# Damage Control Domain: Using Petri Nets for Intelligent Scheduling

Vadim V. Bulitko & David C. Wilkins

Beckman Institute, University of Illinois at Urbana-Champaign
405 N. Mathews Ave., Urbana, IL 61801, USA
{bulitko, dcw}@uiuc.edu

**Abstract.** Many Artificial Intelligence decision-making systems operate in cycles. A typical problem-solving cycle might consist of deliberation, scheduling, and execution steps. At the deliberation step a number of feasible actions are generated. They are then evaluated by a scheduler at the scheduling step and the most suitable action is performed at the execution step. This paper presents a scheduling approach that uses the mathematically well-founded framework of Petri Nets as an environment model. The paper extends the classical Petri Nets (PT-nets) model in various ways and shows their applications to the test-bed domain of ship damage control. The use of a scheduler based on Time Interval Petri Nets in the domain of ship damage control resulted in 21 of 160 ships being lost in an immersive simulated environment. This is a 46% improvement over the performance of subject experts where 39 ships were lost.

## Introduction

A typical problem-solving cycle within a decision-making AI system might consist of the following three steps: deliberation, scheduling, and execution. For instance: in blackboard systems [13, 6, 22, 4, 19] during the deliberation step the knowledge sources are consulted and a set of proposed actions is posted on the blackboard; at the scheduling step the proposed actions are evaluated using some utility-type metrics and the best action is selected; and finally at the execution step the selected action is passed to the actuators/environment and the blackboard is updated to reflect the new state of the environment. The cycle is then repeated [e.g. 3, 5, 4, 18].

Scheduling is important because often several actions are feasible at a time while the resources are typically limited and don't allow to take all the actions. Equally important, some actions can be mutually exclusive and even contradictory. Therefore, the most suitable actions should be selected and executed. Naturally a powerful scheduling mechanism can often noticeably improve the overall operation of the AI system it is used in. Some interesting relevant experiments are presented in [3].

This paper focuses on the scheduling step by proposing a model-based scheduling approach. The scheduler represents the environment model as a Time Interval Petri Net (TIPN). The reward values come from a static state evaluator that evaluates TIPN-predicted states. Previous model-based scheduling approaches include: decision-theoretic dynamic programming [24], qualitative reasoning [12], and numerical simulation [10].

There are several strong motivations for the use of Time Interval Petri Nets for blackboard scheduling including:

1. Higher-level Petri Nets, and TIPNs in particular, have been shown to have good representational and reasoning power for domains with complex causality and time-structure function models [2]. They are capable of representing context (i.e. variables), timing information, and procedural knowledge. Furthermore, several promising machine learning algorithms have been recently developed. We are currently studying the degree of automation those methods deliver in synthesizing TIPNs. The ability to synthesize TIPN models automatically together with the experiments described in this paper suggest that TIPNs scale up to practically useful domains.

2. The Petri Nets community has developed a host of powerful theoretical methods for Petri Nets analysis. While the extra representational and inferential power of TIPNs makes most of questions undecidable in the general case, some special cases can still be analyzed. Analysis methods that have been relevant to our refinement of the TIPN domain model include the automatic graph reachability and liveness analysis. Many of those methods are available as software packages developed by the Petri Net community. The ability to analyze Petri Nets facilitates optimizing and refining the TIPN models used for scheduling.

## Overall Scheduler Design

At a high level the main idea of our *scheduling approach* is rather simple: predict the state(s) of the environment given the action being evaluated; then evaluate the states using a static state evaluator. The reward values are combined probabilistically to come up with a single utility value for each action. The actions are sorted by that value and the best one is executed.

Our approach is model-based [24]. That is, unlike Q-learning, an explicit model of the environment is used. The environment model has the following two main functions: (1) it envisions the future states of the environment given the current state and the action the agent is about to take; and (2) it envisions the reward the agent will get if it takes the action. While in decision theory such a model is often represented as the transition probability and reward functions, in our case the environment is modeled as a Time Interval Petri Network (TIPN). The TIPN itself does not contain any reward information. The reward values come from a static state evaluator [3].

The state evaluator is used as a "black box" that has several inputs representing the state of the environment and several outputs representing the severity of the state with regard to the problem-solving tasks (Figure 1). In the test-bed domain of ship damage control the environment is the ship (either real or simulated) from a perspective of the Damage Control Assistant (DCA). A DCA is the officer in charge of handling crises aboard a ship. The DCA's functions include recognizing crises and properly responding to them by giving orders to repair stations and the like. Thus, the input of the state evaluator is a state of the ship: status of compartments and vital equipment with regard to fire, flooding, etc. The output is severity of the ship state and is inversely proportional to the time till a predicted major disaster ("time till ship lost") [3]. Artificial neural networks [9] and decision trees [20,21] are used as the internals of the "black box". Both have shown comparable performance on the data sets that were used.

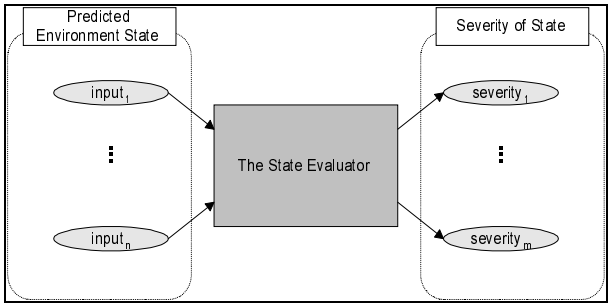


**Fig. 1.** Static state evaluator as a black-box with state attributes as inputs and state severity values as outputs. The predicted environment state is obtained from the environment model (the predictor) while the multi-dimensional severity of state constitutes a score of the state with regard to our decision-making task.

## Time Interval Petri Nets

This paper proposes a set of extensions for making Petri Nets suitable for scheduling in AI systems. It also compares the proposed extensions to related PN research.

### TIPN Extensions

While being a powerful and attractive modeling tool, the classical Petri Nets (also known as Place Transition or PT nets) [17,23] lack several features helpful for modeling complex real-time systems [11]. The four limitations of PT-nets this paper addresses are as follows: (1) propositionality (lack of colour or timestamp support); (2) instantaneous transitions; (3) lack of uncertainty support; (4) inability to do zero-testing.

To address those concerns several extensions are made to PT-nets. The extended formalism is called Time Interval Petri Nets (TIPNs). The extensions are illustrated with a simple example (Figure 2) which introduces an TIPN modeling fire ignition. Related Petri Net extensions by other authors will be considered as we proceed.

1. There is now support for context by making each token bear an identifier. An identifier is a set of pairs <*type*,*value*>. Example: a token with identifier *{<Space, 3-370-0-E>}* in place "*Ignition Temperature*" is interpreted as "*Compartment 3-370-0-E reached the ignition temperature*". This extension addresses the propositionality and lack of variables/context issue of PT-nets. Similar extensions have been suggested by other authors under a common name of colouring. In Coloured Petri Nets each token is assigned a vector of colour values [1,11]. Our approach is slightly different in that we explicitly specify the type of each value in the identifier.

2. In addition to a domain identifier each token now has a time *interval* associated with it. Time intervals are of form $[t_{beginning}, t_{ending}]$ that represents our belief on when an attribute *acquired* its value. Example: the token with domain identifier *{<Space, 3-78-0-M>}*, time interval *[3:12, 3:44]* in place "*Ignited*" represents our belief that

compartment *3-78-0-M* became ignited sometime between 3min 12sec and 3min 44sec of the scenario time. Naturally, if we know the time precisely the beginning and ending times will coincide. This extension allows to do temporal reasoning explicitly. It also supports reasoning under uncertainty as follows. If we consider a timestamp of an event to be a random variable then our time intervals translate into well-known *N*% confidence intervals where *N* is close to 100% (e.g. 95%). Similar extensions were produced by other researchers. Petri Nets extended in such a way are known as time or timed Petri Nets [1,2,11]. Like in [1,2] we maintain the central model clock so that the transitions fire in the order of their enabling times. However, while most researchers timestamp their tokens, we mark each token with a time interval that has timestamp as its special case (i.e. when the ends coincide).

3. Transitions are no longer instantaneous but have a delay interval associated with each of them. Each transition *t* has an associated delay interval $[\Delta_{min}, \Delta_{max}]$. A translation to the probability theory can be done as follows: if *t*'s delay $\Delta_t$ is a random variable then $[\Delta_{min}, \Delta_{max}]$ is the *N*% confidence interval for $\Delta_t$ (again *N* is close to 100%). This extension brings us closer to the event calculus as opposed to the situational calculus [24]. Again there are other researchers who have used time intervals for the transitions. Merlin in [14,15] has used time intervals to represent minimal and maximal enabling times, however the Petri Nets themselves were not coloured. Van der Aalst in [1] used transition delay intervals in his ITCPN (Interval Timed Coloured Petri Nets). Our approach differs from ITCPN in that we maintain the interval calculus *throughout* the representation while ITCPN tokens are timestamped (i.e. have a point-value time).

4. New types of arcs are introduced. Those include:
   - NOT-arcs (inhibitory arcs) mandate that a transition the arc leads to will not fire if there is a corresponding token in the originating place. Inhibitory arcs make TIPNs Turing-machine-equivalent in terms of modeling power [23].
   - Double-ended arcs do not withdraw tokens from their enabling places (e.g. the arc between "*Flammable Materials*" and the transition "*Ignition*" in Figure 2). It should be noted that our model doesn't allow the very same firing set to fire twice. In the example below (Figures 2 and 3) both compartments will get ignited only once.
   - Operator-arcs have arbitrary operators associated with them.

   Related work has been reported in [1,11].

**TIPN Operation**

In this section we step through a simple example showing an TIPN in action. We will start with the an TIPN modeling ignition. Figure 2 presents the initial configuration (step 0) where place "*Flammable Materials*" is filled with two tokens labeled with compartment (room) names.

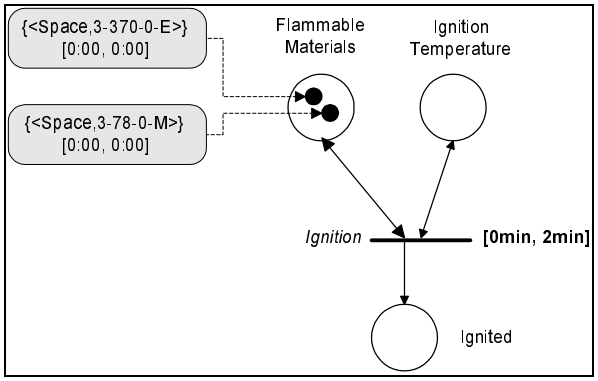


**Fig. 2.** TIPN modeling fire ignition at a coarse level. Step 0: the initial configuration

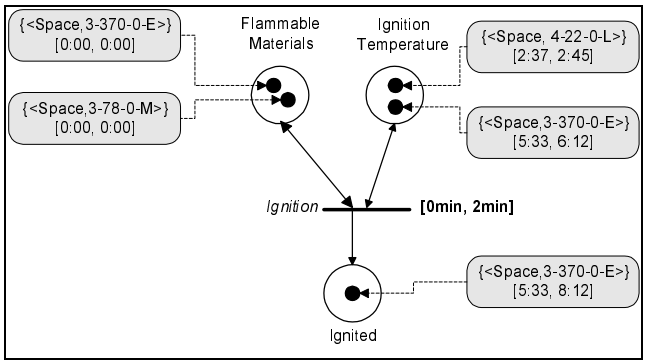


**Fig. 3.** TIPN modeling fire ignition at a coarse level. Step 2: generator room becomes ignited

Both tokens have their time intervals set to *[0:00,0:00]* meaning that the compartments have been flammable since the beginning of scenario. The other enabling place is labeled "*Ignition Temperature*" and contains compartments that have reached a sufficiently high temperature.

In step 0 that place has no tokens in it meaning that all compartment are cool enough. The two places are enabling places for the "*Ignition*" transition representing the event of ignition. The transition is labeled with the time delay interval. The output label of the transition is naturally labeled "*Ignited*" representing a compartment being ignited.

In step 1 a new token is posted in place "*Ignition Temperature*". However, the token has a label *{<Space, 4-22-0-L>,[2:37,2:45]}* indicating that the passageway *4-22-0-L* became hot sometime between *2:37* and *2:45* scenario time. This news has no effect on the net since the passageway is not known to contain flammable materials.

In step 2 the generator room compartment *3-370-0-E* becomes hot and a new token is posted in place "*Ignition Temperature*". This time the space labels match and the transition fires resulting a token in place "*Ignited*" and reflecting upon the event of generator room ignition (Figure 3).

There are several observations to make: (1) the enabling places retained their tokens as the enabling arcs were double-ended; (2) the time-interval of the new token was affected by the delay interval of the transition.

**Designing a TIPN-based Scheduler**

In this section we will go through a possible process of designing a system that uses TIPNs for model-based scheduling. An instantiation of this process for the ship damage control domain is presented in a later section together with the experimental results.

The first step in designing a system that uses TIPNs for model-based scheduling is to construct an TIPN that represents the physical causality and timings in the domain. The places typically correspond to states of the environment and the transitions reflect the state change information and timings. The arcs represent the causality information and procedural knowledge (through the operators attached to them). This is currently the most labor-intensive step since it is done by manually eliciting knowledge from domain experts. Automating this step is an area of current research.

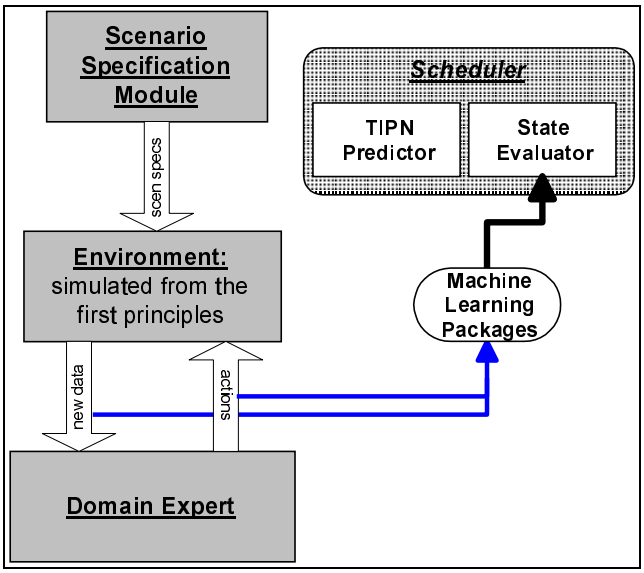


**Fig. 4.** Designing a state evaluator: a number of scenarios are being run with a "teacher" (i.e. a human domain expert or another expert system) and the environment states and the agent's actions are being logged and used to train the static state evaluator.

The second step is to design a state evaluator to work with the TIPN predictor. One of the important decisions to make is the choice of state severity scale and state attributes. Once those two are selected one can turn to designing the evaluator's internals. Our design decisions have been to utilize a static state evaluator using artificial neural networks or decision rules as the internals. Fortunately, both formalisms allow for automated synthesis as follows. A number of training traces can

be collected (either off actual or simulated system logs as shown in Figure 4. The simulation could be done with some existing, possibly slow, off-line simulator). Each state in the traces is to be represented with the attributes chosen and to be annotated with the severity values as per the severity metric chosen. Finally the annotated traces can be used as the training samples to synthesize an ANN or a decision tree. Commercial packages could be used for this substep [9, 20, 21].

Once trained the system can be used as highlighted in the introduction. Roughly speaking, every feasible action suggested at the deliberation step is fed into the TIPN environment model together with the current state of the environment. The TIPN model attempts to predict the effects of the action. The predicted states are evaluated with the state evaluator and the action with most desirable effects is carried out. A detailed description could be found in [3].

As demonstrated in the experimental evaluation section, the system trained in this manner can indeed outperform its teacher.

## Experimental Evaluation

### Implementation in Minerva-DCA

The TIPN framework has been tested as a part of Minerva-DCA blackboard expert system written for the ship damage control domain [25]. Below are highlights of the TIPN implementation within Minerva-DCA while the most detailed treatment is given in [3].

The deliberation module of Minerva-DCA generates a number of feasible actions while solving a crisis. Each action corresponds to a high-level goal (e.g. process(hypothesis), findout(finding), etc.). Example: action "*fight fire in X*" might correspond to hypothesis "*fire in compartment X*".

Minerva-DCA has a hand-coded TIPN part of which is shown in Figure 5. Each deliberated action is fed into the TIPN to assess utility of the action. Simply put, overall utility of an action is the sum of its operational and goal utilities. Operational utility is the difference between the value of predicted ship state if the action is taken and the value of predicted ship state of the action is not taken. Intuitively, operational utility reflects how much good an action is going to do us if taken now. Goal utility reflects how important the goal, that the action being evaluated is pursuing, is. It is defined as the difference between the value of the predicted ship state assuming that the goal hypothesis holds and the value of the predicted ship state assuming it doesn't hold. Examples: operational utility of action "*investigate compartment for fire*" is low since it doesn't improve state of the ship, however, its goal utility is high since a [potential] fire is an important issue. On the other hand, action "*fight fire*" will have both high operational and goal utilities.

In our experiments, we have limited the lookahead to 10 minutes. In other words, anything predicted further into the future would be ignored. The output of the TIPN was fed into a state evaluator. The single top-ranked action was carried out.

Following the design process outlined in the previous section we had first handcoded an TIPN to model fire and flood related phenomena that take place aboard a DDG-51 class destroyer. We had then used Minerva-4 equipped with a handcoded rule-based scheduler [4] to run damage control scenarios automatically and collect the

training data. The training data was used to generate a decision rule board evaluator using C5.0 package [20,21]. The scenarios were run within the DC-Train ship damage control simulator used at the Surface Warfare Officer School (SWOS) in Newport, Rhode Island.

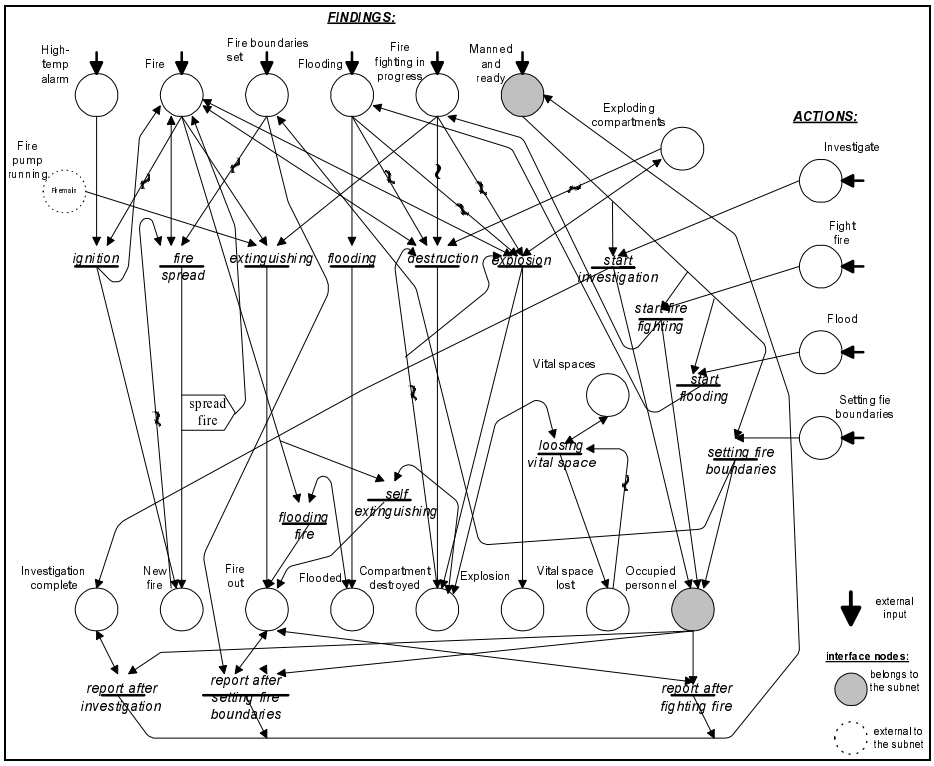


**Fig. 5.** A fragment of Minerva-DCA's Time Interval Petri Net modeling fire-fighting activities including physical phenomena and personnel effort

### Hardware and Software Implementation Details

Minerva-DCA is implemented in LPA Prolog and runs under Windows NT. An ODBC interface integrates it into the DC-Train simulated environment which uses Microsoft Access databases to represent the state of the ship and other dynamic and static domain information. The graphical user interfaces are implemented in Visual C++ and run as separate tasks under Windows NT. The graphical user interfaces and Minerva-DCA exchange information through the ODBC interface and the dedicated Access files.

In our tests Minerva-DCA has been running at 100-800 blackboard cycles per second speed on a dual-Pentium II-400MHz workstation.

### DC-Train 1.0 Runs

To evaluate the performance approximately 500 scenarios have been run within the DC-Train ship damage control simulator routinely used at the Navy's Surface Warfare Officers School (SWOS) in Newport, Rhode Island. We have then compared Minerva-DCA with an TIPN scheduler to Minerva-4 with a *handcoded rule-based scheduler* [4] and to Navy officers at SWOS. The results are shown in Figure 7.

Any scenario could have had one of the three outcomes: "ship lost", meaning that a major disaster such as a missile compartment explosion has been reached; "ship possibly saved", meaning that at 25 minutes scenario time the ship was still alive yet there were active crises; and "ship saved", meaning that there were no active crises at the 25-minute mark.

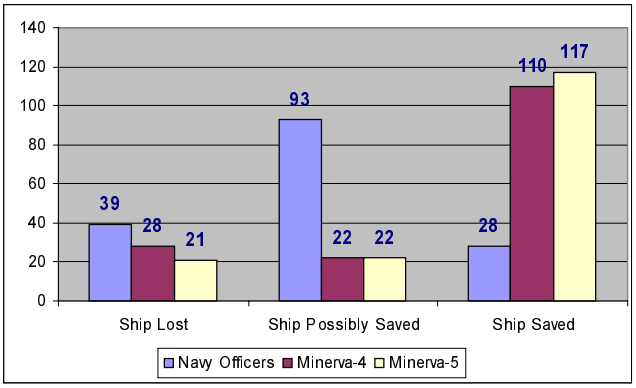


**Fig. 6.** Minerva-DCA vs. Minerva-4 vs. Navy officers experimental comparison

The use of the TIPN based scheduler resulted in 21 ships being lost. This is a 25% improvement over the use of a handcoded rule-based scheduler wherein 28 ships were lost. And, it is a 46% improvement over Navy officers where 39 ships were lost.

### Application Use and Pay-Off

A proof-of-principle prototype, called DC-Train 1.0, was first used at the Navy officer training school in Newport, Rhode Island, in March 1997. A refinement of this system, called DC-Train 2.0, was permanently deployed at SWOS in December 1998. It is to be regularly used in the six-week damage control course, which has approximately 30-50 Navy officers every six weeks. It provides the damage control course with its first tool that can generate arbitrarily many damage control scenarios and thereby provide extensive and immersive whole-task training.

To date, the controlled evaluation has been limited to each person solving four simulated scenarios at most, and has not involved giving the subjects feedback using the critiquing and advising facilities of Minerva-DCA. The initial limitations on the numbers of scenarios solved and the withholding of automated feedback are necessary to establish a performance baseline. Experts have, however, validated the accuracy of the feedback generated by the critiquing and advising facilities of Minerva-DCA

The primary benefits of the deployed immersive simulator and trainer described in this paper include (1) decreased load on the damage control course instructors, (2) the opportunity for the students to get a comprehensive feedback on their performance anytime, and (3) uniform and standardized scoring with a graphical feedback. This is in contrast to the current practice, where an instructor is always present when a student solves a scenario, which places a very large time-demand on the instructors.

## Related Work in Scheduling

### Model-based Approaches

Three alternative model-based approaches that we considered are dynamic programming, numerical simulation, and quantitative reasoning. We will briefly discuss each of these in turn.

- A *dynamic programming* formulation of a scheduling problem explicitly represents all the states, and can typically manage problems with in the order of 50,000 states [7, 16, 24]. This method does not appear to be suitable for the blackboard scheduling problem associated with the ship damage control decision making task because there are on the order of $10^{1000}$ states in our current test-bed system [5].

- *Numerical simulation* methods [10] can explicitly model an environment by simulating it from first-principles. Minerva-DCA works in conjunction with a numerical simulator for training the decision making system, but it can't be used for scheduling because it is approximately three orders of magnitude too slow.

- *Qualitative reasoning* methods [12] are another major method of explicitly modeling a complex environment. Their major strength is the ability to simulate a domain qualitatively. Time Interval Petri Nets have been found to be useful because the ship damage control domain requires simulation of a lot of quantitative and qualitative simulation. And the analysis tools that are possible because of the strong mathematical foundation of Petri Nets, such as reachability analysis and deadlock analysis, led us to explore the use of Petri Nets.

## Summary

This paper has presented the formalism of Time Interval Petri Nets that was created for blackboard scheduling. An TIPN-based scheduler should be usable as an environment model in a variety of systems. Combined with a state evaluator, it provides both envisioned states and associated rewards. The other main contribution presented here is the deployed application, Minerva-DCA, which was able to achieve a considerable level of expertise in the proof-of-principle domain of ship damage control.

The proposed approach has a number of strengths including high prediction speed, variable lookahead time, and well-studied formalism of Petri Nets. It also opens a number of directions for future research as follows.

## Future Research Directions

Now that Petri Nets have been indicated to be a useful formalism for blackboard scheduling, an area of future research is to further explore their relationship to network representations that have been more heavily used in AI, and to see the extent that the benefits can be combined or lead to cross-fertilization.

One particular direction that we are currently exploring is automated synthesis of the scheduler knowledge layer. Previous versions of the Minerva expert system used a rule-based scheduler allowing for similarity learning methods [8]. Minerva-DCA uses a TIPN-based scheduler, and currently the Time Interval Petri Networks are manually designed. While being a reasonable approach in some domains, handcoding might be too costly and error-prone in other domains. This branch of research can take advantage of machine learning methods to automatically extract the training data and generate the TIPNs based on it.

## Acknowledgements

Sebastian Magda has done most of the TIPN coding and has contributed several interesting suggestions. Discussion with KBS members and Dr. Valeriy K. Bulitko was definitely of help. The research has been supported in part by ONR Grant N00014-95-1-0749, ARL Grant DAAL01-96-2-0003, and NRL Contract N00014-97-C-2061.

## References

1. Van der Aalst, W.M.P. 1993. Interval Timed Coloured Petri Nets and their Analysis. In M. Ajmone Marsan editor, Application and Theory of Petri Nets, volume 691 of Lecture Notes in Computer Science, pp. 453 - 472. Springer-Verlag, Berlin.
2. Van der Aalst, W.M.P. 1994. Putting Petri nets to work in industry. Computers in Industry, 25(1):45--54.
3. Bulitko, V. 1998a. Minerva-DCA: A Multifunctional Dynamic Expert System. MS Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign.
4. Bulitko, V.; Wilkins, D.C. 1998b. Minerva: A Blackboard Expert System for Real-Time Problem-Solving and Critiquing. Tech. Report UIUC-BI-KBS-98-003, University of Illinois at Urbana-Champaign.
5. Bulitko, V.; Wilkins. D.C. 1999. Automated Instructor Assistant for Ship Damage Control. In the Proceedings of The Eleventh Innovative Applications of Artificial Intelligence Conference. (accepted for publication).
6. Carver, N.; Lesser, V. 1994. The Evolution of Blackboard Control Architectures. In Expert Systems with Applications--Special Issue on the Blackboard Paradigm and Its Application, Volume 7, Number 1, pp. 1-30, Liebowitz. New York, Pergamon Press.
7. Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In Proceedings of ICML'98 conference.
8. Donoho, S. 1993. Similarity-based learning for recursive heuristic classification. Master's thesis. University of Illinois at Urbana-Champaign.
9. Haykin, S. 1994. Neural Networks: A Comprehensive Foundation. Macmillan College Publishing Company.
10. Heath, M. 1996. Scientific Computing: An Introductory Survey. McGraw Hill Text.
11. Jensen, K. 1997. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Monographs in Theoretical Computer Science, Springer-Verlag.
12. Kuipers, B. 1994. Qualitative Reasoning. The MIT Press.

13. Larsson, E.; Hayes-Roth, B.; Gaba, D. 1996. Guardian: Final Evaluation. Knowledge Systems Lab, Stanford University. TechReport KSL-96-25.
14. Merlin, P. 1974. A Study of Recoverability of Computer Systems. Ph.D. Thesis. University of California, Irvine.
15. Merlin, P.; Faber, D.J. 1976. Recoverability of Communication Protocols. IEEE Transactions on Communication, 24, pp. 1036-1043.
16. Mitchell, T. 1997. Machine Learning. WCB/McGraw-Hill.
17. Murata, T. 1989. Petri Nets: Properties, Analysis, and Applications, in the proceedings of the IEEE 77, 541-580.
18. Najem, Z.H. 1993. A Hierarchical Representation of Control Knowledge For A Heuristic Classification Shell. Ph.D. Thesis. Department of Computer Science. University of Illinois at Urbana-Champaign.
19. Nii, H.P. 1989. Blackboard Systems. in Barr, A.; Cohen, P.R.; Feigenbaum, E.A. eds. 1989. The Handbook of Artificial Intelligence. Volume IV, Chapter XVI. Addison-Wesley.
20. Quinlan, J.R. 1986. Induction of Decision Trees. Machine Learning. 1(1), 81-106.
21. Quinlan, J.R. 1993. C4.5: Programs for Machine Learning, San Mateo, California: Morgan Kaufmann.
22. Park, Y.T; Tan, K.W.; Wilkins, D.C. 1991. Minerva 3.0: A Knowledge-based Expert System Shell with Declarative Representation and Flexible Control. UIUC, Department of Computer Science.
23. Peterson, J.L. 1981. Petri Nets Theory and Modeling of Systems. Prentice-Hall, Inc.
24. Russell, S.; Norvig, P. 1995. Artificial Intelligence: A Modern Approach. Prentice-Hall, Inc.
25. Wilkins, D.C.; Sniezek, J.A. 1997. An Approach to Automated Situation Awareness for Ship Damage Control. KBS Tech. Report UIUC-BI-KBS-97-012. Beckman Institute. University of Illinois at Urbana-Champaign.