A Coloured Petri Net based Tool for Course of Action Development and Analysis

Lin Zhang* **Lars M. Kristensen**** **Chris Janczura*** **Guy Gallasch**** **Jonathan Billington****

* Systems Simulation and Assessment Group
Defence Science and Technology Organisation
Edinburgh, SA 5111, Australia
{Lin.Zhang,Chris.Janczura}@dsto.defence.gov.au

** Computer Systems Engineering Centre
School of Electrical and Information Engineering, University of South Australia
Mawson Lakes, SA 5095, Australia
{Lars.Kristensen,Guy.Gallasch,Jonathan.Billington}@unisa.edu.au

Abstract

This paper presents a Course Of Action Scheduling Tool (COAST) to support operations planning in the Australian Defence Force (ADF). COAST is a software tool based on Coloured Petri Nets (CPNs or CP-nets) and Design/CPN. It is developed with an underlying conceptual framework that complements the ADF planning process. At the core of COAST is a CPN model that formally specifies the execution of tasks in a course of action. During planning, the CPN model is instantiated with concrete tasks for execution and analysis. COAST has a client-server architecture. The client provides a graphical user interface for task instantiation and lines of operation analysis. The server uses the instantiated CPN model to conduct state space analysis for generating and analysing lines of operation.

Keywords: command and control, military operations planning, computer tool support, courses of action, Coloured Petri Nets.

1 Introduction

Military operations planning is one of the primary functions performed by a military Headquarters (HQ). For many staff, planning presents a major challenge due to several factors including time pressure, ambiguity in guidance, uncertainty, complex situations, and a requirement for seamless collaboration both within and externally to their organisations. The Australian Defence Force (ADF) follows a systematic planning process to manage these factors. The process stipulated in the ADF doctrine (ADF 1999) includes four consecutive and iterative steps: mission analysis, course of action (COA) development, course of action analysis, and decision and execution. In mission analysis, a military commander and the staff attempt to analyse the intent of their superior commander in the context of their environment to arrive at an unambiguous understanding of their mission. A mission comprises a list of essential *tasks* that must be carried out to fulfil the mission. During COA development, the commander and staff consider possible options for the achievement of the mission, and these options are called *courses of action*. Each COA is then developed to include a number of action tasks that need to be scheduled. A small number of COAs are analysed for improvement through simulation and wargaming during COA analysis. Finally, the commander decides on one COA that is to be developed into a plan for execution.

A number of research projects have been carried out within the Australian Defence Science and Technology Organisation (DSTO) to support military planning. One of the projects was devoted to modelling the ADF planning process for the analysis of its efficiency when applied to a particular military Headquarters (Kristensen et al. 2002). To assist planning staff in mission analysis and early stages of COA development, a decision support tool based on Bayesian belief networks (Jensen 1996) was developed (Falzon et al. 2001). This tool represents elements of military forces for the determination and analysis of critical elements that need to be exploited in the case of a threat force or protected in the case of a friendly force. The methods of affecting the identified critical elements form the basis of COAs, and the tasks designed to implement the methods become building blocks of the COAs. However, a COA is considered incomplete until the tasks are logically sequenced, and scheduled in the constraint of assigned resources. The orderings of tasks are called *lines of operation*.

The purpose of the COA Scheduling Tool (COAST) is to support the sequencing and scheduling of tasks to form lines of operation that are an implementation of *suitable* and *feasible* COAs. A suitable COA is one that, when executed, can achieve the commander's mission. A COA is feasible if the sequence of tasks in the COA can be executed with the assigned forces and other resources. The lines of operation developed with the assistance of COAST form the basis for further quantitative analysis using metrics such as probabilities of success and loss of resources to compare and evaluate possible lines of operation.

The execution, synchronisation, and timing of tasks in a COA and their use of resources in a COA can be viewed as a concurrent system. This means that the execution of

tasks in a COA is naturally represented using modelling languages aimed at concurrent systems such as Coloured Petri Nets (Jensen 1992).

The contribution of this paper is to present COAST and show how CPNs and the supporting Design/CPN tool (CPN Group 1996) have been used to engineer a tool theoretically founded on formal methods, but where the use of formal methods is transparent to its user. The transparent use of CPNs is essential, as we wish planners to use COAST without any knowledge of CPNs and their analysis. The work presented in this paper and in (Falzon et al. 2001) is part of an integrated modelling environment (In-MODE) developed within DSTO for military planning.

The idea of extracting CPN models from Design/CPN in executable form to use them in the implementation of a system was also used in (Mortensen 2000) to obtain the implementation of an access control system. In (Mortensen 2000) the CPN model was embedded into an environment that allowed the CPN model to control the sensors and actuators of the alarm system via a dedicated network. For COAST, we have embedded the CPN model in an environment for communicating with the COAST client, i.e., the graphical user interface.

The COAST client provides a front-end that makes the use of CPNs and state space analysis transparent to the user. This is important since commanders and planners are not expected to be able to use CPNs. The idea of providing a transparent and application specific interface to a CPN model was also considered in (Lindstrøm 2001) where a web interface for simulation of CPN models was considered. A main difference in our approach is that we consider analysis of CPN models, whereas (Mortensen 2000 and Lindstrøm 2001) only consider execution of CPN models.

The development of software tools to support military planning is an active area of research. Most software planning tools are focussed on supporting information sharing, group authoring, workflow automation, and other information management functions within a generic planning process to enhance collaboration among staff. Such tools can be referred to as collaborative planning tools. Some attention has been given to the development of decision support tools that aid the development, representation and analysis of COA during planning processes, and the effort up to 1988 was summarised in (Andriole and Hopple 1988). Planning and planning aids have been a subject of study in the area of artificial intelligence. The programs of work reported in (Andriole and Hopple 1988) represented an artificial intelligence approach, and addressed tactical planning/replanning problems with lesser degrees of ambiguity and uncertainty than the ones that we consider in this paper. Related work on developing decision support tools for planning has also been reported in (Wagenhals et al. 1998 and Wagenhals 1999). The approach in (Wagenhals et al. 1998 and Wagenhals 1999) involved construction of a timed influence network model of a mission using the Situational Influence Assessment Module (SIAM) tool (SAIC 1998). Subsequently, the influence network model is used to form the basis of a timed CPN model of sequenced and timed COAs. The CPN model of COAs was then analysed, using state space techniques, to assess the impact of sequencing and timing of events on the mission objective. The work reported in (Wagenhals 1999), however, does not consider resource implications of tasks or events. Another main difference between our approach and (Wagenhals 1999) is that we have one CPN model capable of executing any COA, whereas a CPN model is generated for each influence net in (Wagenhals 1999). A CPN model capable of executing any influence net was later presented in (Lindstrøm and Haider 2001) together with discussion of the pros and cons of this CPN model compared with the model in (Wagenhals 1999).

Other recent related work involves the development of a Tool for Operational Planning, Force Activation and Simulation (TOPFAS) to support North Atlantic Treaty Organisation (NATO) operational planning (Thuve 2001). TOPFAS emphasises the use graphical tools to visualise layouts of plans, layout of critical events in a plan, COA, and geographical location of tasks. A key output of the tool is a Statement of Requirements, which is a detailed description of what is needed to conduct an operation.

The remaining part of this paper is organised as follows. Section 2 describes our conceptual framework and approach to supporting COA development and analysis. Section 3 provides an overview of the constructed CPN model. Section 4 gives an overview of COAST focusing on the software architecture and the tool components. Section 5 presents the graphical user interface of COAST, and Section 6 presents analysis algorithms used in the COAST server. Finally, Section 7 gives the conclusions and an outline of future work. The reader is assumed to be familiar with the basic ideas of Petri nets.

2 Conceptual Framework

This section presents the conceptual framework underlying COAST. The purpose of developing this framework has been to identify and develop an understanding of the central concepts in COA development when going from a COA task specification to the corresponding lines of operation. The development of the framework is based on our earlier work in (Zhang et al. 2000 and Zhang et al. 2001), the ADF doctrine, and by studying tasks in representative COAs. The formalisation of this framework using CPNs will be presented in Section 3.

A key question that we intend for COAST to help answer is this: *given a list of tasks that are designed by military planners developing a COA, is there one or more lines of operations that when executed, will lead to the achievement of a mission without violating the constraint of given resources?* The question is significant for two reasons. Firstly, it helps to test the suitability of a proposed COA. For instance, a positive answer to the above question proves the logic in the proposed COA, i.e., the execution of tasks logically leads to the achievement of the mission. Secondly, the tool helps to test the feasibility of a proposed COA. For instance, the existence of an executable task sequence implies that

there is no conflict in resource requirements of tasks over the period of execution. In a military operation that has few tasks, human operators would be able to handle the work of sequencing and scheduling with a degree of confidence and rigour. But when the numbers of tasks and resources become large as in the case of most military operations, COAST would offer great assistance to planning.

The key construct in our framework is *tasks* as they (for our purposes) form the basic buildings blocks of a COA, and it is tasks that are to be sequenced and scheduled to form the lines of operation. According to the ADF doctrine, every military operation must have a pre-defined military *end state* to characterise its mission. A military end state is defined as the set of desired conditions beyond which the use of military force is no longer required to achieve national objectives. The keyword in this definition is *condition*, which we model as an assertion that can be true or false. Accordingly, an *initial state* represents the set of valid conditions at the beginning of an operation under planning. A line of operation specifies a possible way in which tasks can be sequenced (executed) to reach a desired end state. In its simplest form, a line of operation specifies the start-time and end-time for each task.

Having introduced the concept of initial state and end state, it follows that the tasks that are to be devised by planners describe actions that need to be taken by military forces to achieve the desired end state. It is however important to note that the development of appropriate tasks for the achievement of a desired end state is not trivial, it requires sound judgement and enormous military experience.

Given that tasks are devised to achieve a desired end state, how does a software tool such as COAST deduce the order of the tasks (apart from planner imposed sequences)? We assert that tasks are related and can be ordered by virtue of conditions, e.g., the execution of certain tasks requires conditions that can be present only when certain other tasks have been or are being executed. A line of operation is deemed to have reached a desired end state if the set of conditions that constitute the end state are present. The other important factor that determines if a task can be executed is the availability of resources required for the task.

From the above it follows that we consider two factors that determine whether there exist ordered sequences of tasks that lead to the achievement of planning objectives: cause-effect relationships among tasks in terms of conditions, and resource requirements of tasks. To the best of our knowledge, most planning tools address only one of these two factors. In general, once a task terminates some or all of its *post-conditions* will allow other tasks (if any) to have these conditions as *pre-conditions* for execution. Tasks can be executed in sequence or concurrently provided the conditions and resource constraints are satisfied.

The concept of tasks, conditions, and resources are illustrated with an example task named *Amphibious Assault* shown in Figure 1. The task is characterised by its attributes: *Task name*, *Pre-conditions*, *Post-conditions*, and *Resources*. There are additional task attributes not shown here, such as *triggers*, *task duration*, and *probability of success*. There are two kinds of pre-conditions for tasks: start and execute pre-conditions. *Start pre-conditions* specify a set of *pre-conditions* that has to be satisfied for the task to start executing. A start pre-condition may be invalidated when the task starts. An example of this is intelligence information. If a task has the presence of certain intelligence information as a start pre-condition, starting the task will usually make the intelligence information outdated and hence invalid. *Execute pre-conditions* are conditions that have to be satisfied for the task to execute. If an execute pre-

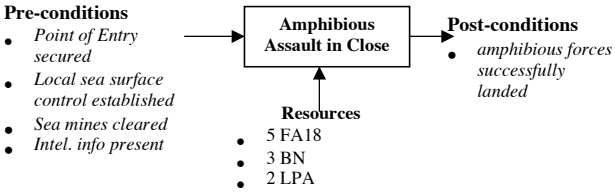

Figure 1: Example of a task.

condition of a task becomes invalid, the task will have to be *aborted*. An example of this is a task that requires air support to execute. If e.g., the task providing air support fails, the task will have to be aborted.

There are three kinds of post-conditions (also referred to as effects) for tasks: immediate, duration, and termination post-conditions. *Immediate post-conditions* are conditions that become valid when the tasks starts execution, and remains valid until possibly the start of other tasks invalidates them. *Duration post-conditions* are conditions that remain valid only while the task is executing. *Termination post-conditions* are conditions that become valid upon a successful termination of the task.

As mentioned earlier, the construction process of lines of operation is governed by the task conditions and task resources being satisfied. The start and termination of tasks affects the available resources and changes the current set of valid conditions.

The execution of tasks may also *fail* and thereby fail to establish some of its effect. The failure of a task may in turn imply that the desired end state cannot be reached. The failure of a task implies that its duration conditions are no longer valid which may in turn mean that the execute pre-conditions of other tasks are no longer valid and hence such tasks will have to abort. A task has special failure and abort conditions, which describes how failure and abortion of the task affects its conditions. The execution of tasks may also result in resources being lost. Execution of tasks and the set of valid conditions may also be affected externally from the environment.

A set of tasks can be *start-synchronised* meaning that tasks in this set have to start execution at the same time. One motivation behind this concept is that many tasks require simultaneous start to achieve the maximum effect. Similarly, a set of tasks can be *end-synchronised* meaning that the tasks in this set will have to terminate at the same time.

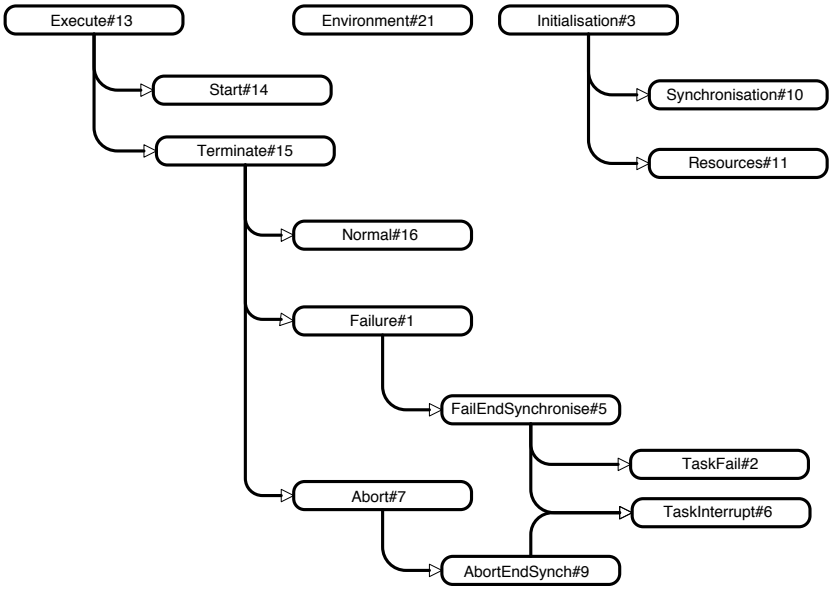

Figure 2: Hierarchy page of the CPN model.

3 CPN Modelling

In this section we show how the conceptual framework presented in the previous section has been formalised by the construction of a CPN model - in the following referred to as the CPN Task model. The CPN Task model captures the possible executions of tasks in a COA given the synchronisations between tasks in terms of conditions, start- and end-synchronisations, and available resources. The CPN Task model is complex and cannot be presented in full in this paper. Our objective is to provide an overview describing its structure and how it captures the execution of tasks in a COA according to our conceptual framework.

A key requirement to the CPN Task model was that it should be able to capture the execution of tasks in any COA. The reason behind this requirement is that we want to extract the CPN Task model from Design/CPN in executable form and embed it into COAST. Once embedded in executable form in COAST, it will not be possible to modify any of the Petri net structure (i.e., places, transitions, arcs, colour sets, and arc inscriptions) of the CPN Task model, only the initial marking can be changed. This requirement implies that tasks, start- and end-synchronisations, resources, and conditions have to be modelled as tokens of complex colour sets (types) that are being manipulated by the CPN Task Model. This ensures that the CPN Task model is highly parametric in that only its initial marking depends on the COA to be analysed. The CPN Task model may therefore be considered a virtual machine for the execution of tasks in a COA.

3.1 Overview

The hierarchy page of the CPN Task model is shown in Figure 2. Each node in Figure 2 corresponds to a page (module) of the CPN Task model and an arc between two nodes specifies that the destination page of the arc is a sub-page (sub-module) of the source page of the arc. The CPN Task model is timed since capturing time taken to execute tasks is important for obtaining and evaluating lines of operation. The CPN Task model comprises 14 pages, which can be grouped into three sets.

Initialisation. The *Initialisation* page and its two sub-pages are responsible for the initialisation of the CPN model, i.e. the initial distribution of tokens on the places of the CPN model according to the set of tasks, resources, initial conditions, and start- and end-synchronisations.

Execution. Page *Execute* and its 9 sub-pages model the execution of tasks. The sub-page *Start* models the start of executing tasks. The *Terminate* page and its sub-pages model the termination of tasks. The normal termination of tasks is modelled by page *Normal*. The failure of a task (which in turn causes termination of the task) is modelled by page *Failure* and its sub-pages. The abortion of tasks (as a consequence of other tasks failing or aborting) is modelled by page *Abort* and its sub-pages.

Environment. Page *Environment* captures external events (such as triggers) that may affect conditions and the execution of tasks. These external events currently include change of conditions and circumstances that cause failure of tasks.

3.2 Modelling Execution of Tasks

Figure 3 shows page *Execute* which is the top-most page modelling the execution of tasks, and Figure 4 lists the definition of the most important colour sets (types) in the CPN Task model. We explain both figures in more detail below.

All tasks in the COA are initially present as tokens on the place *Idle* (Figure 3) having the colour set *Task* (Figure 4). The colour set *Task* is a complex record colour set with a field for each of the attributes of a task. For brevity, we have omitted some of the fields in the record colour set. Altogether there are 22 fields in the *Task* colour set. As an example, the *name* field is used to specify the name of the task, and the *startprecond* field is used to specify the set of start pre-conditions that must be satisfied for the task to start execution. The colour set

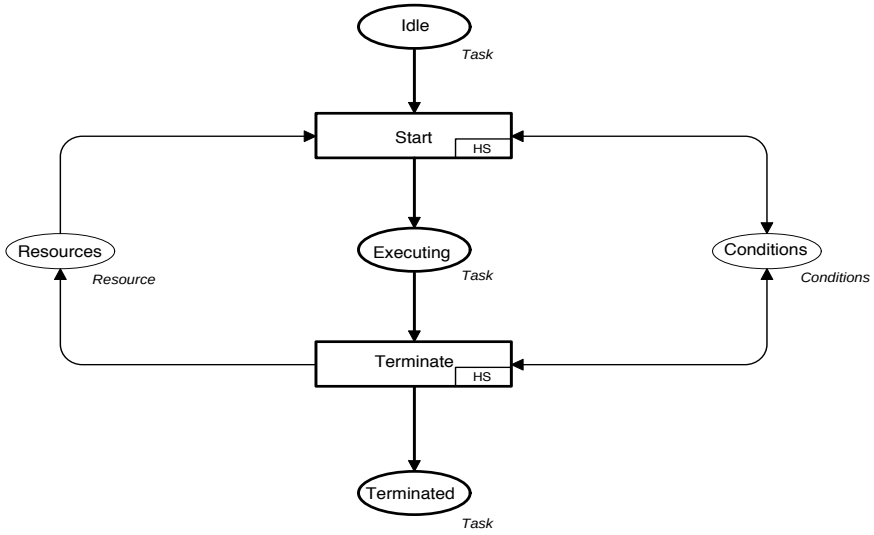

Figure 3: Page Execute.

SConditions is declared as a list of *SCondition* and is used for the specification of conditions for tasks. As an example, the pre-conditions of a task are specified by listing the names of the conditions required to be valid for the task to start. In a similar way, the field *resources* of type *Resources* is used for the specification of resources required for the task to start.

We have used strings (colour set *Name*) for the identification of tasks, resources, and conditions. In this way we avoid limiting the domain of tasks, resources, and conditions to a fixed set. This is important since we want to be able to handle COAs containing tasks that refer to conditions and resources that are not known in advance. We have used the `union` type constructor to ensure that conditions and resources are different types.

The current set of available *Resources* and the current state of *Conditions* are modelled by the accordingly named places in Figure 3. The current state of the conditions are represented by one token on place *Conditions* of colour set *Conditions*. This token is a list with an entry for each condition. A condition is modelled by the colour set *Condition* which is a pair consisting of the name of the condition and a Boolean. This Boolean is used to specify whether the condition is currently valid (true) or invalid (false). The current state of the conditions is modelled as a list to make it simple to inspect all conditions at once during start and termination of tasks. The current set of available resources is present as a multi-set of tokens on place *Resources* with colour set *Resource*.

The sub-pages of the substitution transition *Start* model in more detail the start of executing a task. When a task starts executing the corresponding token will be removed from *Idle* and put on place *Executing*. At the same time, the resources required by the task will be removed from place *Resources* as these resources are now being used and hence currently unavailable. Moreover, the state of the current set of conditions is updated according to the condition attributes of the task under execution. The sub-pages of the *Start* substitution transition also model the start synchronisation of tasks ensuring that tasks that are start-synchronised will start at the same time.

The sub-pages of the substitution transition *Terminate* model in more detail the termination of tasks. When a task terminates, the corresponding tokens will be removed from place *Executing* and put on place *Terminated*. Also, the resources used by the task will be returned (taking possible loss into account), and the current set of conditions is updated according to the condition attributes of the task.

The modelling of start and termination of tasks in the sub-pages of *Start* and *Terminate* involves the use of rather complex Standard ML (Ullman 1998) arc inscriptions, and will not be described in detail in this paper.

```
color Name = string;
color Bool = bool;
color NamexBool = product Name * Bool;
color Condition = union
                  CONDITION : NamexBool;
color Conditions = list Condition;
color SCondition = union
                   SCONDITION : Name;
color SConditions = list Scondition;
color Resource = union
                 RESOURCE : Name;
color Resources = list Resource;
color Task = record
           name           : Name *
           startprecond   : SConditions *
           exeprecond     : SConditions *
           startpostcond  : SConditions *
           durpostcond    : SConditions *
           termpostcond   : SConditions *
              . . .
           resources : Resources ;
```

Figure 4: Colour set declarations.

4 COAST Overview and Architecture

The CPN Task model presented in the previous section constitutes the semantic foundation of COAST. Figure 5 shows the client-server software architecture of COAST. The *COAST client* is implemented in the JAVA programming language whereas the *COAST server* is implemented in Standard ML of New Jersey (SML/NJ) (Ullman 1998). The core of the *COAST client* is the

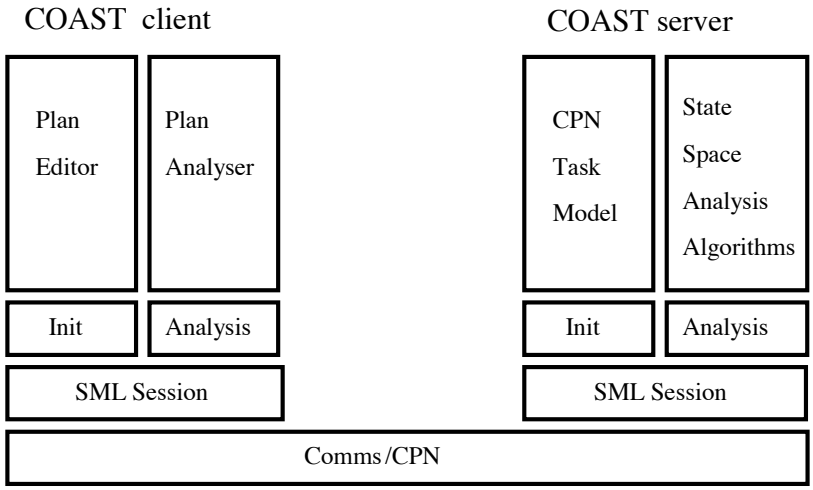

Figure 5: Software architecture of COAST.

graphical user interface (GUI) for the *Plan Editor* and the *Plan Analyser*. The Plan Editor allows the user to specify tasks, resources, and synchronisations for a COA that is to be developed and analysed. The Plan Analyser allows the user to generate, visualise, and evaluate lines of operations. The Plan Editor and Analyser will be presented in more detail in Section 5. The core of the server is the *CPN Task Model* and the *State Space Analysis Algorithm*s that constitute the computational engine of COAST. The server is implemented by extracting the compiled CPN Task model and an extended set of State Space Analysis Algorithms from the Design/CPN tool embedded in the SML/NJ runtime system. We will present the state space analysis algorithms in more detail in Section 6.

Communication between the client and the server is based on Comms/CPN (Gallasch and Kristensen 2001), a high-level library supporting communication between CPN models and external applications using TCP/IP. An *SML Session* layer has been implemented on top of Comms/CPN. This layer provides the service that allows the client to invoke functions in the server and receives corresponding results. The SML Session layer is implemented by allowing the client to submit SML code, to the server for evaluation. The fact that the server contains the SML run-time system and interpreter is exploited here. The received SML code is then evaluated and executed by the server, and results are sent back to the client. The *Init* layer is built on top of the SML Session layer and implements the Remote Procedure Call (RPC) mechanism that allows the client to initialise the CPN Task model, i.e. to set its initial marking and parameters according to the COA that is to be analysed. The *Analysis* layer is similar, and implements the RPCs that allow the client to conduct analysis of the COA by invoking the analysis functions in the State Space Analysis Algorithms component.

The advantage of the client-server architecture is that it allows the computationally demanding analysis to be executed on a powerful workstation while running the GUI (client) on a less powerful PC. Another reason for the choice of a client-server architecture was that most potential users of COAST would be using PCs running Windows. Design/CPN, used to create the CPN Task model that provides the semantic foundation of our approach, is only available on Unix-based systems. With the client-server architecture, the client can be running on a Windows-based PC and the server on a Unix-based workstation. With the next generation of CPN Tools (CPN Group 2002) it will become possible to also run the server on Windows-based systems.

5 Graphical User Interface

A COA being developed and analysed with COAST consists of four components. The first component is the set of tasks associated with the COA. Subsequently, all potential lines of operation generated for this COA will contain tasks that are a subset of this set. The second component is the task synchronisation information. This provides constraints upon the scheduling of tasks, by specifying tasks that are to be start- and/or end-synchronised. The third component is a pool of assigned resources from which the tasks can draw in order to be carried out. The fourth component is the set of conditions used as pre-conditions and post-conditions of tasks.

The COAST client provides a GUI that allows users to interact with both the COAST Editor and Analyser in order to input these four components and to develop the corresponding lines of operation. Figure 6 provides an example of the details typically shown by the GUI during COA editing. As can be seen, the GUI consists of a main window titled *In-MODE/COAST*, along with a menu bar and four separate, movable and resizable sub-windows. These sub-windows display details of the four components of the COA: the upper left sub-window contains the list of tasks, the upper right contains the synchronisation information, the lower left contains the assigned resources, and the lower right contains the list of conditions. As part of the COAST Editor, each of these sub-windows has an associated Editor form.

The menu bar is used to initiate entry of COA information into the tool, enables modification of an existing COA, and provides ability to generate lines of operation for a COA. Briefly, the items on the menu bar: *Plans, Tasks, Synchronisations, Resources* and *Conditions* provide the basic functionality to manipulate (create, load, edit, etc.) the elements from each of the four

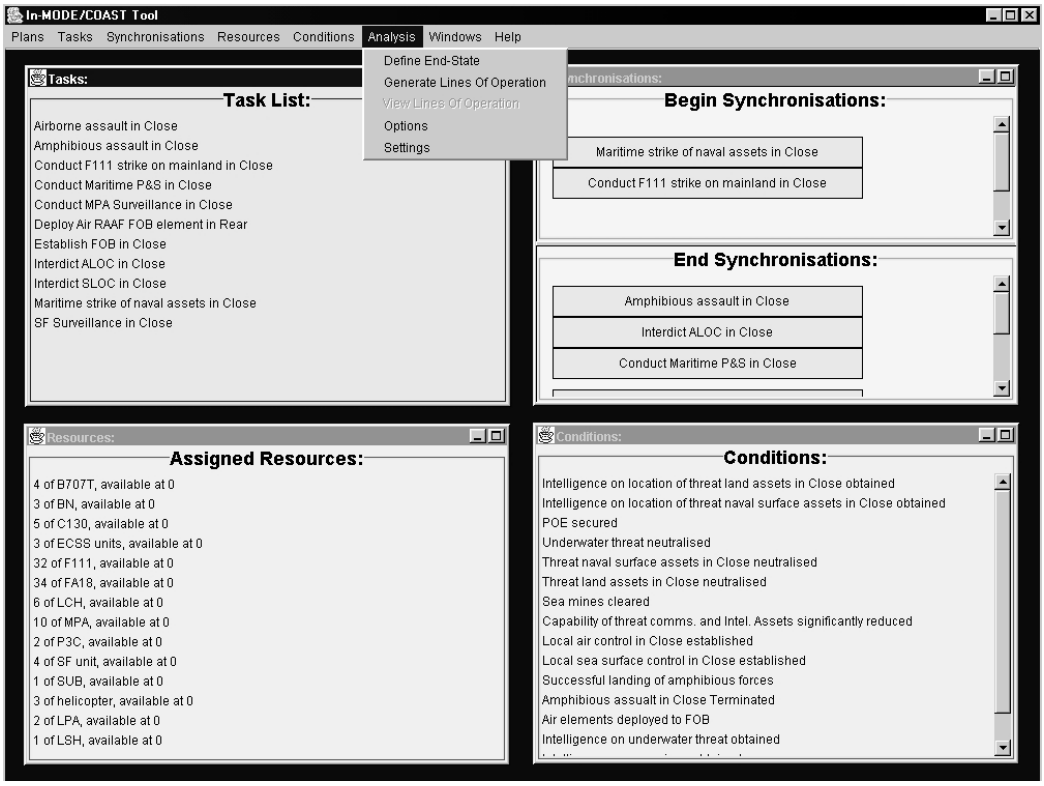

Figure 6: Main GUI window of COAST.

components shown in Figure 6. The *Analysis* menu item allows the desired End-State for any line of operation to be specified, for the COA to be sent to the COAST server for analysis, and for the viewing of generated lines of operation. Details of the generation of lines of operation by the server and the viewing of such lines are described in more detail in the next section.

Figure 7 illustrates the Task Editor form with our earlier example task: *Amphibious assault in Close* being edited. This same editor form is used when editing existing tasks and also when creating new tasks from task templates or from scratch. The information contained within the Task Editor form can be divided into three parts.

The first part, at the top of the Task Editor form, encompasses a number of attributes that could be described as General Attributes. As can be seen in Figure 7 there are two text fields to allow entry of a task name (*Name*) and a comment (*Comment*) to describe the task. Below these two fields are six other fields that take the form of pull-down lists. *Duration* is an integer specifying the length of the task in hours. *Success Probability* allows the planner to specify the chance of success of this task expressed as a percentage. *Environment* specifies the military environment to which this task belongs, either Maritime, Air, Land or Joint. *Battlespace* defines the area in which the task will be acting, either in Close, in Deep or in Rear. Important tasks can be marked as such using the *Tags* field – in this case the task is marked with the DP (Decisive Point) tag. The *Trigger* field allows a trigger to be specified, such as a calendar date, to trigger the execution of a task. The values of these six attributes can be changed using the associated pull-down lists.

The second part is called *Task Resources*, which displays and allows the editing of resource requirements for a task to begin execution as well as any resource losses upon successful task completion, interruption and failure. A set of four selections in the form of buttons is provided on the left of this section. They are labelled *Required, Lost if successful, Lost if interrupted,* and *Lost if failed*. Switching between these allows the editing of corresponding resource requirements or losses. This section also contains two lists, called *Task Resources* and *Assigned Resources*. The Task Resources list contains the list of resources associated with this task corresponding to whichever of the four buttons that is currently selected. Figure 7 shows the Required field as being currently selected. The Assigned Resources list corresponds to the pool of resources described earlier in this section. Between these two lists is a mechanism for transferring resources from one to the other reflecting the allocation of resources to tasks. The resources required for any given task must be a subset of the resources specified as Assigned Resources. In addition, the resource losses (on success, interruption or failure) for any given task must be a subset of the resources required for that task.

The third part called *Task Conditions* allows the pre-conditions and post-conditions associated with this task to be specified.

6 Lines of Operation Analysis

One of the primary aims of analysis is to support the planner in obtaining the lines of operation for the COA given the set of tasks, assigned resources, synchronisations, and initial conditions. Another important aim is to support the comparison between lines of operation by

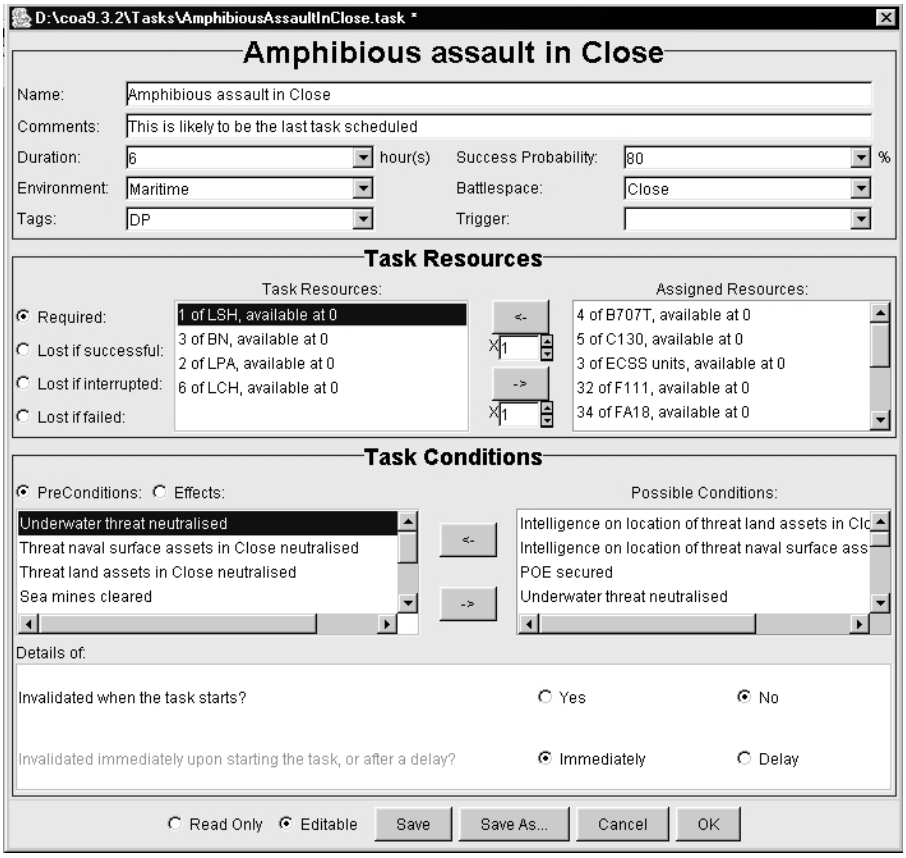

Figure 7: Task Editor.

means of quantitative measures such as duration, probability of success, and cost. For example, the campaign duration, overall probability of success, and the campaign cost in terms of the total loss of resources can be calculated for each line of operation obtained in order for the commander to make informed decisions. In this paper we focus on how the possible lines of operation are obtained.

An important aspect of the analysis is to support the planner in identifying inconsistencies and deficiencies in the set of tasks in the COA, and to determine whether the COA is feasible, i.e. can be executed with assigned resources. Problems in the COA will manifest themselves by the absence of lines of operation leading to the desired end state. Inconsistencies in a COA can for instance be caused by conditions not having been established for executing a certain essential task. If such inconsistencies are detected, the planner will have to go back and modify the COA, e.g. by adding new tasks, modify existing tasks, and adding resources. The construction of a consistent and feasible COA is therefore an iterative process.

The analysis in the COAST server is based on the state space method of CPNs (Jensen 1992). The basic idea of state spaces (also called reachability/occurrence graphs) is to compute all possible executions of the CPN model and represent these as a directed graph where the set of nodes represents the set of reachable states and the arcs correspond to transition occurrences. In terms of the CPN Task model, the state space represents all possible ways to execute the set of tasks in the COA given the assigned resources and interactions between tasks in terms of synchronisations, resources, and conditions. The structure of the CPN Task model guarantees that the state space is finite, i.e. has a finite number of nodes and arcs, for *any* finite initial set of tasks, resources, and synchronisations. Moreover, since tasks in a COA are only executed once, the state space is an acyclic graph.

A path in the state space corresponds to the execution of a set of tasks in the COA. The basic idea is to derive the lines of operation for the COA from the paths in the state space and return them to the COAST client. Figure 8 gives an example of how the lines of operations are presented to the planner in a graphical form in the client. In this example, a Gantt chart is used to visualize the sequencing of tasks representing a single line of operation returned from the server. Time markers have been placed at 50-hour intervals, represented by vertical dashed lines. Task names are listed in a column on the left, sorted in ascending task start time order. Each task is represented by a rectangular bar, which is positioned according to the task start time, and sized horizontally relative to the task duration.

To obtain the lines of operation, the planner needs to characterise the desired end states. This is done by specifying a set of conditions that all must be satisfied for a state to qualify as a desired end state. The generation of the lines of operation is based on a breadth-first traversal of the state space. The idea is to associate with each node (state) of the state space the lines of operation that can lead to this state. A line of operation is represented as the start-time and end-time of each task. The lines of operation for a given node (state) is computed from the

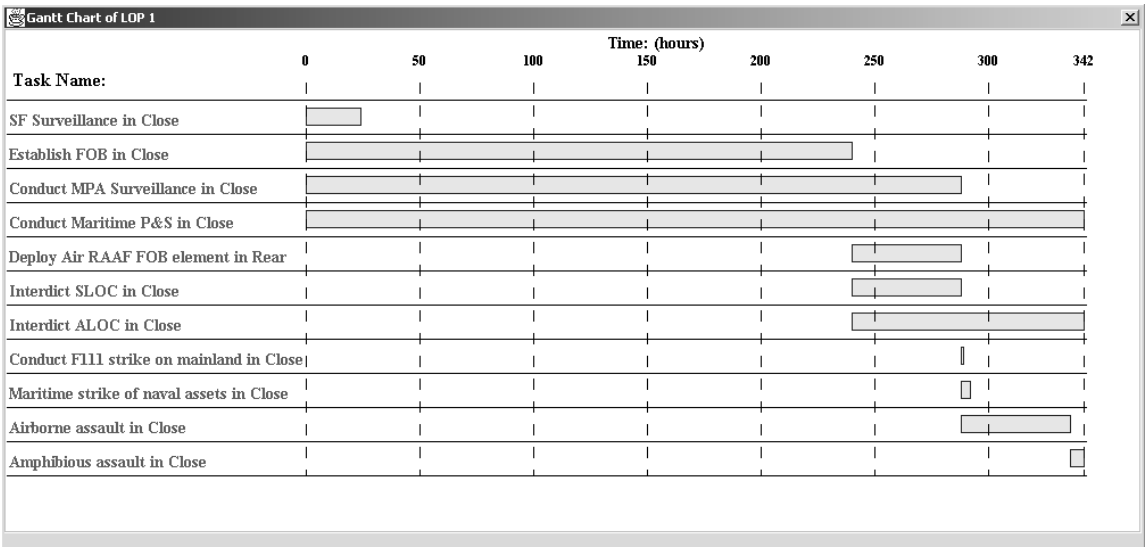

Figure 8: Visualisation of a line of operation.

lines of operation associated with its immediate predecessor states in the state space. The breadth-first generation and the fact that the state space is acyclic, ensure that the lines of operation have been fully computed for the predecessor states of a given state S before the lines of operation are computed for the state S. The breadth-first generation also ensures that the shortest lines of operation are obtained.

The breadth-first traversal starts from the initial state, which has the empty lines of operation assigned to it. When processing a state S during the breadth-first traversal, the lines of operation for a successor state S' of S are computed by adding the lines of operation computed for S' until now (as it may have other predecessors) and the lines of operation stored with S. The lines of operation for S are extended before being added to the lines of operation for S' in case the output arc from S to S' corresponds to the start or termination of a task. Any duplicate lines of operations are removed from the updated lines of operation stored with S'. This removal of duplicates is required since the state space represents all interleaved executions of the CPN Task model, and hence two different paths in the state space can represent the same line of operation in terms of start-time and end-time for tasks. After S has been processed, the lines of operation stored with S are deleted as they are not required any longer. The breadth-first traversal is truncated at states that qualify as desired end states according to the set of conditions specified by the planner. When the breadth-first traversal terminates the lines of operation are obtained from the lines of operation associated with the set of desired end states where the breadth-first traversal was truncated.

The state space for a relatively simple COA with 11 tasks, 89 resources of different kinds, 16 conditions, 2 start-synchronisations, and 1 end-synchronisation had only 144 nodes and 171 arcs, and could be generated in a few seconds on a standard PC running Linux. The generation of the lines of operation was also done in a few seconds. A state space with 144 nodes and 171 arcs is a small state space even for 11 tasks, and the main reason for this is that the pre- and post-conditions of tasks puts rather tight constraints on the possible interleavings of tasks. This observation justifies that the often encountered explosion in the number of reachable states caused by representing all possible interleaved executions is not going to be of major concern in analysing COAs and obtaining the lines of operation. State spaces hence seem to provide a feasible analysis approach.

7 Conclusions and Future Work

Planning in the defence context is a very important but complex activity that requires the use of a military commander's experience and intuition, supported by suitable computer tools. The military planning process commences with a mission statement by the commander, which sets the highest-level objective for a military campaign. To achieve this objective a set of strategies are canvassed, which lead to the development of a number of COAs. These COAs are at a high-level of abstraction. Each of these COAs specifies a set of tasks that need to be sequenced in a manner that satisfies various constraints (e.g. on resources), including the important goal of achieving the desired outcome of the campaign. Each COA can then be refined into a set of lines of operation that specify the sequencing of tasks. These lines of operation are then analysed and chosen for further elaboration into a detailed plan.

This process can be seen to have similarities with the notions of top down design (start with the most abstract specification of the mission), utilising successive refinement steps (see how to achieve the mission via COAs, then determine the lines of operation, and elaborate into a detailed plan), that are used in systems and software engineering. This led us to consider the application of formal methods to provide a rigorous approach to the development of military plans.

In this paper we have presented COAST, a tool for COA development based on CPNs. The specific benefits of using CPNs lie in the capability to construct a highly compact and highly parameterized CPN model where the

COAs to be developed and analyzed are provided by setting the initial marking (state) of the CPN model. Moreover, CPNs allow models to be timed which is necessary to obtain the lines of operation in the COA. The specific benefit of using Design/CPN was that the architecture of Design/CPN makes it possible to extract the CPN model in executable form and embed it into COAST. In effect this means that the process of formalizing our conceptual framework for COA development and COA task execution gives an implementation of this framework that can be directly used as the semantic foundation of COAST.

A prototype of COAST is now available, and has been used to assess a military exercise. The paper shows how a line of operation can be generated by COAST and presented to the planner as a Gantt chart (a sequence of time delineated tasks). This is just a first step. Further feedback to the planner is envisaged, including presenting problems with lines of operation, e.g. that are generated but do not satisfy the desired properties of feasibility and suitability. We also believe that the approach presented in this paper, and COAST, will be applicable in many civilian domains such as emergency services planning and deployment, foreign aid projects, developing logistics for sporting events and large scale construction projects.

Acknowledgement. The authors wish to acknowledge the members of Systems Simulation and Analysis Group in DSTO for their support for the development of COAST, and ADF military planners for their ideas and inspiration.

References.

ADF (1999): *Joint Military Appreciation Process. Volume 9 of Operations Series, Joint Planning*. Australian Defence Force, Chapter 8.

Kristensen, L.M., Mitchell, B., Zhang, L. and Billington, J. (2002): Modelling and Initial Analysis of Operational Planning Processes using Coloured Petri Nets. In Proceedings of *Workshop on Formal Methods Applied to Defence Systems*. Adelaide, June 2002.

Jensen, C.V. (1996): *An Introduction to Bayesian Networks. London UCL Press*, pp. 48-52

Falzon, L., Zhang, L. and Davies, M. (2001): *Hierarchical* Probabilistic Models for Operational-Level Course of Action Development. In *Proc. of the 6th Command and Control Research and Technology Symposium*, Annapolis, MD, VA, June 2001.

Jensen, K. (1992): *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volumes 1-3, Monographs in Theoretical Computer Science, Springer Verlag, 1992-1997.

CPN Group (1996): *Design/CPN Online*. www.daimi.au.dk/designCPN

Mortensen, K.H. (2000): Automatic Code Generation Method based on Coloured Petri Net Models Applied to an Access Control System. In *Proc. of International Conference on Application and Theory of Petri Nets*. Vol. 1825 of Lecture Notes in Computer Science, pp. 367-386. Springer-Verlag.

Lindstrøm, B. (2001): Web-based Interfaces for Simulation of Coloured Petri Net Models. In *International Journal on Software Tools for Technology Transfer*, Volume 3, Issue 4, pp. 405-416. Springer-Verlag.

Andriole, S.J. and Hopple, G.W. (1988): Tactical Planning and Replanning. In *Defence Applications of Artificial Intelligence*. Andriole, S.J. and Hopple, G.W. (eds.), Lexington Books.

Wagenhals, L. (1999): *Course of Action Development and Evaluation Using Discrete Event System Models of Influence Nets*, PhD. Thesis, George Mason University.

Wagenhals, L., Shin, I. and Levis, A.H. (1998): Creating Executable Models of Influence Nets with Coloured Petri Nets. In *International Journal on Software Tools for Technology Transfer*, Volume 2, Issue 2, pp. 168-181. Springer-Verlag.

SAIC (1998): *Situational Influence Assessment Module User's Guide*, Version 2.0, Science Application International Corporation, McLean Virginia.

Lindstrøm, B. and Haider, S. (2001): Equivalent Coloured Petri Nets Models of a Class of Timed Influence with Logic. In *Proceedings of Workshop and Tutorial on CPNs and CPN tools*, pp. 35-54. DAIMI-PB 554, Department of Computer Science, University of Aarhus.

Thuve, H. (2001): TOPFAS (Tool for Operational Planning, Force Activation and Simulation), *6th International Command and Control Research and Technology Symposium*, Annapolis, MD, VA, June 2001.

Zhang, L., Falzon, L., Davies, M. and Fuss, I. (2000): On Relationships between Key Concepts of Operational Level Planning. In *Proc. of the Command and Control Research and Technology Symposium*, Australia War Memorial, Canberra ACT, Australia, 24-26 October 2000.

Zhang, L., Mitchell, B., Falzon, L., Davies, M., Kristensen, L.M. and Billington, J. (2001): Model-based Operational Planning Using Coloured Petri Nets. In Proc. of *the 6th Command and Control Research and Technology Symposium*, Annapolis, MD, VA, June 2001.

Ullman, J.D. (1998): *Elements of ML Programming*. Prentice-Hall International.

Gallasch, G. and Kristensen, L.M. (2001): Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In *Proceedings of Third Workshop and Tutorial on CPNs and CPN Tools*, pp. 79-93. DAIMI PB-554, Department of Computer Science, University of Aarhus.

CPN Group (2002): CPN Tools. www.daimi.au.dk/CPNtools