Using Petri Nets to Represent Context in Blackboard Scheduling

Vadim V. Bulitko & David C. Wilkins
Beckman Institute
University of Illinois at Urbana-Champaign
405 N. Matthews Ave.
Urbana, IL 61801
bulitko@uiuc.edu & dcw@uiuc.edu

Keywords: Blackboard Scheduling, Modeling, Petri Nets in Artificial Intelligence, Decision Support, Intelligent Tutoring, Temporal Reasoning

Abstract[1]

The blackboard approach has been successfully used in a variety of domains. A typical problem-solving cycle consists of deliberation, scheduling, and execution steps. This paper presents a scheduling approach that uses the mathematically well-founded framework of Petri Nets as an environment model to represent the scheduling context. The paper extends the classical Petri Nets model in various ways and shows their decision making and tutoring applications in the test-bed domain of ship damage control. The use of a scheduler based on Extended Petri Nets resulted in 21 of 160 ships being lost in an immersive simulated environment. This is a 46% improvement over subject experts where 39 ships were lost.

Introduction

A typical problem-solving cycle within the blackboard framework [Larsson&Hayes-Roth96, Carver&Lesser94, Park91, Nii89, Bulitko98b] consists of the following three steps: 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 reflected the new state of the environment. The cycle is then repeated [Bulitko98b, Najem93]. This paper focuses on the scheduling step. A reader interested in the deliberation step and its suitability for the scheduling approach can be referred to [Bulitko98b, Bulitko98a].

In this paper we propose a model-based scheduling approach. The scheduler represents the context of scheduling via an environment model implemented with Extended Petri Nets (EPNs). The reward values come from a static state evaluator that evaluates EPN-predicted states.

Previous model-based scheduling approaches include: decision-theoretic dynamic programming [Russell95], qualitative reasoning [Kuipers94], and numerical simulation [Heath96].

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

1) EPNs have good representational and reasoning power for domains with complex causality and time-structure function models. They are capable of representing context (i.e. variables), timing information, and procedural knowledge. The experiments described in this paper suggest that EPNs scale up to practically useful domains.

2) EPNs allow for explanation of their operation. An explanatory facility is particularly valuable in human-computer interactive decision support and intelligent tutoring.

3) The Petri Nets community has developed a host of powerful theoretical methods for Petri Nets analysis. Methods relevant to our refinement of the EPN domain model include the automatic graph reachability analysis and deadlock analysis. Many of those methods are available as software packages developed by the Petri Net community. The ability to analyze Petri Nets facilitates debugging of the EPN models used for scheduling, their optimization, and extensions.

Overall Design

At a high level the main idea of our 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 [Winston84]. 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. Figure 1 presents a high level

operation of the Minerva-5 expert system that uses this approach. Further details on Minerva-5 can be found in [Bulitko98a].

Our approach is model-based [Russell95]. That is, unlike Q-learning an explicit model of the environment is used. An 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 an Extended Petri Network (EPN). The EPN itself does not contain any reward information. The reward values come from a static state evaluator [Bulitko98a].

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 2). 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, flood, etc. The output is severity of the ship state and is proportional to the time till a predicted major disaster ("time till kill point") [Bulitko98a]. Artificial neural networks [Haykin94] and decision trees [Quinlan86, Quinlan93] are used as the internals of the "black box". Both have shown comparable performance on the data sets that were used.

Extended Petri Nets

Petri Nets is a formalism proposed by C. A. Petri with some early applications to modeling communication protocols and distributed software systems [Murata89]. Petri Nets have traditionally been viewed as being best suited for performance evaluation and communication protocol analysis in the areas of distributed software, database, and communication systems, programmable logic and VLSI design, formal languages, and logic programs [Murata89, Peterson81]. Later on higher-order Petri Nets extended with color and time information have been developed and applied to modeling and analysis of complex real-world systems such as production plant operations [vanderAalst94, vanderAalst93, Jensen92, Merlin74, Merlin76]. Attractive properties of Petri Nets include the solid and well developed mathematical foundation, a number of theoretical and practical analysis methods, and a great variety of supporting software packages. Petri Nets continue to be widely used and there are several periodic conference exclusively devoted to their development [e.g. ICATPN-99].

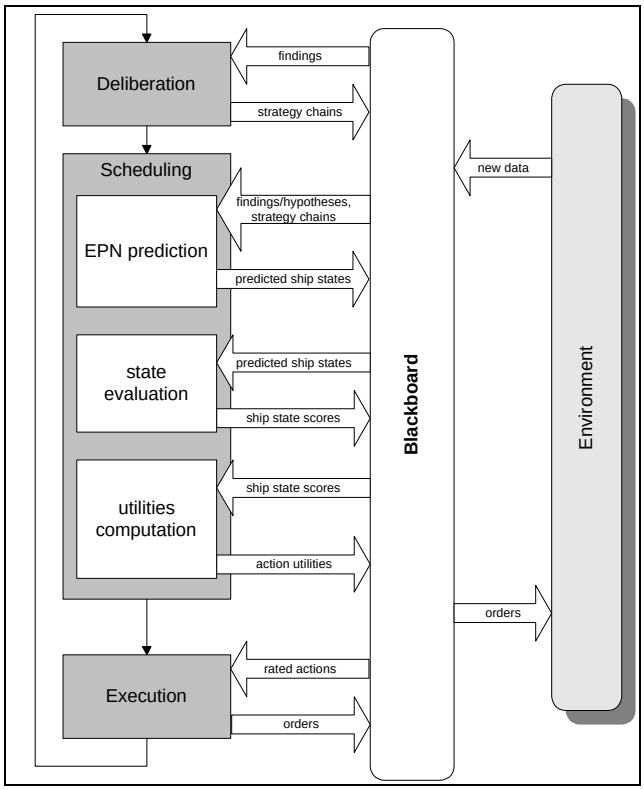

Figure 1. Minerva-5 Operation: the blackboard framework and problem-solving cycle stages

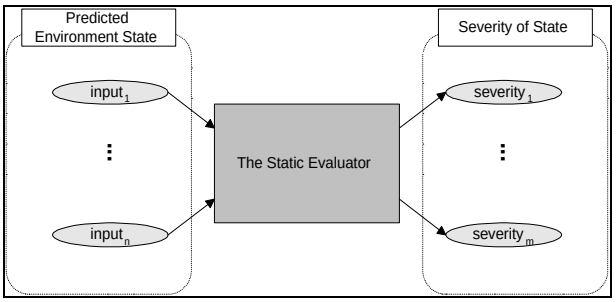

Figure 2. Static state evaluator as a black-box with state attributes as inputs and state severity values as outputs

This paper proposes a set of extensions for making Petri Nets suitable for scheduling in Artificial Intelligence (AI) systems. It also compares the proposed extensions to related research with applications in "non-AI" domains.

Classical Petri Nets

This section will briefly present the classical Petri Nets (also known as place-transition or PT-nets) following the notation of [Peterson81]. A Petri Net (PN) C is a tuple $C=<P,T,I,O>$ where P is a set of places, T is a set of transitions, I is a set of transition inputs, and O is a set of transition outputs. A marking of Petri Net C is denoted by

m and a marked Petri Net is referred to as $M=<P,T,I,O,m>$. A marking is an n-vector of natural numbers where n is the number of places ($n=|P|$).

An example will demonstrate the workings of Petri Nets. Consider a very simple Petri Net "Firefighting" shown in Figure 3. The large circles represent places $P=\{$"*Compartment is engulfed*" (p_1), "*Firefighting in progress*" (p_2), and "*Fire is out*" (p_3)$\}$. There is a single transition $T=\{t_1\}$ (shown as a thick horizontal black line) that has p_1 and p_2 as its input (enabling) places, and p_3 as its output place. Accordingly $I=\{<t_1,p_1>, <t_1,p_2>\}$ and $O=\{<t_1,p_3>\}$. Sets I and O are shown graphically as arcs between the places and the transitions. As mentioned above a marking assigns tokens to the places. In the Figure 3 tokens are represented as black dots inside the places. $<1,1,0>$ marking is shown.

A Petri Net marking represents a snapshot of a system being modeled by the net. This example models the process of fighting fire. The marking shown represents the situation of the compartment being engulfed and fire being fought. The transitions represent state changes. In the example the single transition expresses the rule that if the compartment is engulfed and the fire is being fought the fire will be put out. A transition fires when all of its enabling places have the appropriate number of tokens in them. Each arc leading from a place to a transition requires a presence of one token in the originating place. Firing a transition removes all the enabling tokens from the corresponding enabling places and adds tokens to the output place. In our example, firing transition t_1 would result in marking $<0,0,1>$ corresponding to empty places p_1, p_2 and place p_3 having one token (i.e. the fire is out).

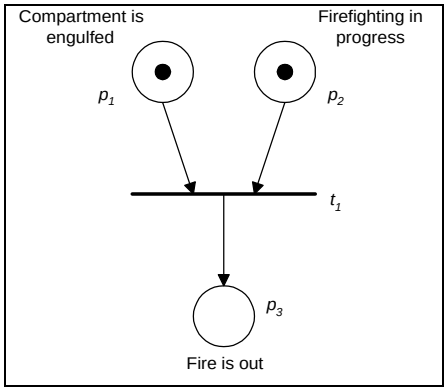

Figure 3. A Classical (PT) Petri Net modeling extinguishing a fire

While being conceptually simple, Petri Nets can be used to model a wide spectrum of systems (e.g. multi-agent systems sharing resources, operating systems, etc.) [Peterson81]. A number of Petri Nets-related theoretical results have also been developed [Peterson81].

The Petri Net Extensions

While being a powerful and attractive modeling tool, the classical Petri Nets lack several features helpful for modeling complex real-time systems. The five limitations of the classical Petri Nets the paper addresses are as follows:

- Classical PNs are essentially *propositional* (i.e. they have no variables or functions). Therefore the modeled system is described in an attribute-value way. To model a large system containing many similar components would require replication of a number of tokens. For example: if to extend the toy PN described above to model more than one compartment (say *N similar* compartments), would require replication of the *same* network *N* times. Clearly such redundancy will have adverse effects on PN performance in terms of speed and space.
- Classical PNs represent evolution of a modeled system as a sequence of states with *instantaneous* transitions. So in a way, the dynamics are modeled by means of *situational calculus* [Russell95]. While such representation is adequate for certain models, there are systems that require explicit time and duration representation. In our toy PN example putting a fire out takes no time at all. In reality the information about duration of various transitions (such as putting out a fire) is often critical. An alternative formalism, *event calculus* [Russell95], represents events as entities with explicit duration, beginning, and ending times.
- There is *no uncertainty support* in classical PNs. The entire state of the world is assumed to be specified exactly. This is an unrealistic assumption in many real-world domains.
- All the enabling places are *positive* in the sense that the transition firing requires tokens to be there. It is trickier to express a rule that has negation in it (e.g. *No* "Firefighting" and "Engulfment" result in "Fire Spread"). In fact, PT-nets are incapable of zero-testing in the general case [Peterson81]. That fact reduces their modeling power and causes application inconveniences.
- Every time a transition fires all the enabling tokens are *withdrawn* from the enabling places. This doesn't always correspond to the actual system behavior where an attribute might result in a transition firing yet the value of that attribute remains unchanged (e.g. the place "Fire" should not loose its tokens when transition "Fire Spread" fires). In classical PNs extra arcs would be needed to just put withdrawn tokens back in.

To address those concerns several extensions are made to the classical PN model. The new formalism is hence called Extended Petri Nets (EPNs). The extensions are then illustrated with another simple example. The new toy example (Figure 4) introduces an EPN modeling fire

spread. Related Petri Net extensions by other researchers will be considered as the paper proceeds.

- 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 {<Room, 3-370-0-E>, <Station, "Repair Locker 5">} in place "Firefighting in progress" is interpreted as "Repair Locker 5 fire fighters are fighting fire in room 3-370-0-E". This extension addresses the propositionality and lack of variables/context issue of the classical PNs. 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 [van der Aalst93, Jensen92]. Our approach is different in that we (1) explicitly specify the type of each value in the identifier and (2) do not require the colours of a token belong to the colour set of the place the token is in.

- 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: token with domain identifier {<Room, 3-78-0-M>}, time interval [3:12, 3:44] in place "Compartment is engulfed" represents our belief that compartment 3-78-0-M *became* engulfed sometime between 3min 12sec and 3min 44sec of the scenario time. Of course, 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 way are known as *time* or *timed Petri Nets* [e.g. van der Aalst93, van der Aalst94]. 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).

- 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 Δ_t is a random variable then $[\Delta_{min},\Delta_{max}]$ is the *N*% confidence interval for Δ_t (again *N* is close to 100%). This extension brings us closer to the event calculus as opposed to the situational calculus. Again there are other researchers who have used time intervals for the transitions. Merlin in [Merlin74, Merlin76] has used time intervals to represent minimal and maximal enabling times, however the Petri Nets themselves were not coloured. van der Aalst in [van der Aalst93] 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).

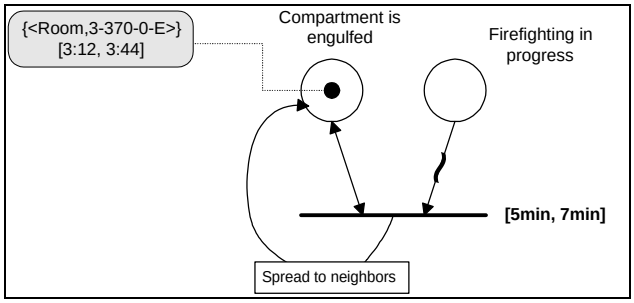

Figure 4. Extended Petri Net modeling fire spread

- New types of arcs are introduced. Those include:
  - NOT-arcs (negation or inhibitory arcs) mandate that a transition the arc leads to will not fire if there is a corresponding token in the originating place (e.g. the arc between place "Firefighting in progress" and the transition in Figure 4);
  - double-ended arcs do not withdraw tokens from their enabling places (e.g. the arc between "Compartment is engulfed" and the transition in Figure 4);
  - operator-arcs have arbitrary operators associated with them. In Figure 4 the arc coming out of the transition creates multiple tokens corresponding to the compartment's neighbors). Related work has been reported in [Jensen92].

EPN Operation

In this section we step through a simple example showing an EPN in action. We will start with the an EPN modeling ignition. Figure 5 presents the initial configuration where place "Flammable Materials" is filled with two tokens labeled with compartment (room) names. 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.

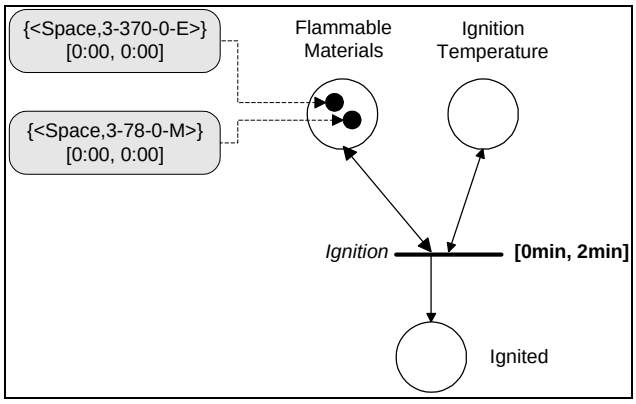

Figure 5. Step 0: the initial configuration

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 got hot sometime between 2:37 and 2:45 scenario time. This news has no effect on the subnet since the passageway doesn't contain flammable materials (as far as we are aware).

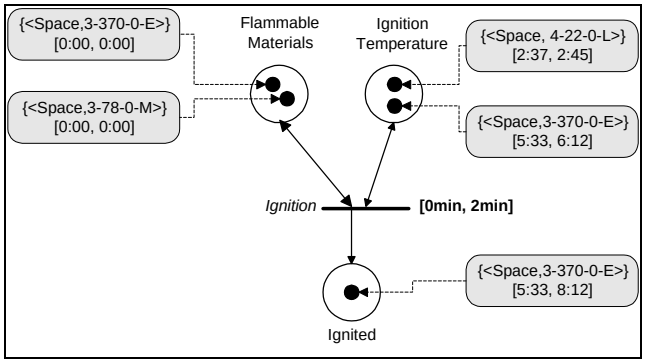

Figure 6. Step 2: generator room gets ignited

In Step 2 the generator room 3-370-0-E gets 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 6).

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 got affected by the delay interval of the transition.

Designing a EPN-based scheduler

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

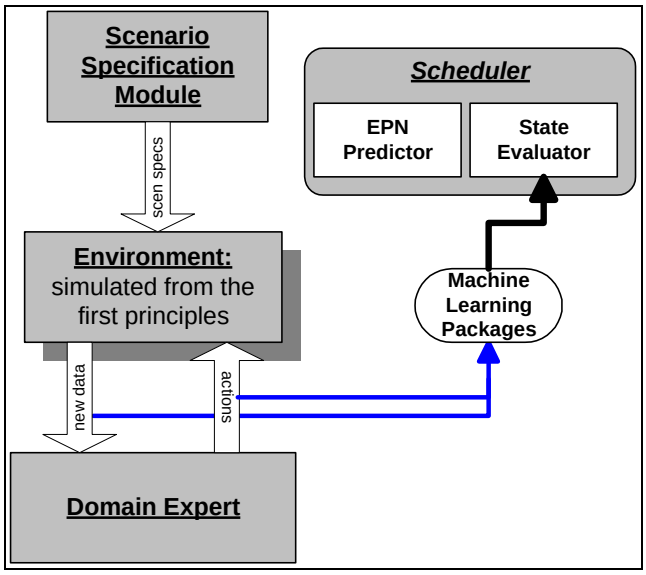

Figure 7. Designing a state evaluator: a number of scenarios are being run with a "teacher" (i.e. a human domain expert or another expert system) and the environment states and the agent's actions are being logged and used to train the static board evaluator.

The first step is to construct an EPN that represents the physical causality and timings in the domain. The places would typically correspond to states of the environment and the transitions would 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 hand eliciting knowledge from a domain expert. Automating this step is an area of current research.

The second step is to design a state evaluator to work with the EPN 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 7. 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 could be used as the training samples to synthesize the ANN or a decision tree. Commercial packages could be used for this substep.

Once trained the system could be used as described in the previous section and outlined in Figure 1. As demonstrated in the experimental evaluation section the system trained in

this manner can indeed outperform its teacher.

Experimental Evaluation

Implementation in Minerva-5

The EPN framework has been tested as a part of Minerva-5 blackboard expert system that has been written for the ship damage control domain [Wilkins97]. Below is a description of the EPN implementation within Minerva-5 while the most detailed treatment is given in [Bulitko98a].

The deliberation module of Minerva-5 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-5 has a handcoded EPN part of which is shown in Figure 8. Each deliberated action is fed into the EPN 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.

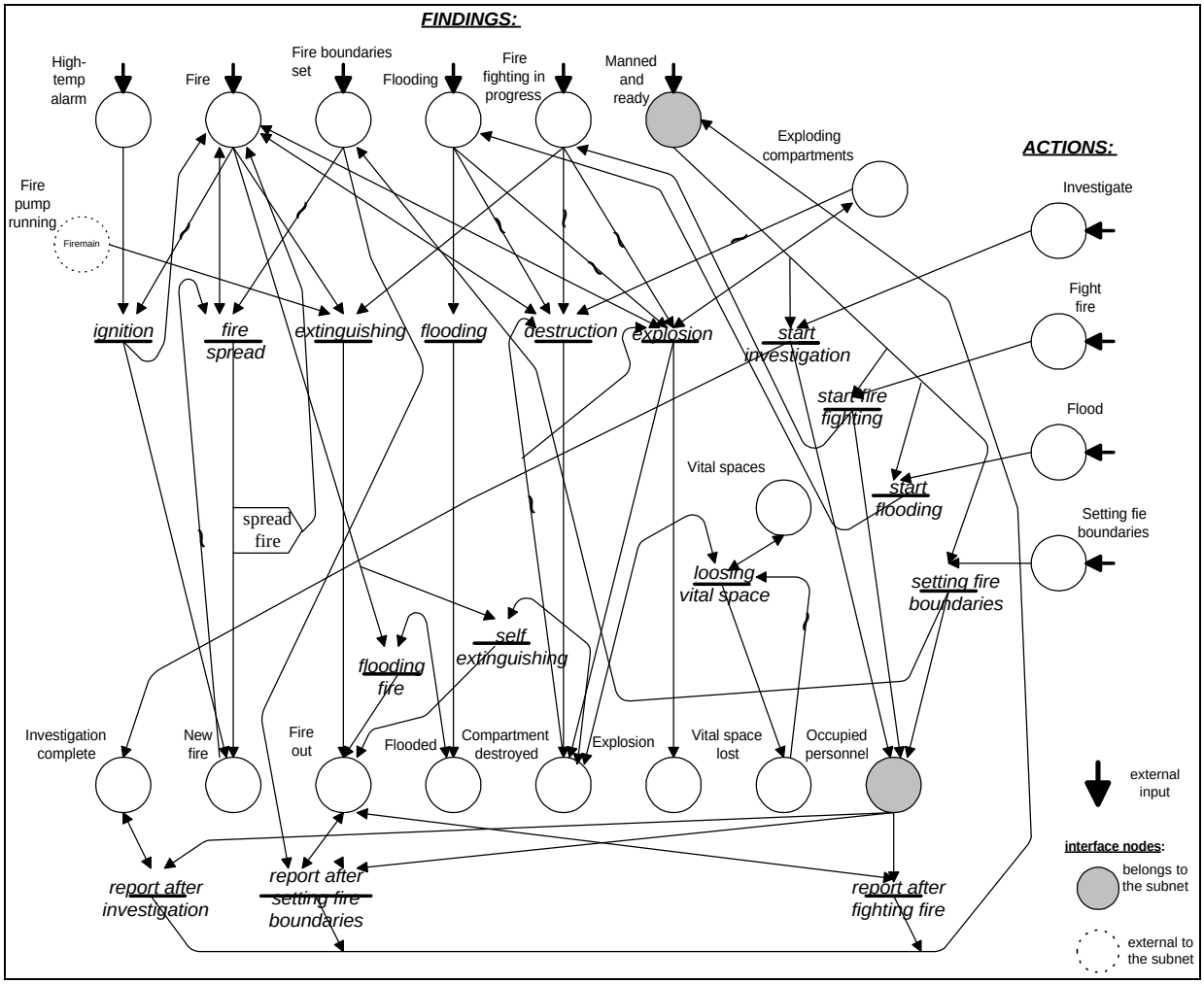

Figure 8. A fragment of Minerva-5's Extended Petri Net modeling fire-fighting activities including physical phenomena and personnel effort

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, it is goal utility is high since a [potential] fire is an important issue. On the other hand, action "*fight fire*" will have both operational and goal

utilities high.

We limited the lookahead to 10 minutes. In other words, anything predicted further into the future would be ignored. The output of the EPN 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 EPN 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 [Bulitko98b] 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. The scenarios were run within the DC-Train ship damage control simulator used at the Surface Warfare Officer School (SWOS) in Newport, R. I.

DC-Train 1.0 Runs. To evaluate the performance 500 scenarios have been run within the DC-Train ship damage control simulator routinely we developed that is used at the Navy officers school in Newport, R. I. We have then compared Minerva-5 using the EPN scheduler to Minerva-4 that had a handcoded *context-free* rule-based scheduler [Bulitko98b] and to Navy officers at SWOS. The results are presented in Figure 9.

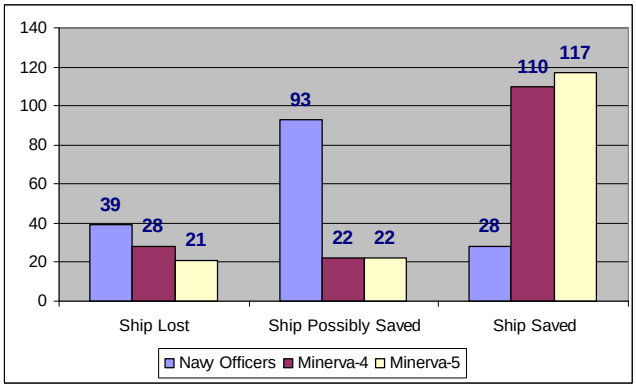

Figure 9. Minerva-5 (EPN-based scheduler) vs. Minerva-4 (context-free rule-based scheduler) vs. Navy officers experimental comparison

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

The use of the EPN 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.

Applications

In our research we have investigated applications of Extended Petri Networks to human-computer collaborative decision-making systems and intelligent tutoring systems. While a detailed discussion could be found elsewhere [Bulitko99a], we will present the highlights below.

Problem-solving. EPNs could be used for problem-solving as a part of a blackboard based decision-making system. The previous section has described one such deployed application, Minerva-5. The attained problem-solving expertise could be used as the basis for the following functions.

Advising facilities allow an expert system, such as Minerva-5, to explain its motivation for the actions taken. Both blackboard deliberation and EPNs allow for natural language output. Details could be found in [Bulitko98a]. Minerva-5 is equipped with a graphical user interface (GUI) presenting the user with graphical and natural language output [Bulitko99b].

Critiquing facilities use differential analysis as the basis. Creative user's actions that don't have a match in the expert system line of reasoning are fed into the EPN predictor for evaluation. Again, Minerva-5 is equipped with a critiquing GUI [Bulitko99a, Bulitko99b, Bulitko98a]. Critiquing naturally relates to performance scoring insomuch as every user action could be assigned an explainable score. A scoring package has been developed for Minerva-5 and deployed at a Navy training school at Newport, R.I.

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 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 [Dietterich98, Mitchell97, Russell95]. This method is not 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.

Numerical simulation methods [Heath96] can explicitly model an environment by simulating it from first-

principles. Minerva-5 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 [Kuipers94] are another major method of explicitly modeling a complex environment. Their major strength is the ability to simulate a domain qualitatively. Extended Petri Nets have been found to be useful because the ship damage control domain requires simulation of a lot of quantitative and qualitative processes. 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.

Discussion

This paper has presented the formalism of Extended Petri Nets that was created for the Blackboard scheduling domain. An EPN-based scheduler should be able to be used as an environment model in a variety of systems. Combined with a state evaluator it provides both envisioned states and associated rewards. Another main contribution of the research is applying the approach in a proof-of-the-principle domain of ship damage control. The system has been deployed and routinely used at a Navy training school for automated intelligent tutoring.

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 shown to be a useful formalism for Blackboard scheduling, an area of future research is to more deeply 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 Learning EPNs. Currently the Extended Petri Networks are manually designed. While being a reasonable approach in some domains, handcoding might be too costly in other domains. The consistency might become an issue as well. This branch of research can take advantage of machine learning methods to automatically extract the training data and generate the EPNs based on it.

Acknowledgements

Sebastian Magda has done most of the EPN 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

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.

van der Aalst, W.M.P. 1994. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45--54.

Bulitko, V. 1998a. Minerva-5: A Multifunctional Dynamic Expert System. MS Thesis. Department of Computer Science, University of Illinois at Urbana-Champaign.

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.

Bulitko, V.; Wilkins. D.C. 1999a. Automated Instructor Assistant for Ship Damage Control. In the Proceedings of The Eleventh IAAI Conference (accepted for publication).

Bulitko, V.; Wilkins. D.C. 1999b. Minerva-DCA: An Intelligent Agent for Ship Damage Control. Proceedings of the PACLP'99 conference (accepted for publication).

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.

Dietterich, T. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In Proceedings of ICML'98 conference.

Donoho, S. 1993. Similarity-based learning for recursive heuristic classification. Master's thesis. University of Illinois at Urbana-Champaign.

Haykin, S. 1994. *Neural Networks: A Comprehensive Foundation*. Macmillan College Publishing Company.

Heath, M. 1996. *Scientific Computing: An Introductory Survey*. McGraw Hill Text.

Jensen, K. 1992. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag.

Kuipers, B. 1994. *Qualitative Reasoning*. The MIT Press.

Larsson, E.; Hayes-Roth, B.; Gaba, D. 1996. Guardian: Final Evaluation. Knowledge Systems Lab, Stanford University. TechReport KSL-96-25.

Merlin, P. 1974. A Study of Recoverability of Computer Systems. Ph.D. Thesis. University of California, Irvine.

Merlin, P.; Faber, D.J. 1976. Recoverability of Communication Protocols. *IEEE Transactions on Communication*, 24, pp. 1036-1043.

Mitchell, T. 1997. *Machine Learning*. WCB/McGraw-Hill.

Murata, T. 1989. Petri Nets: Properties, Analysis, and Applications, in the proceedings of the IEEE 77, 541-580.

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.

Barr, A.; Cohen, P.R.; Feigenbaum, E.A. eds. 1989. *The Handbook of Artificial Intelligence*. Volume IV, Chapter XVI by Nii, H.P. Addison-Wesley.

Quinlan, J.R. 1986. Induction of Decision Trees. *Machine Learning*. 1(1), 81-106.
Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning*, San Mateo, California: Morgan Kaufmann.

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.

Peterson, J.L. 1981. *Petri Nets Theory and Modeling of Systems*. Prentice-Hall, Inc.

Russell, S.; Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc.

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.

Winston, P.H. 1984. *Artificial Intelligence*. The second edition. Addison-Wesley Publishing Company.

The International Conference on Application and Theory of Petri Nets. The 20th annual meeting will be held in June 1999.