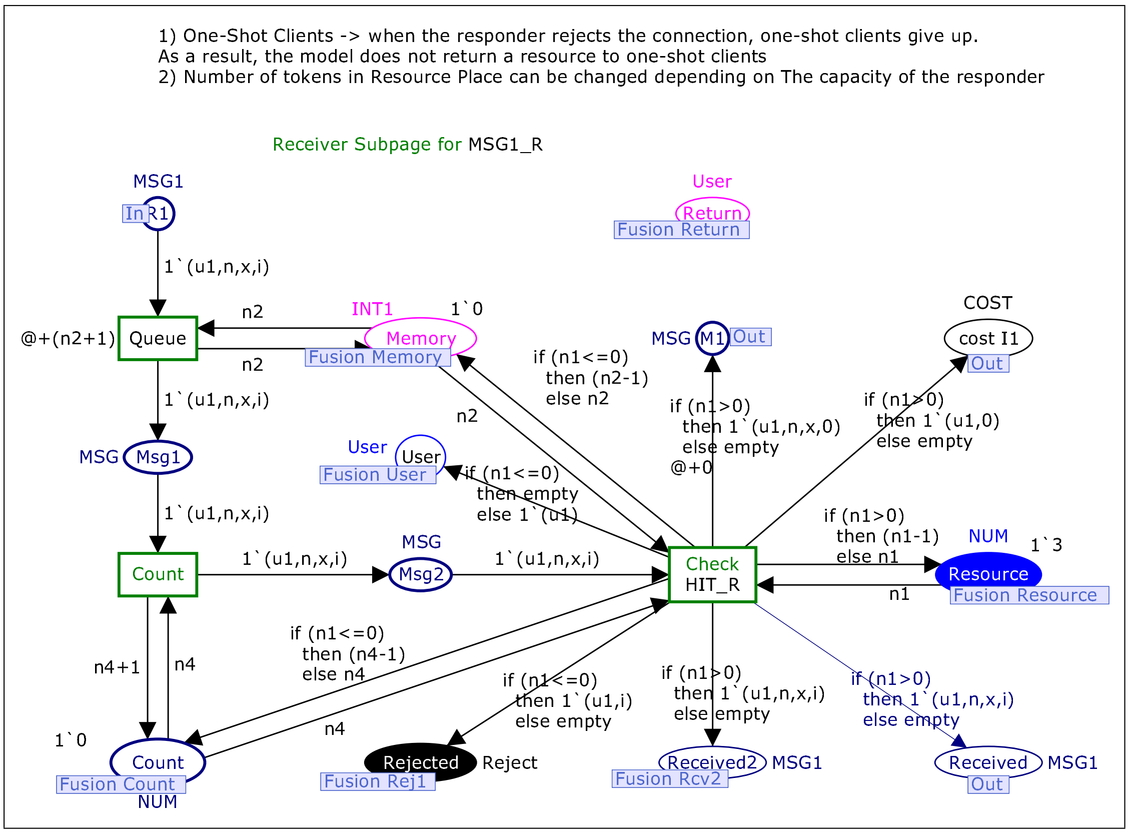

1) One-Shot Clients -> when the responder rejects the connection, one-shot clients give up.
As a result, the model does not return a resource to one-shot clients
2) Number of tokens in Resource Place can be changed depending on The capacity of the responder
Receiver Subpage for MSG1_R
MSG1
In
R1
1`(u1,n,x,i)
@+(n2+1)
Queue
n2
INT1
1`0
Memory
Fusion Memory
n2
User
Return
Fusion Return
MSG
M1
Out
COST
cost I1
Out
1`(u1,n,x,i)
MSG
Msg1
User
User
Fusion User
if (n1<=0) then (n2-1) else n2
n2
if (n1>0) then 1`(u1,n,x,0) else empty @+0
if (n1>0) then 1`(u1,0) else empty
if (n1<=0) then empty else 1`(u1)
1`(u1,n,x,i)
Count
1`(u1,n,x,i)
MSG
Msg2
1`(u1,n,x,i)
Check HIT_R
if (n1>0) then (n1-1) else n1
NUM
1`3
Resource
Fusion Resource
n1
n4+1
n4
if (n1<=0) then (n4-1) else n4
n4
if (n1<=0) then 1`(u1,i) else empty
if (n1>0) then 1`(u1,n,x,i) else empty
if (n1>0) then 1`(u1,n,x,i) else empty
1`0
Count
Fusion Count
NUM
Rejected
Reject
Fusion Rej1
Received2
MSG1
Fusion Rcv2
Received
MSG1
Out

Fig. 7. The Responder's Subpage at the First Stage

Another transition in this subpage is $CheckHIT_R$. In order to process the job, the responder has to have available memory resource. Otherwise, the responder will reject requested messages until some of them are removed from the connection queue. In order to define the responder's capability, we insert a *resource* place in this stage. We have to initially specify the number of available connection queue size before activating the simulation. This number represents the responder's capacity (in the case of *memory* resources) in order to serve initiator's messages simultaneously without degradation of services. Note that each packet requires only one resource (one connection queue) during the process at the responder's machine. Once those messages are processed and removed from the connection queue, they will return resource tokens to the responder's machine.

In addition, the quantity at the resource place represents not only available connection queue, but indirectly represents *CPU* usage on the responder's machine as well. We shall explain this concept by giving an example. Compare two messages in which message one is in stage one, while message two is in stage three. The responder has to spend a similar amount of connection queue, a token per message, for serving both of them. However, in the case of CPU usage, the responder has to waste more power for message two than message one because the main task at stage three is to verify the puzzle solution and the signature, while the task at stage one is only choosing the puzzle difficulty and returning it to the initiator. By specifying the time usage from Table 2 for individual cryptographic transitions, we can infer that the longer period that the message is processed in the responder's machine, the more CPU usage it takes from the responder.

5.1 Experiment 3: Non-adjustable Client Puzzles

The purpose of the third experiment is to examine the minimal DoS-resistant mechanism. To achieve this, we run the simulation under four specified attacks and the combination of four strategies (defined as *All*) with the non-adjustable client puzzle. We initially fix k=1, i.e. the easiest value[4], because hc prefers to spend nothing expensive for establishing a connection under normal circumstances.

Different from the experiment of a HIP cost-based model, we allow a responder to participate with a pair of initiators (hc and an individual type of ad). We assume that the responder has to deal with different strategies of adversary and different amounts of packets which consist of both legitimate and bogus messages. Considering the number of packets, hc can initiate the amount of requests (C) at 80%, 100%, and 150% of the responder's capacity (R). Meanwhile, a single type of ad can flood the amount of bogus requests (Z) at 100%, 200%, and 1000% of the responder's capacity (R).

In order to examine the tolerance of HIP protocol under different attack strategies, each individual adversary has been made a pair with an honest client during the protocol execution. Under four specified attacks, there are three possible situations to cause a responder to waste computational resources by the adversary.

1. All values of the third message including a puzzle solution J, an encrypted part $K_e\{HI_I\}$, and a digital signature sig_I are valid. This will force the responder to process a gradual authentication and complete the final step of the communication.
2. Only the client puzzle solution J is valid. This situation also causes the responder to perform the puzzle solution verification, decryption, and the signature verification. The responder can detect the attack only at the final step of authentication.
3. The client puzzle solution J is invalid, so the responder computes only a cheap hash function to verify the solution and then this connection will be terminated whether the remaining messages are valid.

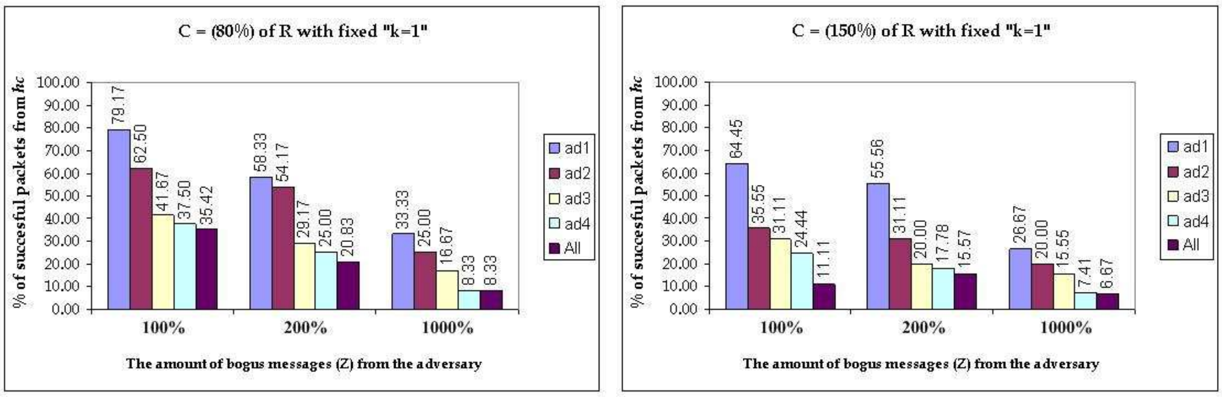
Finally, by inserting places for displaying the number of completed and rejected messages at the responder's network, the number of successful legitimate requests that the responder can serve under different adversary's abilities is measured as the percentage for the protocol evaluation.

The Experimental Results: Figure 8 represents the percentage of successful legitimate connections compared among three different amounts of bogus messages (Z=100%, 200%, and 1000% of the responder's capacity R) from five adversarial strategies (in the combination strategy, *All*, each adversary type has the same amount of bogus messages that makes the total number equivalent to the specified quantity). When we prohibit the responder's ability to adjust k, the percentage of successful messages from hc to obtain service will drop drastically when ad increases the number of bogus messages. Comparing different types of ad, the most effective is ad4 who sends bogus messages to the responder by crafting messages randomly. This is because ad4 can flood a large number of messages to overwhelm the responder's resource quicker than the others, which causes the responder to reject the next incoming messages from hc. Although packets from ad4 will be detected and discarded by the responder, these packets can be re-generated and flooded by ad4 as soon as ad4 receives returned packets from the responder at phase two.

Comparing ad1 and ad4, even though both of them craft random messages, ad4 can achieve the goal at higher rate than ad1 because the responder can process the incoming request at step 1 and clear a queue faster than at step 3. At step 1, the responder only participates in the protocol by choosing the puzzle difficulty (k) and pre-computed information, and returns it to ad1. Although, ad1 can re-generate bogus messages after receiving replied messages, this does not cause the responder to reject a large number of messages because HIP mitigates such problem by adopting a stateless-connection. On the other hand, the task of ad4, to fill-up the responder's queue at step 3, can be achieved more easily than ad1 because the process of checking a puzzle solution and a digital signature takes longer than the whole process at step 1.

Considering ad2 and ad3 who attempt to deny service at phase 3 by computing the puzzle solution, the results show that ad3 succeeds at higher proportion than ad2. This is because ad3 can flood attack messages faster than ad2 who must engage in the correct generation of message two. Nonetheless, both adversaries can force the responder to engage in the signature verification. Although ad4 can flood a large number of messages at step 3 as well as ad2 and ad3, ad4 cannot force the responder to engage in expensive operations

[4] If we choose k=0 which means no client puzzle is required, we cannot see the difference of costs between ad3 and ad4 because the task of both adversaries will be the same.



(a) hc's load = 80% of R (b) hc's load = 150% of R

Fig. 8. Percentage of throughput from hc with k=1

because the responder is able to detect the message forgery at the cheap puzzle verification process. However, without the assistance of puzzle difficulty, the percentage of successful messages in the case of hc and ad4 is lower than the others because ad4 floods message three at the highest rate. As a result, the most effective adversary to deny service to the responder would be ad4 that attacks the verification phase. Most key agreement protocols incorporate verification tasks that would be susceptible to resource exhaustion attacks.

Finally, the result of the combination of all attack techniques shows that when the responder has to deal with all types of adversary, the percentage of legitimate users served by the responder will fall significantly with increment of bogus messages. Once we can identify the most effective scenario, we will apply this technique to the fourth experiment for investigating the usefulness of puzzle difficulty.

5.2 Experiment 4: Adjustable Client Puzzles

The purpose of this experiment is to measure toleration of the responder when adjustable client puzzles are implemented. The result can be used to compare with the experiment on the cost-based model for confirming whether Meadows's cost-based framework is efficient for evaluating the DoS-resistant protocols or not.

To examine the protocol, we allocate two possible values, $k = 1$ and $k = 10$ for the responder. We choose those two values because they do not add computational effort to hc and the total task is still in the acceptable threshold comparing to tasks on the responder (see Figure 6(a)). In order to allow the responder to flexibly adjust puzzle difficulty between those two values more efficiently, we simply insert a counter into the model for measuring the condition of a responder's workload. Once the workload has reached the maximum tolerance, the responder will increase the puzzle difficulty to the higher level for delaying the incoming rate of requested messages.

At the beginning of the protocol assessment, we allow both hc and an individual type of ad to make requests at the same time as the responder. However, the responder is able to process requests only one message at a time. So, a concept of the responder queue is implemented for arranging an order of incoming packets. Figure 9 illustrates a construction and a simulation of the HIP protocol by means of Timed CPNs.

Considering the initiator's packet, there are different actions from initiators and the responder during the protocol run. The hc initiates a request only once and keeps waiting to process next steps. This delay is described by means of Timed CPNs, i.e. every transition which relates to cryptographic operations is defined as a timed process. During the simulation, if requests from hc have been rejected under DoS circumstances, hc gives up to open another session. On the other hand, there are two different situations that packets from ad are rejected by the responder; 1) the responder detects the bogus messages during the verification steps, and 2) the responder does not have enough resources for serving any requests. Once the responder detects the attack and rejects those packets, ad will lose those packets from the system.

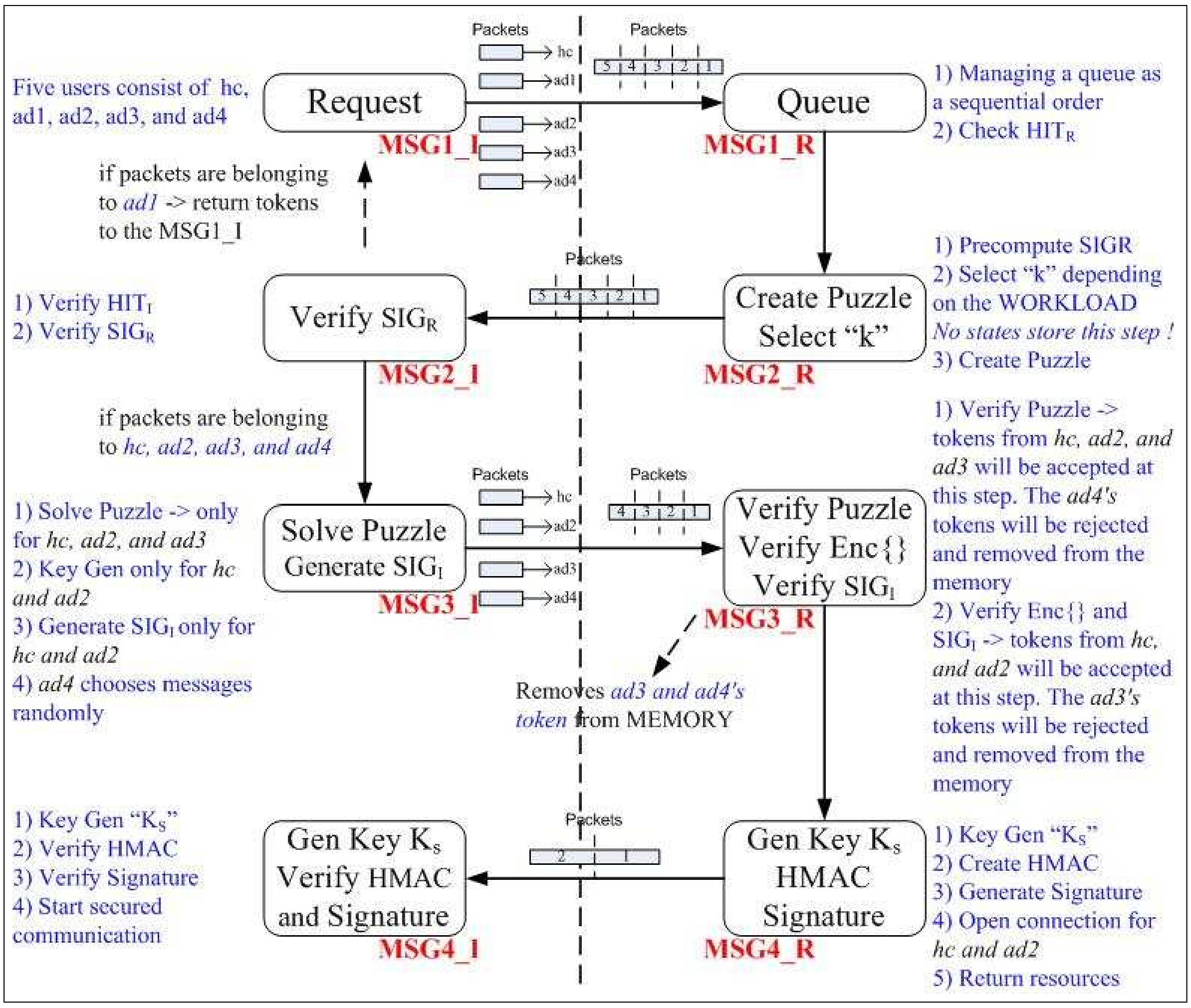
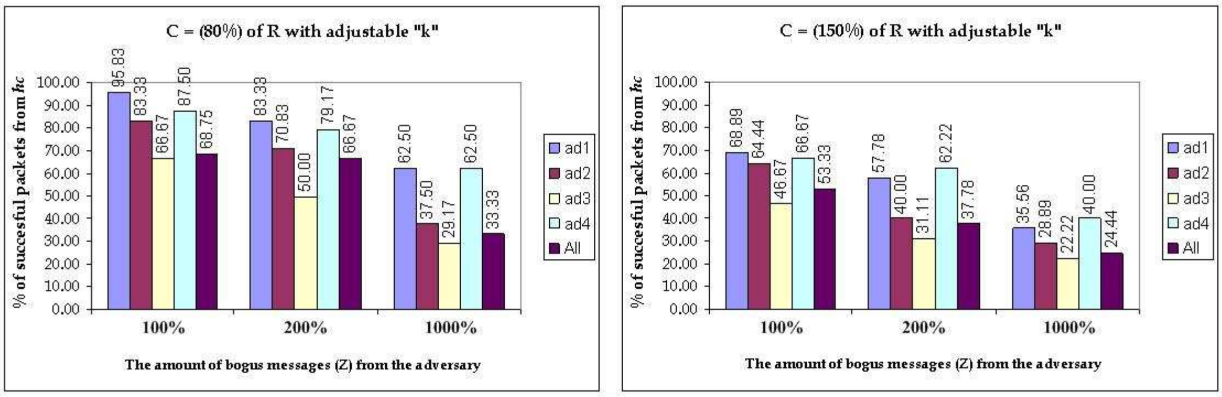

Five users consist of hc, ad1, ad2, ad3, and ad4
Request
MSG1_I
Packets
hc
ad1
ad2
ad3
ad4
Queue
MSG1_R
1) Managing a queue as a sequential order
2) Check HIT_R
if packets are belonging to ad1 -> return tokens to the MSG1_I
1) Verify HIT_I
2) Verify SIG_R
Verify SIG_R
MSG2_I
Create Puzzle Select "k"
MSG2_R
1) Precompute SIGR
2) Select "k" depending on the WORKLOAD
No states store this step !
3) Create Puzzle
if packets are belonging to hc, ad2, ad3, and ad4
1) Solve Puzzle -> only for hc, ad2, and ad3
2) Key Gen only for hc and ad2
3) Generate SIG_I only for hc and ad2
4) ad4 chooses messages randomly
Solve Puzzle Generate SIG_I
MSG3_I
Verify Puzzle Verify Enc{} Verify SIG_I
MSG3_R
1) Verify Puzzle -> tokens from hc, ad2, and ad3 will be accepted at this step. The ad4's tokens will be rejected and removed from the memory
2) Verify Enc{} and SIG_I -> tokens from hc, and ad2 will be accepted at this step. The ad3's tokens will be rejected and removed from the memory
Removes ad3 and ad4's token from MEMORY
1) Key Gen "K_S"
2) Verify HMAC
3) Verify Signature
4) Start secured communication
Gen Key K_s Verify HMAC and Signature
MSG4_I
Gen Key K_s HMAC Signature
MSG4_R
1) Key Gen "K_S"
2) Create HMAC
3) Generate Signature
4) Open connection for hc and ad2
5) Return resources

Fig. 9. Protocol Steps of HIP Timed CPNs Model

The Experimental Results: To adjust the puzzle difficulty, we allocate two possible values for the responder to determine. Under normal circumstances, the responder selects k=1, which means the easiest puzzle solution is required from the initiator. Once the responder receives more requested packets than its maximum capacity to handle, the responder raises the puzzle difficulty. In the experiments described here, we choose k=10. Because this puzzle technique is a hash-based puzzle, this value will help the responder to slow down the incoming rate by requiring the work of the initiator to solve puzzles at the factor of 2^{10}.

Similar to the representation of Figure 8, Figure 10 illustrates that the number of attacking machines that the responder can tolerate is increased to a higher proportion compared to the result of experiment 1. Another interesting result is that the successful rate of an honest client's message in the case of `ad4` is higher than for the fixed value k=1. The reason is that `ad4` does not compute the puzzle solution, so, no matter what the puzzle difficulty is, `ad4` can flood the bogus messages at the similar speed as experiment 1. However, at that amount of bogus messages, there are only messages from `ad4` (no legitimate traffic because `hc` has to spend some amount of time to solve the puzzle solution), or just a few messages from `hc` that arrive to the connection queue before the responder increases puzzle difficulty. As a result, the responder can validate the puzzle solution before the next group of messages has arrived. Undoubtedly, these bogus messages from `ad4` will be rejected at the first step of verification which requires only a short period and



(a) hc's load = 80% of R (b) hc's load = 150% of R

Fig. 10. Percentage of throughput from hc with k is chosen between 1 and 10

removes such attacks from the connection queue. However, this situation does not occur in the case of ad3 because they have to spend some amount of time to solve the puzzle as well as hc.

6 Conclusion and Future Work

This work has achieved the aims of extending the Meadows's cost-based framework to provide more accurate representation of computational cost and shown the potential of automated analysis. Moreover, we have explored unbalanced computational vulnerabilities on HIP which cause the responder to deplete resources and then terminate all processes by developing formal analysis based on Meadows's cost-based framework and Time CPNs simulation-based analysis. By comparing experimental results from both techniques, we have found a limitation of Meadows' framework to define the ability of advanced adversaries and address DoS vulnerabilities in DoS-resistant protocols.

In future work, we plan to extend this research by using the model checking capabilities of CPN tools to automatically verify the system by traversing the model and checking whether the cost tolerance between initiator and responder exceeds some reasonable threshold. Moreover, the power of adversaries can be extended in different ways in order to model more powerful attacks. For example, the advanced adversary, who attempts to attack the protocol at the third message, can be extended to flood reused packets from previous connections, eavesdrop messages from a valid communication, or craft bogus messages using existing messages including valid or invalid puzzle solutions as well as digital signatures. By inserting such advanced abilities to the model, we also require a technique to measure and identify cost of those operations in order to achieve a formal analysis.

In addition there are a number of other promising directions for this research.

- Our model can be used to analyse a variety of protocols to provide a comparison of the effectiveness of different protocols in DoS prevention.
- Our model can be integrated into a model for security analysis of authentication and key establishment properties to create a unified protocol analysis tool covering resistance to DoS attacks as well as more traditional security goals.

References

1. W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *the 9th ACM conference on Computer and communications security*, pages 48–58, Washington, DC, USA, 2002. ACM Press.

2. I. Al-azzoni. The Verification of Cryptographic Protocols using Coloured Petri Nets. Master of Applied Sciences Thesis, Department of Software Engineering, McMaster University, Ontario, Canada, 2004.
3. T. Aura, A. Nagarajan, and A. Gurtov. Analysis of the HIP Base Exchange Protocol. In *Proceedings of 10th Australasian Conference on Information Security and Privacy (ACISP 2005)*, volume 3574 of *Lecture Notes in Computer Science*, pages 481 – 493, Brisbane, Australia, Jun 2005. Springer-Verlag.
4. T. Aura and P. Nikander. Stateless Connections. In *International Conference on Information and Communications Security*, pages 87–97, Beijing, China, Nov 1997. Springer-Verlag.
5. T. Aura, P. Nikander, and J. Leiwo. DoS-resistant authentication with client puzzles. In *Security Protocols Workshop 2000*, pages 170–181. Cambridge, Apr 2000.
6. J. Beal and T. Shepard. Deamplification of DoS Attacks via Puzzles. Available: `http://web.mit.edu/jakebeal/www/Unpublished/puzzle.pdf`, 2004.
7. Computer Emergency Response Team (CERT). Denial-of-Service Attack via ping. [Online]. Available: `http://www.cert.org/advisories/CA-1996-26.html` [Accessed: August 2004], 1996.
8. Computer Emergency Response Team (CERT). TCP SYN Flooding and IP Spoofing Attacks. [Online]. Available: `http://www.cert.org/advisories/CA-1996-21.html`, 1996.
9. S. Christensen and K. H. Mortensen. Teaching Coloured Petri Nets- A Gentle Introduction to Formal Methods in a Distributed Systems Course. In *ICATPN*, pages 290–309, 1997.
10. Wei Dai. Crypto++ 5.2.1 Benchmarks. [Online]. Available: `http://www.eskimo.com/~weidai/benchmarks.html` [Accessed: Nov 2005], 2004.
11. E. M. Doyle. Automated Security Analysis of Cryptographic Protocols using Coloured Petri Net Specification. Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada, 1996.
12. A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0. Internet Draft, Internet Engineering Task Force, November 1996. `http://wp.netscape.com/eng/ssl3/draft302.txt`.
13. V. Gupta, S. Gupta, S. Chang, and D. Stebila. Performance Analysis of Elliptic Curve Cryptography for SSL. In *WISE'02: Proceedings of the 3rd ACM Workshop on Wireless Security*, pages 87–94, Atlanta, GA, USA, 2002. ACM Press.
14. Y. Han. Automated Security Analysis of Internet Protocols using Coloured Petri Net Specification. Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada, 1996.
15. M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS 99)*, Sep 1999. Also available as http://citeseer.nj.nec.com/238810.html.
16. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Springer-Verlag, 2nd edition, Vol. 1-3, April, 1997.
17. A. Juels and J. Brainard. Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks. In *the 1999 Network and Distributed System Security Symposium (NDSS '99)*, pages 151–165, San Diego, California, USA, Feb 1999. Internet Society Press, Reston.
18. P. Karn and W. A. Simpson. Photuris: Session-Key Management Protocol. Experimental RFC 2522, IETF, Mar 1999. `http://www.ietf.org/rfc/rfc2522.txt`.
19. C. Meadows. A Cost-Based Framework for Analysis of DoS in Networks. *Journal of Computer Security*, 9(1/2):143–164, Jan 2001.
20. H. C. Moon. A study on formal specification and analysis of cryptographic protocols using Colored Petri Nets. Master of Science Thesis, Institute of Science and Technology, Kwangju University, Korea, 1998.
21. R. Moskowitz. The Host Identity Protocol (HIP). Internet Draft, Internet Engineering Task Force, Jun 2006. `http://www.ietf.org/internet-drafts/draft-ietf-hip-base-06.txt`.
22. B. B. Neih and S. E. Tavares. Modelling and Analysis of Cryptographic Protocols using Petri Nets. In *Advances in Cryptology*, pages 275–295, Berlin, German, 1993.
23. C. A. Petri. *Kommunikation mit Automaten.* PhD Thesis, Institut fur Instumentelle Mathematik, Schriffen des IIM, 1962.
24. J. Smith, J. M. González Nieto, and C. Boyd. Modelling Denial of Service Attacks on JFK with Meadows's Cost-Based Framework. In *Fourth Australasian Information Security Workshop (AISW-NetSec 2006)*, volume 54, pages 125–134. CRPIT series, 2006.
25. J. Smith, S. Tritilanunt, C. Boyd, J. M. González Nieto, and E. Foo. DoS-Resistance in Key Exchange. *International Journal of Wireless and Mobile Computing (IJWMC)*, 1(1), Jan 2006.
26. Z. Tan, C. Lin, H. Lin, and B. Li. Optimization and Benchmark of Cryptographic Algorithms on Network Processors. *IEEE Micro*, 24(5):55–69, Sept/Oct 2004.
27. The Department of Computer Science, University of Aarhus, Denmark. CPN Tools: Computer Tool for Coloured Petri Nets. [Online]. Available: `http://wiki.daimi.au.dk/cpntools/cpntools.wiki`, 2004.

Game Coloured Petri Nets

M. Westergaard

Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark,
Email: mw@daimi.au.dk

Abstract. This paper introduces the notion of game coloured Petri nets. This allows the modeler to explicitly model what parts of the model comprise the modeled system and what parts are the environment of the modeled system. We give the formal definition of game coloured Petri nets, a means of reachability analysis of this net class, and an application of game coloured Petri nets to automatically generate easy-to-understand visualizations of the model by exploiting the knowledge that some parts of the model are not interesting from a visualization perspective (i.e. they are part of the environment, and not controllable by the system itself, or they are part of the system itself and therefore we need not worry about them).

1 Introduction

The coloured Petri nets (CPNs or CP-nets) [14] formalism has proven itself useful for modeling concurrent systems such as network protocols [11,12,17,18,21] and workflows [4, 15]. One problem that is often encountered is how to distinguish between the system itself and its environment [30], as we would often like to make the assumptions about the environment explicit even though they are not directly part of the system we want to model. Normally we would model the environment simply as a part of the model and at best accompany the model by an informal textual description of what parts of the model comprise the environment or, at worst, let this information be implicit. If the modeler is a bit more thorough, he will split the model up into separate modules and put the environment in certain modules and the modeled system in other modules. The problem with the first of approach is that the information about which parts of the model are the actual system is not part of the modeled system, thereby making it impossible to use this information during analysis or other treatments of the model. If, on the other hand, we put the model of the environment in separate modules, we may obtain an unnatural model in which the natural flow of the system is not readily visible if the flow consists of frequent interleavings of actions between the modeled system and the environment.

In this paper we will study another way to model the environment for coloured Petri nets. The idea is to separate the actions of the system into two parts, the controllable and the uncontrollable actions. The controllable actions are, as the name suggests, controllable by the system we model. The system can choose which and when to execute controllable actions, e.g. transmit a packet onto the network. The uncontrollable actions are not controllable by the system, but can occur whenever they are enabled. The inspiration of this is classical games such as tic-tac-toe, in which two players, cross and naught, play against each other. Say we model the game from the point of view of the player drawing crosses. The action of adding a new cross to the board is controllable, whereas the action of adding a naught is uncontrollable.

While it is interesting in itself to be able to study small toy-games such as tic-tac-toe using coloured Petri nets, the real power of separating the actions into uncontrollable and uncontrollable ones comes when regarding more realistic examples, e.g. a model of a network protocol. The network protocol contends against the network, which may lose packets, duplicate packets, or even alter packets. In this case all actions of the protocol (such as retransmitting a packet) are controllable, whereas the actions of the environment (such as transmitting or dropping the packet) are uncontrollable. This illustrates how separating the actions into controllable and uncontrollable actions allow us to explicitly specify what parts of the model comprise the system and what parts comprise the environment. Actions of the system are modeled as controllable transitions (by convention drawn as a rectangle) and actions of the environment are modeled as uncontrollable transitions (drawn as a dashed rectangle). Allowing this distinction directly in the model has several uses. Firstly, it alleviates the need for an informal textual description of which parts of the model are the modeled, interesting system, and which parts just make the assumptions of the environment explicit. Secondly, we can use the distinction to do better analysis of the properties of the system, and to automatically generate strategies (corresponding to programs) making sure, e.g. that the system always reaches a state where all packets have been successfully transmitted. Thirdly, we can use the extra information to generate visualizations of the system allowing users to interact with the model without looking at the model. We could have chosen to distinguish controllable actions from uncontrollable actions on the level of tokens or transition modes, but have chosen to make the distinction on the level of transitions for simplicity. Refer to the conclusion (Sect. 6) for a brief discussion of another way to do the distinction.

The contribution of this paper is three-fold: The introduction of game coloured Petri nets, the adaptation of an algorithm for reachability analysis of finite games [2] to game coloured Petri nets, and, thirdly, an application of game coloured Petri nets to automatically tie CPN models to visualizations.

The rest of this paper is structured as follows: Firstly, in Sect. 2, we will give an informal introduction to game CP-nets and introduce a simple example, which will be used in the rest of the paper. Secondly, in Sect. 3, we will formalize the notion of game coloured Petri nets (game CPNs or game CP-nets). In Sect. 4, we outline how to do reachability analysis of game CP-nets. After that we will turn to a simple yet powerful application of games to automatically generate visualizations of CPN models. Finally, we will give our conclusions and provide directions for future research in Sect. 6.

2 Example

In this section, we will introduce a simple example game CP-net, which we will use in the following sections. The example is a slight modification of one of the sample nets supplied with CPN Tools, namely a simple stop-and-wait protocol.

The modified example can be seen in Fig. 1. In the example, a sender, on the left, wants to transmit some packets lying on the place Send. The sender has a counter telling which packet to send next, NextSend. When the transition Send Packet occurs, it puts the packet onto the network (place A). Now the network, in the middle of the model, can either choose to Transmit Packet or Drop Packet. These two actions are uncontrollable, modeling that the protocol has no control over what happens on the network. If the packet is dropped, it is simply removed from place A, otherwise it is moved onto place B, where the receiver, on the right of the model, can Receive Packet. When this happens, either the packet is received for the first time or it is a retransmission. The receiver keeps track of which packet it expects by a counter on the place NextRec. If the sequence number of the incoming packet is not equal to the expected sequence number, the packet is discarded. Otherwise, the new packet is stored on the place Received. In either case, an acknowledgment is sent back to the network by putting a token on place C, containing the sequence number of the next expected packet. The acknowledgment can be either transmitted or dropped by the network. If it arrives safely on place D the sender can Receive Ack, and update the number of the next packet to send, and start a new cycle sending the next packet.

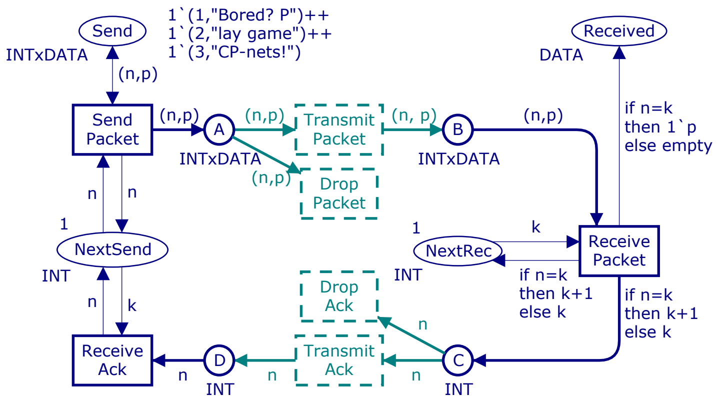


Fig. 1. A simple stop-and-wait protocol modeled using game CP-nets.

In this example we have modeled all actions of both sender and receiver as controllable and all actions of the network as uncontrollable. That is, we regard the network protocol as the modeled system and the network as the environment. We could have modeled the actions of e.g. the receiver as uncontrollable as well, thereby only regarding the sender as the modeled system.

3 Formal Definition

In this section we will give a formal definition of game coloured Petri nets (game CPNs or game CP-nets). The intuition of the definition is that we partition the set of transitions into controllable and uncontrollable transitions. We will do this by first recalling the definition of standard coloured Petri nets.

We will assume that the relations $<$, $=$, $\leq$, $>$, and $\geq$, and operations $+$ and $-$, on multi-sets are defined as usual. We use $\mathbb{N}^S$ to denote the set of all multi-sets over S.

Definition 1 (Coloured Petri net (Def. 5.1 and 5.2 in [13][1])). *A* coloured Petri net *is a tuple, $\mathcal{CPN} = (P, T, D, Type, Pre, Post, M_0)$, where*

- *P is a finite set of* places*,*
- *T is a finite set of* transitions *such that $P \cap T = \emptyset$,*
- *$D \neq \emptyset$ is a finite set of non-empty* types*,*
- *$Type : P \cup T \to D$ is a* type function *assigning a type to each place and transition,*
- *$TRANS = \{(t, m) \mid t \in T, m \in Type(t)\}$ is the set of all* transition modes*,*
- *$\mathbb{N}^{PLACE} = \mathbb{N}^{\{(p,g) \mid p \in P, g \in Type(p)\}}$ is the set of all* markings*,*
- *$Pre, Post : TRANS \to \mathbb{N}^{PLACE}$ are the* backward *and* forward incidence functions*, assigning to each arc an* annotation*, and*
- *$M_0 \in \mathbb{N}^{PLACE}$ is the* initial marking*.*

The state of a CP-net is given by a marking of the places, which is a multi-set, $M \in \mathbb{N}^{PLACE}$.

Definition 2 (Enabling and occurence of transitions (Def. 5.3.1 and 5.4 in [13])). *A transition mode, $(t, m) \in TRANS$, is* enabled *in marking $M \in \mathbb{N}^{PLACE}$ if $Pre(t, m) \leq M$. A transition $t \in T$ is enabled if there exists $m \in Type(t)$ such that (t, m) is enabled. If (t, m) is enabled in M, it may* occur *and lead to a marking M'. This is written $M \overset{(t,m)}{\to} M'$, where M' is defined by $M' = M - Pre(t, m) + Post(t, m)$.*

Definition 3 (Game coloured Petri net). *A* game coloured Petri net *(game CP-net or GCPN) is a tuple, $\mathcal{GCPN} = (P, T_c, T_u, D, Type, Pre, Post, M_0, W)$, where*

- *T_c is a finite set of* controllable transitions*,*
- *T_u is a finite set of* uncontrollable transitions *such that $T_c \cap T_u = \emptyset$,*
- *$\mathcal{CPN} = (P, T_c \cup T_u, D, Type, Pre, Post, M_0)$, the* underlying coloured Petri net*, is a coloured Petri net, and*
- *$W : \mathbb{N}^{PLACE} \to \{\mathtt{tt}, \mathtt{ff}\}$ is a predicate identifying* winning markings*.*

[1] In this paper we will use the term coloured Petri net rather than high-level Petri net as used in [13].

The notions of markings, transition modes, enabling, and occurrence for game CP-nets are the same as for coloured Petri nets. Furthermore, we shall allow the notation $TRANS_c = \{(t, m) \mid t \in T_c, m \in C(t)\}$ for the *controllable transition modes* and $TRANS_u = \{(t, m) \mid t \in T_u, m \in C(t)\}$ for the *uncontrollable transition modes*

One thing to note is that we do not impose any ordering on controllable or uncontrollable actions. We could have required that a controllable action must always be followed by an uncontrollable action and vice versa, but we will not do this, as enabling suddenly becomes dependent on the level of detail used to model the system. If such behavior is required (e.g. in the case of the tic-tac-toe game), it must be modeled explicitly. We will discuss this further in Sect. 5.4, where we talk about fairness when using game CP-nets to automatically generate visualizations of models.

4 Analysis

For finite games, i.e. games constructed by separating the actions of finite automata into controllable and uncontrollable actions, we can do several kinds of analysis. One of the simplest properties we can check is a reachability problem, namely whether it is possible to find a strategy ensuring we will end up in one of the winning states. A strategy is a mapping from states to controllable transitions. A strategy is a *winning strategy* iff we are ensured, no matter what uncontrollable transitions are executed, to end up in one of the winning states if we in every state pick the transition specified by the strategy. The reachability property is interesting when the set of the winning states, W, represent desirable final markings, e.g. that all packages have been received successfully. A winning strategy corresponds to a recipe for what the modeled system should do in order to ensure it will end up in a desirable state. Using a solution for the simple reachability problem, we can also solve the dual safety problem, whether we can ensure that no matter which uncontrollable transitions are executed, we will never reach a state which is not winning. This property is interesting for reactive systems, where the set W corresponds to states where nothing bad has happened, e.g. that the system has not dead-locked or received a damaged packet.

Rather than going through the definitions required to solve the reachability problem directly for game CP-nets, we will go through how to translate the reachability problem for game CP-nets into a reachability problem for finite state systems. This, of course, cannot be done in general (as the reachability graph of the underlying CP-net can be infinite), but we do it in a way that ensures that if the reachability graph of the underlying CP-net is finite, the algorithm will terminate with the correct answer.

In [5], Cassez, David, Fleury, and Larsen instantiate an algorithm by Liu and Smolka from [19] to obtain an efficient (and optimal) algorithm to decide whether a given finite reachability game has a winning strategy and to extract that strategy. The intuition of the algorithm is to calculate a minimal fix-point of all good states, Win, where all states in W are good and all states where we can take a controllable step to a good state and all uncontrollable steps lead to a good state are good. This corresponds to the intuition that in any given state, we have a winning strategy if we can execute a controllable transition and end in a new state where we have a winning strategy, and that no matter what the opponent does, we end up in a state where we have a winning strategy. The algorithm assumes a finite game as a tuple $(Q, q_0, Act_c, Act_u, \delta, Goal)$ (all states, the initial state, the controllable and uncontrollable actions, the transition relation, and the winning states). The adaptation of the algorithm to game CP-nets is easy. Given a game CP-net, $\mathcal{GCPN} = (P, T_c, T_u, D, Type, Pre, Post, M_0, W)$, we will assume that all types in D are finite and that the number of tokens on all places are bounded. Then we can take $Q = \mathbb{N}^{PLACE}$, $Act_c = TRANS_c$, and $Act_u = TRANS_u$, all of which are finite. We set $Goal = W$, $q_0 = M_0$, and let $\delta = \{(M, (t, m), M') \mid M \overset{(t,m)M'}{\rightarrow}\}$. We then obtain Algorithm 1. The algorithm works by forward traversal of the reachability graph. Whenever a state is found to be winning, it is added to Win (ll. 3, 10, and 19). Whenever we find a new state, we mark it as dependent on the winning status of its successors (ll. 9 and 22). When a state is marked as winning, we schedule all states dependent on its winning status for re-evaluation (ll. 13 and 18). With a little cleverness in the implementation of line 16[2], the algorithm is shown to find a winning strategy in time linear in the number of nodes and edges in the reachability graph. We use the standard notation that the empty conjunction is tt and the empty disjunction is ff.

A nice property of this algorithm is that the while loop (ll. 5–25) has the invariant that if $\mathsf{Win}[M] = \mathsf{tt}$ then there exist a winning strategy for $\mathcal{GCPN}$ where we use M as the initial marking (rather than M_0). Furthermore this invariant holds irregardless of which element the operation *pop* picks in line 6. The first property allows us to implement early termination by adding the additional constraint $Win[M_0] \neq \mathsf{tt}$ to the while loop in line 5, and the second property allows us to do more intelligent search for winning strategies. The algorithm has been implemented in the model checker included in the BRITNeY Suite [28, 29]. The current implementation uses either a simple depth-first search or a breath-first search. The depth-first search has a tendency to find winning strategies faster than the breath-first search, as it favors making controllable moves rather than waiting for the opponent to make “good” uncontrollable moves. More intelligent search can be made by making e.g. an $\alpha - \beta$ search [25] as known from the AI world to allow computers to play difficult games, such as chess.

[2] Rather than re-evaluating both the conjunction and the disjunction each time we reach this line, we notice that the entire expression is true iff just one state reachable by a controllable action is winning and if all states reachable by uncontrollable actions are winning. Using an integer to keep track of how many states reachable by an uncontrollable action are currently not marked as winning and a boolean to keep track of whether a state reachable by a controllable action has been marked as winning, we only do a constant amount of work each time we need to re-calculation the expression.

Algorithm 1 SolveGCPN

```
Require: $\mathcal{GCPN} = (P, T_c, T_u, D, Type, Pre, Post, M_0, W)$, a game CP-net
 1: Storage $\leftarrow \{M_0\}$
 2: Waiting $\leftarrow \{(M_0, (t,m), M') \mid M_0 \overset{(t,m)}{\rightarrow} M'\}$
 3: Win$[M_0] \leftarrow W(M_0)$
 4: Depend$[M_0] \leftarrow \emptyset$
 5: while Waiting $\neq \emptyset$ do
 6:   $(M, (t,m), M') \leftarrow pop($Waiting$)$
 7:   if $M' \notin$ Storage then
 8:     Storage $\leftarrow$ Storage $\cup \{M'\}$
 9:     Depend$[M'] \leftarrow \{(M, (t,m), M')\}$
10:     Win$[M'] \leftarrow W(M')$
11:     Waiting $\leftarrow$ Waiting $\cup \{(M', (t',m'), M'') \mid M' \overset{(t',m')}{\rightarrow} M''\}$
12:     if Win$[M']$ then
13:       Waiting $\leftarrow$ Waiting $\cup \{(M, (t,m), M')\}$
14:     end if
15:   else {reevaluate}
16:     Win$^* \leftarrow \bigwedge_{(t_u,m_u) \in TRANS_u,\, M^{(t_u,m_u)} \rightarrow M'}$ Win$[M'] \wedge$
          $\bigvee_{(t_c,m_c) \in TRANS_c,\, M^{(t_c,m_c)} \rightarrow M''}$ Win$[M'']$
17:     if Win$^*$ then
18:       Waiting $\leftarrow$ Waiting $\cup$ Depend$[M]$
19:       Win$[M] \leftarrow \mathtt{tt}$
20:     end if
21:     if Win$[M']$ then
22:       Depend$[M'] \leftarrow$ Depend$[M'] \cup \{(M, (t,m), M')\}$
23:     end if
24:   end if
25: end while
```

Due to the general result of [19], we can replace the calculation in line 16 of the algorithm with any monotone property (any propositional formula not using negation or implication). For example, we may find that it is more intuitive that a state is winning if we have a controllable move leading to a winning state *or* all uncontrollable moves lead to winning states.

We also notice that we never use that $\mathbb{N}^{PLACE}$ (the syntactically possible states of $\mathcal{GCPN}$) is finite, but only that the reachable states of $\mathcal{GCPN}$ are, so we can in fact remove the requirement that all types in D are finite as long as we only use a finite subset of the values. This allows us to analyze arbitrary game CPN models using the algorithm. If the algorithm terminates, we will know that the result is correct (due to the loop invariant, which is proven correct in [5]).

4.1 Experimental Results

We cannot directly analyze the network protocol from the previous example, as the place A is unbounded (and the algorithm will keep executing the Send

Packet transition, which remains enabled). However, if we put a bound on the number of tokens on the places A, B, C, and D, we can analyze the model. We will then discover that there is no winning strategy (the network can just keep on throwing away packets). If we furthermore limit the number of times the network can drop packets, we will find that we have a winning strategy. When we add a limit on the number of packets on the network and limit the number of times a packet can be lost, we obtain the model in Fig. 2. The only changes from the original net in Fig. 1 is that we have added two new places Capacity (bounding the number of tokens on A, B, C, and D) and May lose (bounding the number of times Drop Packet and Drop Ack can fire). Also the place NextRec has been moved to improve the layout.

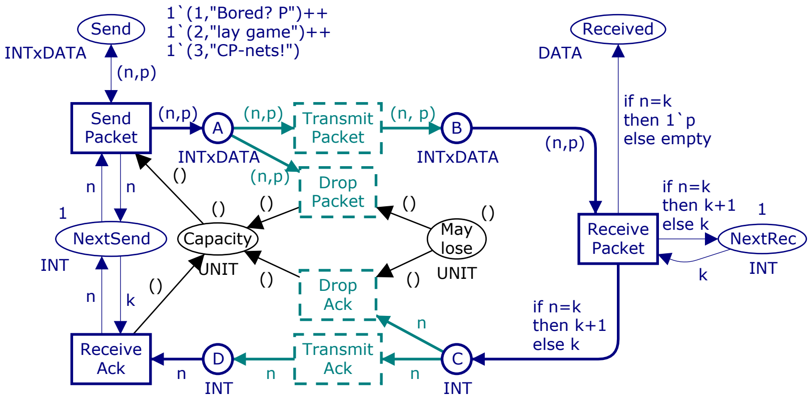


Fig. 2. The stop-and-wait protocol modified for analysis.

We have conducted four series of experiments: two for the model in Fig. 2 and two for the same model but with the May lose place and all connected arcs removed (i.e. removing the limit on the number of packet losses). For each of the two models, we have conducted two series of experiments, one without early termination and one with early termination, where we allow the algorithm to terminate as soon as we have a proof either for or against the existence of a winning strategy. Each series consists of a number of experiments with varying capacity and bound of the number of packets we may lose. We regard as winning states the states where all packets has been received successfully. All series are conducted using a recursive depth-first traversal and using double hashing [8] with 15 combinations and 10^8 buckets for the Win and Storage data-structures (resulting in a total memory use of about 25 MiB). Depend is represented implicitly on the recursion stack. The experiments are not meant to show the general performance of the algorithm, as this has already been done in [5], but they are meant to give an impression of the performance of the algorithm when adapted to game CP-nets. All experiments are conducted on an Apple MacBook Pro with a 2.16 GHz Intel Core Duo processor[3] and 2 GiB RAM, using BRITNeY Suite version 0.9.75.101.

In Table 1, we see the results from analysis of the model with no limit on the number of losses of packets. The table has 6 columns. The first column, Early termination allowed, indicates whether early termination is allowed or not. The second column, Tokens on Capacity place, indicates how many tokens the place Capacity contains initially. The columns Storage and Win indicates how many states are stored in the corresponding data-structures. Note that we do not store states with no successors in Storage (as they are winning iff they are contained in Win), which is why the number of winning states may be larger than the number of stored states. The next column, Winning strategy exists indicates whether the algorithm concludes that a winning strategy exists. The final column, Execution time, indicates how long time (in seconds) the algorithm used to reach that conclusion.

The first thing we notice is that early termination really speeds the calculation up. It seems that the time spent increases exponentially without and quadratically with early termination. This is a lucky coincidence in this model, as the algorithm can easily see that no matter how many packets we pump onto the network, the network can just drop them and leave us at the initial state, thereby preventing us from even getting started with the protocol. We also note that it seems like the algorithm has a start-up cost of about 0.1 second, which is probably used for allocating the bit-arrays used for storing states.

In Table 2 we see the results for the analysis with a limit on the number of packets the network is allowed to lose. The columns have the same meaning as in Table 1, except we have added a column, Tokens on May lose place, which contains the number of tokens initially on the May lose place.

We notice that the algorithm now states that we do indeed have a winning strategy (and gives us a function which returns for each reachable state which event we should execute in order to win). Furthermore, we notice that when we do not allow early termination, the number of states and the execution time grows approximately linearly in the number of allowed losses (which makes sense, as we basically have a copy of the entire reachability graph for each possible marking of May lose). We also note that the number of states and the execution time when no packets can be lost correspond approximately to the number of states when we allow an arbitrary number of packet losses (this also makes sense, as a packet loss basically goes back an event).

When we allow packet loss, we see, again, that the number of states and the execution time grows linearly with the number of packets the network can lose, and they seem to grow quadratically in the capacity of the network. The calculation is significantly more difficult than before, when there was no limit on the number of packets we can lose, as we will have to search deeper for a winning strategy than we did to find a counterexample previously. We also note a significant speedup compared to not allowing early termination.

[3] The processor contains two cores, but the analysis is currently only able to use one of them.

Table 1. Data from analysis of the network model in Fig. 2 without limit on number of losses (i.e. the May lose place and its surrounding arcs have been removed).

Early termination allowed	Tokens on Capacity place	Storage	Win	Winning strategy exists	Execution time (seconds)
No	1	20	2	ff	0.153
	2	115	32	ff	0.171
	4	1807	1032	ff	0.588
	6	14917	11508	ff	3.208
	8	83173	76208	ff	22.046
	10	355842	365770	ff	115.459
Yes	1	3	0	ff	0.111
	2	4	0	ff	0.116
	10	12	0	ff	0.145
	100	102	0	ff	0.282
	500	502	0	ff	3.392
	1000	1002	0	ff	12.338
	2000	2002	0	ff	55.809
	2800	2802	0	ff	123.180

When we have found a winning strategy (or concluded that no winning strategy exists), we would like to show this strategy to the user. Ways to do this include printing all reachable markings and the winning move or to annotate a graphical representation of the reachability graph with the suggested move. This is very easy to implement, but not very good to convince users. A better way to present a winning strategy to a user is presented in the next section.

5 Automatically Generated Visualizations

While CP-nets are graphical and can be communicated to computer scientists, they contain a lot of details that are not needed to grasp the essentials of a model. Often we also need to communicate the ideas of the model to domain experts in order to validate that the constructed model actually represents the problem domain. In order to facilitate easy communication of (CPN) models, a lot of tools [10, 16, 24, 29] have been conceived with the purpose of constructing domain specific visualizations. These tools either mainly allows simple inspection of the state of the model during execution, or requires that the modeler spends a lot of time constructing an visualization and tying it to the model.

Using the concept of games, we can do better. Firstly, we notice that visualizations often allow the user to experiment with the modeled system and observe how the system reacts to different stimuli, thereby playing the role of the environment. Another kind of visualization allows the user to play the role of the modeled system, contending against the environment. Both of these correspond nicely to how we play a game. We make a move and observe how the opponent

Table 2. Data from analysis of the network model in Fig. 2.

Early termination allowed	Tokens on Capacity place	Tokens on May lose place	Storage	Win	Winning Winning exists	Execution Time (seconds)
No	4	0	1732	2248	tt	0.340
	4	1	3538	4570	tt	0.602
	4	5	10762	13858	tt	1.679
	4	50	92032	118348	tt	18.752
	4	200	362930	466638	tt	140.891
	8	0	82768	120871	tt	16.561
	8	1	165939	242146	tt	39.675
	8	5	498612	727198	tt	131.478
Yes	4	200	10903	11688	tt	1.847
	8	5	290	297	tt	0.147
	10	0	50	51	tt	0.144
	10	1	185	192	tt	0.147
	10	10	737	757	tt	0.232
	10	100	12134	13043	tt	1.594
	10	1000	126699	136647	tt	71.228
	10	2000	253778	273733	tt	260.406
	100	0	410	411	tt	0.335
	100	1	825	827	tt	0.561
	100	10	4608	4625	tt	1.638
	100	100	43630	43946	tt	14.112
	100	500	438319	467416	tt	247.340
	500	73	149511	149636	tt	246.562
	1000	17	72265	72283	tt	247.027
	1500	3	24055	24059	tt	294.097

reacts. So, basically, what we want to is to allow an external visualization component to control one side of the game and let the tool control the other side. In our example, we can let the user control the controllable actions, thereby taking the role of the protocol, or let the user play the role of the environment, thereby letting the user try to make the protocol fail. As the distinction between the modeled system and the environment is part of the model in game CP-nets, it is possible to automatically execute transitions from the environment when the user plays the role of the modeled system (and vice versa).

The rest of this section starts by introducing two simple kinds of automatically generated visualization, then goes on to discuss fairness during the visualization, and finally extends the scope to more general visualizations. All of the visualizations presented in this section have been implemented in the BRITNeY Suite [28, 29].

5.1 Binding Index Inspired Visualization

CPN Tools allows users to select specific modes for each transition by using binding indexes (in CPN Tools the notion of a high-level Petri net Graph [13, Sects. 7–9] is used, where binding elements, a transition and an assignment of values to variables on the surrounding arcs, correspond to transition modes). In Fig. 3 we see an example of a binding index in CPN Tools. Here the transition Transmit Packet is enabled, and the variable n can be assigned either the value 1 or 2, and the variable p can be assigned either "Bored? P" or "lay game". Binding indexes make it possible to exercise very detailed control of the model, but also requires the user to browse through the entire model to find enabled transitions and to select the desired mode.

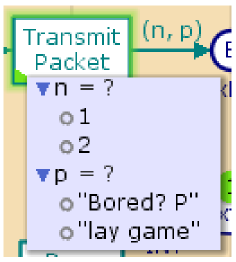


Fig. 3. A binding index from CPN Tools.

One way to automatically generate a "visualization" is to simply show the user a list of all enabled binding elements (i.e. hide any binding elements corresponding to uncontrollable transitions and execute them automatically). An example of this can be seen in Fig. 4. Here we see that the transitions Receive Packet and Send Packet are enabled. We also see that Receive Ack is disabled (normally the list of disabled transitions are hidden, but they can be made visible by the user by clicking on the double arrow). We have currently selected the Receive Packet transition, and we see that we can select one of two binding elements. We note that we do not see any of the transitions for transmitting and dropping packets, as they are part of the environment and cannot be controlled by the user. At the top, we can see that the last move of the tool was to execute the transition Drop Packet and the values assigned to its surrounding variables. This visualization is very straightforward to implement and use, but requires that the user has a very detailed knowledge of the model (or that the model is very simple).

A better way to automatically generate visualizations is to take the idea of the binding index a bit further. We will show a dialog for each enabled transition. The dialog can be placed by the user, allowing him to arrange the constructed work space at will, but will be opened/closed by the system depending on whether the transition is enabled. Instead of just listing the possible assignments, the dialog contains well-known widgets for displaying booleans, integers,

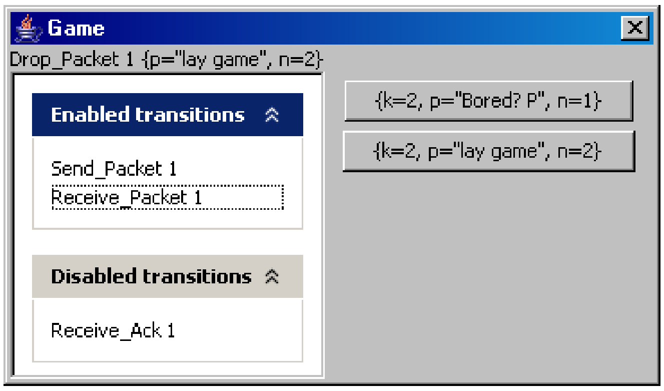


Fig. 4. A visualization generated by showing enabled binding elements.

and strings. More complex types can currently be selected by selecting the values from a list. An example of this can be seen in Fig. 5. Here three transitions are enabled, Send Packet, Receive Packet, and Receive Ack. We can see that we are currently executing Send Packet as it is grayed out and a rotating progress indicator indicates that the transition is being executed. In the Receive Packet window we see that K is assigned the value 3, N to 2, and P to "Rocks". The small triangle with an exclamation mark near the input field of P indicates that the value selected is indeed valid, but it is inconsistent with the other values (the packet containing "Rocks" has a sequence number of 3). We cannot change the value of K—it is shown for informational purposes only (this feature can be used to display a message to the user, e.g. by adding a variable message and assigning it one of the values "Please enter your name" or "The packet has been transmitted", depending on what action is required of the user and the model). In the window for Receive Ack we see that K is assigned to 2 and N to 0. The circle with a cross indicates that the value is not valid, and another value should be chosen.

Apart from being a natural way to select between available tokens, this visualization also allows the user to generate new tokens by adding a controllable source transition, which has free variables. This is also how we would normally generate new tokens in CPN Tools, but CPN Tools only allows users to bind variables freely from "small colour sets", which basically means booleans and enumerations. BRITNeY Suite extends this to also include strings and integers, which e.g. makes it possible to ask a user for name and age and use that later in the model.

This visualization can also be used without using the notions of games, as we only use the separation of transitions to make some choices on behalf of the user. In the next section we shall see an example of a visualization which crucially depends on the separation in order to work by using controllable actions to provide stimuli to the model and uncontrollable actions to provide feedback to the user.

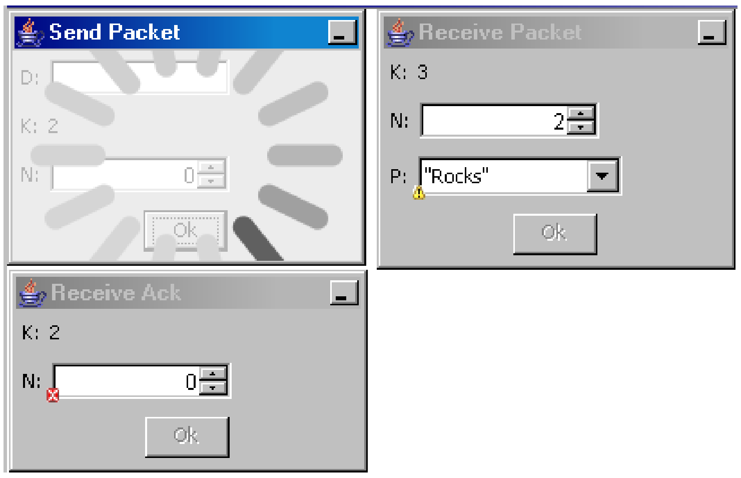


Fig. 5. Visualization generated from the net using common widgets.

5.2 Visualization using SceneBeans

BRITNeY Suite integrates the SceneBeans [26] library for use with visualizations. The library uses an XML specification for describing visualizations and allows the model to interact with the specified visualization by invoking commands in the visualization and receiving events from the visualization.

Normally the programmer would use the interface from Listing 1 and call the functions from declarations or from code segments attached to the transitions of the model. The first two functions in Listing 1 are used to setup or reset the visualization, invokeCommand (l. 5) is used to invoke commands in the visualization, which basically means to show some animation or feedback to the user. The next four functions, in lines 7–10, makes it possible to listen for events from the visualization (e.g. the user has clicked a certain object, dragged a certain object, or some other action). The last two function makes it possible to set certain values in the visualization, such as the position or color of an element. The use of this interface has a tendency to clutter the model and special precautions must be taken in order to turn the visualization on or off, e.g. for analysis.

Rather than letting the user tie the model and the visualization together manually, we shall identify events of the SceneBeans visualization with controllable actions and commands with uncontrollable actions. Thus, whenever an event is dispatched, the system will try to execute the corresponding transition as soon as possible. If an uncontrollable transition is fired, the corresponding command (if one exists) will be called. We shall identify events and commands with transitions simply by name. A more complex mapping can be made (e.g. by introducing a “synchronize” predicate, which determines whether a transition should be synchronized with a command or event), but we prefer this simple approach, as it is simpler to understand for the user, and eliminates the need to manually specify such a predicate.

Listing 1 The interface to the SceneBeans library.

```
signature SCENEBEANS = sig
  val setAnimation: unit -> string
  val reset: unit -> unit

  val invokeCommand: string -> unit

  val hasMoreEvents: unit -> bool
  val getNextEvent: unit -> string
  val peekNextEvent: unit -> string
  val waitForEvent: string -> unit

  val setValue: string * string * string -> unit
  val getValue: string * string -> string
end
```

Suppose we have created an visualization showing two computers and a network. The visualization has commands for moving dots from one computer to the other and vice versa as well as for dropping these dots from the cloud. If we name these commands after the transitions, we can, without altering the net in Fig. 1 at all, generate a SceneBeans visualization like the one in Fig. 6. Here we have a sender (on the left) and a receiver (on the right) and a network connecting them. The dots represent data or acknowledgment packets sent over the network. When a transition fires in the model, the visualization is automatically updated. When we click on the sender, an event is generated, which causes the sender to send a packet, so initially the model does nothing (it waits for packets to appear on the network). We can send as many packets as we want as fast as we want. The model will transmit or drop the packets automatically. We could also have added a way to manually receive packets, but we have chosen that the model can do this itself.

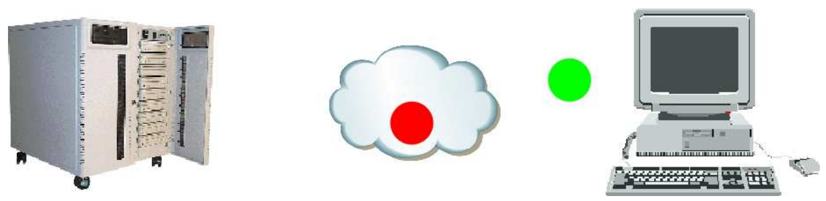


Fig. 6. SceneBeans visualization automatically synchronized with the model.

We note that we only have concepts corresponding to the command and event parts of the SceneBeans interface. It would be very nice to extend the automatic synchronization of SceneBeans visualizations and game CPN models to also include reading from and writing to properties of the visualization, for example to show the NextSend and NextRec counters directly in the visualization, or to attach package numbers to the dots. This could probably be done by associating values in the visualization with tokens on global fusion places in the game CPN model. How to do this is future work.

5.3 Intelligent Playing

Using the algorithm outlined in Sect. 4, we can allow the computer player to play more intelligently by using the winning strategy (if one exists) as guidance rather than random selection.

In principle, this requires that we calculate the entire winning strategy beforehand. As explained in Sect. 4, we know that we can use $\alpha - \beta$ heuristics to speed up the search, and we can use early termination of the analysis algorithm to speed up the initial start up time. If a winning strategy is found during a previous game, it can of course be reused during future visualizations.

Allowing the computer player to use a winning strategy is a nice way to present a winning strategy to the user, as it may be difficult to grasp the winning strategy as a mapping from states to transition modes. By allowing the user to play against the strategy, he can easily convince himself that the strategy proposed by the algorithm is indeed winning. All visualizations introduced in this section can be used to demonstrate that a winning strategy is winning.

5.4 Fairness in Execution

When we make visualizations using games, we may need to impose some fairness during the execution, as the model can be executed fast enough to make it difficult for the user to interact with the model. This can be done in different ways.

The simplest way to impose some kind of fairness is to make the game strictly turn-based: the model makes one step, followed by the user, repeat until no more transitions are enabled. If a transition is not enabled for the player, who is to make the next move, the turn is passed on to the other player. This has the advantage of being simple, easy to understand, and it makes implementing simple games, such as tic-tac-toe very easy. The disadvantage is that depending on the modeling detail level, one player may gain an unfair advantage.

Another way to impose fairness is to allow the model to execute some transitions at will, but force it to synchronize on others. This is in particular useful for SceneBeans visualizations, where the model is often expected to be simulated while the user is observing, but we want interaction to happen immediately when the user requests it. In the network example this corresponds to how the network is able to transmit or drop packets at will, with the user observing the transmissions, but when the user clicks on the sender, the model will immediately send a new packet onto the network.

A third way to impose fairness is to make the execution of transitions take time. The most obvious way to use this is to identify model time with real time and use a coloured variant of Time Petri nets [20]. In this case transitions may only be enabled for a certain period of time or only a certain period of time after another action.

5.5 More General Visualizations

Even though one goal of this work is to automatically generate visualizations, we have also made available a very generic interface, which makes it possible for developers to build visualizations or other programs interfacing with game CPN models using a high-level interface.

The interface is written in Java and gives developers the ability to write their own programs interfacing with the model. The interface, which can be seen in Listing 2, informs a consumer, i.e. any class implementing the interface, whenever a transition becomes enabled or disabled along with the possible bindings of the variables of the transition (ll. 2–3). In addition the consumer is informed whenever the computer makes a move (the uncontrollable moves, l. 8) and allows the consumer to specify a controllable move (l. 5). Furthermore, to make it easy to write programs, which only listens to what happens (and are not interested in interfering with the execution), consumers are also informed when a controllable move is made (l. 7) and when the game has ended (l. 10).

Apart for all of this transition-based information, the consumer has access to the markings of all places in the model as well as type information about all places and variables in the net.

Listing 2 The GameListener interface.

```
1  public interface GameListener {
2    public void transitionEnabled(Instance<Transition> ti, Bindings bindings);
3    public void transitionDisabled(Instance<Transition> ti);
4
5    public Pair<Instance<Transition>, Binding> controllableMove();
6
7    public void controllableMove(Instance<Transition> ti, Binding binding);
8    public void uncontrollableMove(Instance<Transition> ti, Binding binding);
9
10   public void gameOver();
11 }
```

All of the visualizations presented in this section have been implemented using this interface, so even though the interface is simple, it is still very versatile, and has, in addition, been used to automatically generate message sequence charts from game CPN models as well as basis for implementing a workflow system on top of game CP-nets.

6 Conclusion

In this paper, we have defined the notion of game coloured Petri nets. We have argued that they provide a nice way to separate the model of the system from the model of the environment. We have introduced a way of finding winning strategies for game CP-nets by means of reachability analysis and introduced an intriguing application of game CP-nets to visualization of CP-net models.

The distinction between controllable and uncontrollable actions for Petri nets has also been done in [7], but only workflow nets are considered and the paper requires that uncontrollable transitions can only exist in free choices consisting of uncontrollable transitions only. This makes it impossible for uncontrollable actions to depend of different conditions and for controllable actions to co-exist with uncontrollable transitions. Game coloured Petri nets are more widely applicable, as they extend the idea to coloured Petri nets and impose no restrictions on the modeling style, yet still offer powerful analysis techniques.

Future work includes extending the notion of partial observability [3] to game CP-nets. One way to do this, would be to only allow the player to observe certain places. This can be done by only allowing the player to observe global fusion places.

We could also use the notion of symmetry [6] as another way to specify the different players. In our implementation controllability/uncontrollability is tied to transitions, which makes it difficult to implement systems with a number of peers, say a peer to peer network with 3 nodes, where one of the peers is controllable and the other are not. With our implementation, we would need to model "peer 1" separately from "peer 2" and "peer 3", even though they can do the same actions. It would be nice to be able to specify in the declaration of the type representing the network nodes that actions involving "peer 1" are controllable whereas actions involving the other peers are not. This seems trivial to do, but we have to overcome the problem of what happens when one action involves both controllable and uncontrollable parts (say "peer 1" sends data to "peer 2"). It would be natural to look into how symmetries are specified in [9] as annotations on the types.

Another problem we saw during the analysis of the protocol in Sect. 4 was that we did not have a winning strategy if we allowed the network to drop as many packets as it wants to. We saw that for all examples with a bound on the number of packets which could be lost, we actually had a winning strategy. However, using the current method, we do not have a means to prove that we have a winning strategy which either leads us to the desired goal state or which drops infinitely many packets. This corresponds to the difference in checking simple reachability properties (such as safety properties, like deadlock-freeness) versus checking more complex properties (e.g. demand-response) using modal logics such as linear temporal logic (LTL) [23, 27] or computation tree logic (CTL) [1]. In [2,22] the notion of extended goals for games is introduced. It would be nice to come up with an algorithm for deciding extended goals, formulated using either LTL or CTL.

Acknowledgments The author wishes to thank the members of the CPN Group at the University of Aarhus, in particular Kristian Bisgaard Lassen, for discussions on some of the topics described in this paper.

References

1. M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.
2. P. Bertoli, A. Cimatti, M. Pistore, and P. Traverso. A Framework for Planning with Extended Goals under Partial Observability. In *Proc. of ICAPS 2003*, pages 215–225, 2003.
3. P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In *Proc. of IJACI 2001*. AAAI Press, 2001.
4. C. Bossen and J.B. Jørgensen. Context-descriptive prototypes and their application to medicine administration. In *DIS '04: Proc. of the 2004 conference on Designing interactive systems*, pages 297–306, Boston, MA, USA, 2004. ACM Press.
5. F. Cassez, A. David, F. Emmanuel, K.G. Larsen, and D. Lime. Efficient On-the-fly Algorithms for the Analysis of Timed Games. In *Proc. of CONCUR 2005*, volume 3653 of *LNCS*, pages 66–80. Springer-Verlag, 2005.
6. E.M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Logic Model Checking. In *Proc. of CAV'93*, LNCS, pages 450–462. Springer-Verlag, 1993.
7. J. Dehnert. Non-controllable Choice Robustness Expressing the Controllability of Workflow Processes. In *Proc. of ICATPN 2002*, volume 2360 of *LNCS*, pages 121–141. Springer-Verlag, 2002.
8. P.C. Dillinger and P. Manolios. Fast and accurate Bitstate Verification for SPIN. In *Proc. of SPIN 2004*, volume 2989 of *LNCS*. Springer-Verlag, 2004.
9. L. Elgaard. *The Symmetry Method for Coloured Petri Nets.* PhD thesis, Department of Computer Science, University of Aarhus, 2002. Also available as DAIMI PB-564.
10. G. Gallasch and L.M. Kristensen. A Communication Infrastructure for External Communication with Design/CPN. In *Proc. of 3rd Workshop on Practical Use of Coloured Petri Nets and the CPN Tools*, volume PB-554 of *DAIMI*, pages 79–93. Department of Computer Science, University of Aarhus, 2001.
11. S. Gordon, L.M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol. In *Proc. of ICATPN'02*, volume 2360 of *LNCS*, pages 182–202. Springer-Verlag, 2002.
12. B. Han and J. Billington. Formalising the TCP Symmetrical Connection Management Service. In *Proc. of Design, Analysis, and Simulation of Distributed Syste ms*, pages 178–184. SCS, 2003.
13. Software and system engineering – High-level Petri nets – Part 1: Concepts, definitions and graphical notation. ISO/IEC 15909-1:2004, 2004.
14. K. Jensen. *Coloured Petri Nets—Basic Concepts, Analysis Methods and Practical Use. Volume 1: Basic Concepts*. Springer-Verlag, 1992.
15. J.B. Jørgensen and K.B. Lassen. Aligning Work Processes and the Adviser Portal Bank System. In *REBNITA05*, 2005.
16. E. Kindler and C. Páles. 3D-Visualization of Petri Net Models: Concept and Realization. In *Proc. of ICATPN 2004*, volume 3099 of *LNCS*, pages 464–473. Springer-Verlag, 2003.

17. L.M. Kristensen and K. Jensen. Specification and Validation of an Edge Router Discovery Protocol for Mobile Ad-hoc Networks. In *Integration of Software Specification Techniques for Application in Engineering*, volume 3147 of *LNCS*, pages 248–269. Springer-Verlag, 2004.
18. L.M. Kristensen, M. Westergaard, and P.C. Nørgaard. Model-based Prototyping of an Interoperability Protocol for Mobile Ad-hoc Networks. In *Proc. of Fifth International Conference on Integrated Formal Methods*, volume 3771 of *LNCS*, pages 266–286. Springer-Verlag, 2005.
19. X. Liu and S.A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points (Extended Abstract). In *Proc. of ICALP 1998*, volume 1443 of *LNCS*, pages 53–64. Springer-Verlag, 1998.
20. P. Merlin and D. J. Farber. Recoverability of communication protocols - implication of a theoretical study. *IEEE Trans. on Communications*, 24(2):1036–1043, 1976.
21. C. Ouyang and J. Billington. On Verifying the Internet Open Trading Protocol. In *Proc. of 4th International Conference on Electronic Commerce and Web Technologies*, volume 2738 of *LNCS*, pages 292–302. Springer-Verlag, 2003.
22. M. Pistore and P. Traverso. Planning as Model Checking for Extended Goals in Non-Deterministic Domains. In *Proc. of IJCAI 2001*, pages 479–486. AAAI Press, 2001.
23. A. Pnueli. The temporal logic of programs. In *Proc. of 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.
24. J.L. Rasmussen and M. Singh. *Mimic/CPN. A Graphical Simulation Utility for Design/CPN. User's Manual.* `www.daimi.au.dk/designCPN`.
25. Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition).* Prentice Hall, 2002.
26. SceneBeans. Online `www-dse.doc.ic.ac.uk/Software/SceneBeans`.
27. M. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *In proc. of IEEE Symposium on Logic in Computer Science*, pages 322–331, 1986.
28. M. Westergaard. BRITNeY Suite website. `wiki.daimi.au.dk/britney/`.
29. M. Westergaard and K.B. Lassen. The BRITNeY Suite Animation Tool. In *Proc. of ICATPN 2006*, volume 4024 of *LNCS*, pages 431–440. Springer-Verlag, 2006.
30. P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.