A COLORED PETRI NET MODEL
OF TACTICAL DECISION MAKING*

by

Zhuo Lu **
Alexander H. Levis**

ABSTRACT

* This work was supported by the Office of Naval Research under contract no. N00014-90-J-1680.

** The authors are with the C3I Center, George Mason University, Fairfax, VA 22030 USA
INTRODUCTION

One of the long range goals of research on distributed tactical decision making (DTDM) is to develop analytical models of the distributed decision making process that can be used (a) for designing model driven experiments, and (b) for analyzing the performance of alternative organizational designs. The feasibility of the former was demonstrated by Jin and Levis [1]. In this paper, the effort is focused on the latter, performance analysis. The objective is to develop executable models that represent more realistic decision processes. First, they have to be sufficiently detailed to correspond to the decision processes embedded in experiments with human subjects. Second, they must be able to incorporate theories of human decision making under stress and in the presence of uncertainty. [2] The models used this far, [3], based on ordinary Petri Nets and on modest extensions, have proved to be quite useful but are rather limited in their ability to represent complex cognitive tasks and to depict a whole range of interactions among decision makers in a team. [1] In order to address decision making under stress, it is not possible any more to make the assumptions and simplifications found in earlier models.

The basic four stage role model described in [3] has been recast using the Colored Petri Net formalism [4]. In Colored Petri Nets (CPN) the tokens have attributes; the attributes can be represented by a vector where each component of the vector can take values from a set. The places of the Net have Color Sets associated with them; they define the types of tokens that can reside in the place. The transitions of the net have Occurrence Colors associated with them; they are used to define the enabling conditions. The arcs or connectors of the Petri Net are annotated with Boolean expressions that specify the conditions under which the relationship between the place and the transition exists. The arc annotations define the enabling conditions for the corresponding transitions. In addition, guard functions can be associated with the transitions; they can be thought of as additional conditions for enabling of a transition. When a transition fires, the tokens specified by the arc expressions are removed from the input places and the tokens specified by the arc expressions on the arcs between the transitions and the output places are generated in the output places. The result of these generalizations is that very complicated operating procedures can be modeled in a compact form.

The modeling of distributed decision making using Colored Petri Nets will be illustrated by a naval example in which two DMs are conducting a coordinated outer air battle. The model has been implemented using Design/CPN™ [5], a Colored Petri Net editor and simulator. This is the mathematical model of the organization that forms the basis of an experiment being carried out at George Mason University. [2] The simulation model is used to obtain two measures of performance – accuracy and timeliness – in support of the experiment. It is also used to investigate ways of modeling coordination mechanisms in decision making teams.
The results of the simulation have been used to design the experiment in accordance with the steps of the model-driven experimental procedure [1].

THE OUTER AIR BATTLE

The objective of the Naval Outer Air Battle, as abstracted for this example, is to monitor incoming threats so as to intercept the enemy between the radar visible range (the outer circle in Figure 1) and the weapons release range (the line from which missiles can be fired at ships in the battle group – the inner circle in Figure 1). In this environment, two human decision makers (DMs) form the Anti-Air Warfare decision making team assigned to perform the above task. For a full description of the example, see [6]. In the present experiment, a few modifications have been made in the original experiment designed by Jin. [6]

![Figure 1 Defense area for Naval Outer Air Battle](image)

The whole battle area shown in Figure 1 has been divided into three equal sectors as demarcated by the solid lines. Only two of the sectors are active with one DM in charge of each sector. However, each DM can monitor threats in a larger sector; part of his monitoring sector overlaps with the other
DM’s sector of responsibility. The whole active area, excluding the third sector, consists of four separate sectors denoted by r1, r2, r3, and r4 (Figure 1). The sector r1 is the area that only DM1 can monitor and influence; r2 is the area DM1 can monitor and influence and DM2 can monitor but not influence (overlap area); r3 is the area DM2 can monitor and influence and DM1 can monitor but not influence (overlap area); and r4 is the area that only DM2 can either monitor or influence.

When a threat enters either r1 or r4, DM1 or DM2 deal with it individually. When a threat is in r2, DM2 may monitor the threat and communicate his assessment and a recommendation (denoted as “suggestion” in the model) to DM1. DM1 then has a choice of either accepting or ignoring the suggestion. Similarly, if the threat is in r3, DM2 may generate a suggestion and DM1 has the option of taking the suggestion of DM2 or ignoring it. This implies that communications may occur when threats appear in the overlap areas. These communications provide a means of coordination between the two team members.

All threats arrive at the outer boundary of the circular area simultaneously. Since the angles of the incoming threats (bearing) remain constant during a trial in the experiment, the bearing can be used as an identifier for a threat. All the rules for information processing by the DMs (who are assumed to be well trained for the task) are well defined.

When a DM moves the cursor and clicks on a threat in a sector that he can monitor, the following data associated with that threat appear in a window on the display:
- the incoming speed of the threat, \( v \);
- the number of aircraft in the threat, \( m \).

The speed \( v \) can take one of the values in the set \( VS \) which is defined as
\[
VS = \{300, 360, 450, 600, 720, 900, 1200\};
\]
while \( m \) can take one of the values in set \( MS \) which is defined as
\[
MS = \{2, 3, 4, 5, 6\}.
\]
These are the attributes of the threats, the ground truth. However, the DMs do not need to identify these attributes exactly. Two corresponding sets are introduced to describe the choices of the DMs:
- \( VCS = \{ v_c \in VCS \mid \text{slow, medium, fast} \} \);
- \( MCS = \{ m_c \in MCS \mid \text{less5, equal5, more5} \} \);
where 5 is considered the threshold value for the number of aircraft in a threat.

At the first stage, situation assessment, the DMs are expected to process the numerical data on speed and number of aircraft in a threat that appears on the screen as a result of probing and perform the mapping from \( VS \) and \( MS \) to \( VCS \) and \( MCS \), respectively,
\[
VS ----> VCS \quad (v_c = v_c(v));
MS ----> MCS \quad (m_c = m_c(m)).
\]
The correct mappings or rules is defined as shown in Table 1:
Table 1  Rule for classifying speed

<table>
<thead>
<tr>
<th>Speed  v</th>
<th>Speed Class v_c</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>360</td>
</tr>
<tr>
<td>450</td>
<td>600</td>
</tr>
<tr>
<td>720</td>
<td>900</td>
</tr>
</tbody>
</table>

If the size of a threat m is less than five aircraft, then $m_c$ takes the value “less5”; if m is five, then $m_c$ takes the value “equal5”, and if m is more than five, then $m_c$ maps to “more5”.

With this information, the DMs can determine at the second stage of information processing the type of each incoming threat. The set for the type of threat is defined as follows:

$$ZS = \{ z \in ZS \mid F(\text{fighter}), B(\text{bomber}), S(\text{surveillance}) \},$$

The DMs determine the type of the threat $z$ on the basis of $v_c$ and $m_c$, i.e., they perform the following mapping:

$$VCS \ast MCS ----> ZS \quad (z = z(v_c, m_c)).$$

The rules that govern the mapping are shown in Table 2:

Table 2  Rule for determining the type of a threat $z$ ($z = z(v_c, m_c)$)

<table>
<thead>
<tr>
<th>Size</th>
<th>Speed v_c</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Slow</td>
</tr>
<tr>
<td>less5</td>
<td>F</td>
</tr>
<tr>
<td>equal5</td>
<td>B</td>
</tr>
<tr>
<td>more5</td>
<td>B</td>
</tr>
</tbody>
</table>

When a threat is in an overlap area, the responsible DM may make his own decision $z$ before he receives a suggestion $z'$ from the other DM. If he receives the suggestion $z'$ from the other DM, he may accept it as is, he may reject the suggestion and keep his decision $z$ as the final decision, or he may fuse the suggestion $z'$ together with his own decision $z$ to arrive at the final decision $z''$. Note that all three variables $z$, $z'$, and $z''$ take values in $ZS$.

The rules the DM should follow to make the final decision $z''$ when a suggestion $z'$ is accepted are shown in Table 3.

Having established the information patterns between the decision makers and the prescriptive rules that should govern the decision making of each decision maker, attention is shifted to the construction of the Colored Petri Net model.
Table 3  Rule for making final decision $z''$ when the suggestion $z'$ is fused ($z''=z''(z,z')$)

<table>
<thead>
<tr>
<th>DM's Decision</th>
<th>Other DM's Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z$</td>
<td>$z'$</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>S</td>
<td>F</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

THE CPN MODEL OF THE TASK LAYER

One of the key modeling issues in the study of distributed decision making in teams is the representation of the coordination functions among team members. In earlier work, Andreadakis [7] defined the data flow structure of a command and control process and then, in allocating functions to resources, introduced connectivity necessary for the interaction among resources: communications functions, and monitoring and command interactions. That approach has led now to the concept of a two-layered representation of a team: The task layer and the coordination layer. In the former, which is roughly equivalent to Andreadakis’ data flow structure, the basic functionality needed to carry out the team’s decision making task and the corresponding information flows are depicted. In the latter, the pattern of interactions among team members that affect the manner in which the task is executed is depicted. In an ideal case, different coordination models could be applied to the same task model without the need to modify the task model. However, it is not clear yet whether such a clean decoupling is at all possible and under what conditions it would occur, if it were at all possible.

The information flow paths in the task layer are modelled as Colored Petri Nets. Transitions represent processes, places are characterized by the information or conditions they can hold, and tokens denote information or materials or conditions. The color of a token characterizes the information carried by the token.

Many logical relations are not explicit in the task layer model. For example, the logical relations “If there is a token with color C1 in place P, then transition T fires” and “If there is a token with color C2 in place P, then transition T fires” establish conditions under which some process can take place. Now suppose that a C1 token and a C2 token are present in P. A relation is needed to establish some priority between the two enablements. Define this kind of a relation as a coordination relation. Coordination relations (i.e., the functions that realize them) must be executed
first, before the task layer functions present at the same transition. Typical examples include the function of checking the state of a place or the realization of priority rules for queues of tokens in a place. The coordination layer contains the coordination functions that realize the logical coordination relations necessary to execute the operations defined in the task layer.

In this section, all the information flow paths of the organization are represented using the Colored Petri Net (CPN) formalism. The CPN model of the possible information flow paths for Decision Maker 1 (DM1) is shown in Figure 2; the CPN of Decision Maker 2 is identical since in the experiment the two decision makers play identical roles. The CPN model of information flow paths for the whole organization is shown in Figure 3. The overall model also contains the input data preparation stage, i.e., a CPN representation of the model that generates the threats for each trial.

FIGURE 2 - Task Layer CPN of DM1

FIGURE 3 - CPN of Task layer of team

Names of transitions and places in the model (Figure 2 and Figure 3) are represented in bold text. The arc expressions and the guard functions of transitions are shown in plain text. The color sets of places are shown in italics. The color sets and their variables used in the code segments of transitions, places and arc expressions in the model (Figures 2 and 3) are defined in global declaration node. Table 4 contains a list of the color sets and their variables.

Figure 2 shows the Colored Petri Nets model for DM1. A token in Input1 (the input place of DM1) represents a threat which in any of the three sectors that DM1 may monitor, namely, r1, r2, or r3. The color of the token is a 4-tuple (b, r, v, m) where:

- b is the bearing of the threat which can serve as an identifier of the threat;
- r is the sector where the threat is;
- v is the speed of the threat;
- m is the size of the threat.

The transition Div1 performs two tasks. The first task partitions the tokens according to their respective sector: When the attribute r = r1, the output token is generated in place over1; when r = r2, it is generated in over1; and if r = r3 in place suggestion1. The second task maps the speed and the size of the threat from numerical values to their classes according to two functions ppv(v) and ppm(m) defined in the global declaration node, Table 5; these functions implement the well-defined rules described in Tables 1 and 2. The bearing b remains unchanged.
Table 4  Color sets and variables used in the model

Bearing of a threat:
- Color set: BS={0, 1, 2, 3, ..., 240};
- variables: b, b2: BS;
Region of a threat:
- Color set: RS={r1, r2, r3, r4};
- variables: r: RS;
Speed of a threat:
- Color set: VS={v300, v360, v450, v600, v720, v900, v1200};
- variables: v: VS;
Size of a threat:
- Color set: MS={m2, m3, m4, m5, m6};
- variables: m: MS;
Speed class of a threat:
- Color set: VCS={slow, medium, fast};
- variables: vc: VCS;
Size class of a threat:
- Color set: MCS={less5, equal5, more5};
- variables: mc: MCS;
Type of threat:
- Color set: ZIS={S, B, F};
- variables: z, zb: ZIS;

Color sets for the places are defined as:
- Universal color set: E = { e };
- US=BS*RS*VS*MS;
- UCS=BS*RS*VCS*MCS;
- ZS=BS*ZIS;

Table 5  Functions defined in the CPN’s global declaration node

```
color TYPE = with un|ty1|ty2c|ty2n|ty3|ty2|ty3c|ty3n|ty4;
nvar ty,tyb,tty:TYPE;

color CS = with cc1|cc2|c1|c2|c3|c4|c5|c6|c7|c8;
nvar c:CS;
```
fun ppv v300 = slow
| ppv v360 = slow
| ppv v450 = medium
| ppv v600 = medium
| ppv v720 = fast
| ppv v900 = fast
| ppv v1200 = fast;

fun ppm m2 = less5
| ppm m3 = less5
| ppm m4 = less5
| ppm m5 = equal5
| ppm m6 = more5;

fun find_z (slow,less5)  = F
| find_z (slow,equal5)  = B
| find_z (slow,more5)  = B
| find_z (medium,less5) = S
| find_z (medium,equal5) = B
| find_z (medium,more5) = B
| find_z (fast,less5) = S
| find_z (fast,equal5) = F
| find_z (fast,more5) = F;

fun fzzz (F,F) = F
| fzzz (F,B) = S
| fzzz (F,S) = F
| fzzz (B,F) = F
| fzzz (B,B) = B
| fzzz (B,S) = F
| fzzz (S,F) = F
| fzzz (S,B) = B
| fzzz (S,S) = S;

fun bio(b,bb1i,bb2i,bb3i) =
  if (b=bb1i orelse b=bb2i orelse b=bb3i) then (bin)
  else (bout);

A token in r1 requires no communication between the two DMs. The token in nover1 enables transition md1 which fires and generates the final output in place Decision1. The token color changes from (b, r, vc, mc) to (b, z) where:

- vc is the class of speed for the threat;
- mc is the class of size for the threat;
- z is the type of the threat.
The function \( \text{find}_z(v_c,m_c) \) is defined in the global declaration node \( \text{(Table 5)} \); it determines the type of the threat according to the variables \( v_c \) and \( m_c \). This function implements the well-defined rules in Table 2.

If a token is generated in place \texttt{over1}, then transition \texttt{Inter1} is enabled and fires according to the function \( \text{find}_z(v_c,m_c) \). A token with attributes \((b, z)\) is generated in place \texttt{Pdec1}; it contains the identifier of the threat - the bearing - and the decision \( z \) of DM1. At this point, there are two options. Decision maker 1 may

- ignore the suggestion provided by DM2. This means transition \texttt{ntake1} does not modify the attributes of the token when it fires.
- take the suggestion from place \texttt{Suggestion2} (represented by color \((b_2, z_b)\)) together with his own decision (represented by color \((b, z_a)\)) to reach the final decision \((b, z)\). \( z_a \) is DM1's decision on what type the threat is; \( z_b \) is DM2's suggestion on what type the threat is.

The code segment embedded in transition \texttt{take1} is shown in Figure 4:

\[
\begin{align*}
\text{input} & \quad (z_a, z_b); \\
\text{output} & \quad (z); \\
\text{action} & \quad (fzzz(z_a, z_b));
\end{align*}
\]

Figure 4 Code segment for transition \texttt{take1}

The function \( fzzz(z_a, z_b) \) is defined in the global declaration node \( \text{(Table 5)} \); it implements the rules in Table 2.3. The guard function of transition \texttt{take1} is that \( b \) must be equal to \( b_2 \), \( b = b_2 \), which means that only the suggestion corresponding to the threat that DM1 is processing now can be used.

If the threat is in sector \( r_3 \), then a token is generated in \texttt{sug1}. It enables transition \texttt{ms1} and a token is generated in the place \texttt{suggestion1}. The code segment in \texttt{ms1} is the same as in \texttt{md1} (it contains the function \texttt{find}_z). DM1 communicates his suggestion on the type of the threat \( z \) on the basis of the speed and the size of the threat \((v_c, m_c)\).

With this description of the model for DM1, it is straightforward to construct the full model shown in Figure 3. The model for DM2 is symmetric to DM1’s with the suffix 2 in place of 1 in all the transition and place labels. The DM2 model operates in the same way as DM1.

When the experiment starts, all threats arrive at the outer boundary of the area simultaneously. This means all tokens representing the threats are activated in the place \texttt{Input} at the start of the
simulation. No more input tokens can be activated during the experiment. The color of a token is denoted by the 4-tuple \((b, r, v, m)\) as defined earlier.

The data preparation task is carried out by transition **Datapre**:  
- If the token attribute \(r\) takes one of the values \(r1, r2\) or \(r3\) (the threat is in the sectors monitored by DM1), a token is generated in **Input1** (input place of DM1).
- If the token attribute \(r\) takes one of the values \(r2, r3\) or \(r4\) (the threat is in the sectors monitored by DM2), a token is generated in **Input2** (input place of DM2).

The attributes \(b, r, v,\) and \(m\) remain unchanged by the operation in **Datapre**.

Two resource places have been introduced to ensure that in the model each DM processes one threat at a time, as in the experiment. The initial markings for the resource places **Res1** and **Res2** are one token with color **e** in each place [5], which ensures that only one token can be processed at a time by each DM.

**MODELING OF COORDINATION FUNCTIONS USING CPN**

In this section, the implementation of two coordination functions in CPN is described. the two functions are: (a) Checking the marking of a place; and (b) Realizing token priorities in a place. Since the current version of the *Design/CPN* software (Macintosh version 1.5) has no built-in time functions, temporal logic has been modeled as a coordination function.

**Place Markings**

In the Colored Petri Net, a place can hold multiple tokens of different classes, but the information of how many tokens in each class is not readily available. However, a subnet to get the state information of a place.

The marking of a place is the information on the number of tokens of each different class present at the place. For example, if there are two red \((R)\) and three blue \((B)\) tokens in place \(P\), then the marking of place \(P\) is \(2R + 3B\). Consider place \(P\) with one input and one output transition (Figure 5); it can hold \(m\) classes of tokens represented as Class 1, ..., Class \(m\). Design a structure so that the marking of place \(P\) can be obtained.

![Figure 5](image-url) A simple place in a Colored Petri Net
An additional place $S$ is introduced called the status place. The status place $S$ of a place $P$ is a place which links the input transition ($T_{in}$) and the output transition ($T_{out}$) of place $P$ with bidirectional arcs as shown in Figure 6. A token of with $m$ attributes ($n_1$, $n_2$, ..., $n_i$, ..., $n_m$) called the status token of place $P$ always resides in $S$.

![Diagram](image)

**Figure 6** The status place of $P$

Realization procedure:

<table>
<thead>
<tr>
<th>Initialize the status token such that</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_1$ = the number of tokens of Class 1,</td>
</tr>
<tr>
<td>$n_2$ = the number of tokens of Class 2,</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>$n_i$ = the number of tokens of Class $i$,</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>.</td>
</tr>
<tr>
<td>$n_m$ = the number of tokens in Class $m$.</td>
</tr>
</tbody>
</table>

Make the function rule for $T_{in}$ and $T_{out}$ such that whenever $T_{in}$ creates a token in $P$ or $T_{out}$ removes a token from $P$, the magnitude of $n_i$ in $(n_1..n_i..n_m)$ is changed in such a way that it is always equal to the number of tokens of Class $i$ in place $P$.

The concept of the status place can be extended to places with multiple input and output transitions (Figure 7). There is always a status token residing in $S$; its attribute values are the number of tokens of each different class in place $P$. 
Implementing Priority Rules

In Figure 8, the place P can hold m different classes of tokens denoted by Class1, ..., Class_m. Suppose each time T_out fires, one token is removed from P. It is required by the concept of operations being modeled that T_out fire according to pre-defined priority rules (assuming no equal priority for two classes):

Priority[Class(h_1)] > Priority[Class(h_2)] > ... > Priority[Class(h_m)]

where h_1, h_2, ..., h_m are integers in the range [1, m] and h_i ≠ h_j.

The realization of the priority rules consists of implementing a status place S for place P (shown in Figure 9) and defining the function embedded in T_out as follows:
if \( n(h_1) > 0 \), choose token of Class\( (h_1) \) to be fired and \( n(h_1) = n(h_1) - 1 \)
else if \( n(h_2) > 0 \), choose token of Class\( (h_2) \) to be fired \( n(h_2) = n(h_2) - 1 \)
  ...
else if \( n(h_i) > 0 \), choose token of Class\( (h_i) \) to be fired \( n(h_i) = n(h_i) - 1 \)
  ...
else if \( n(h_m) > 0 \), choose token of Class\( (h_m) \) to be fired \( n(h_m) = n(h_m) - 1 \).

Figure 9 Status place for priority rules

*Time Attributes in Colored Petri Nets*

Until the implementation of time in the CPN software is released (Version 2, currently being tested) time can be modeled as a special kind of attribute. The role of time in synchronization may then be considered as a form of temporal coordination.

Let every transition \( i \) have a time delay \( T_{di} \) associated with it and let each token carry a time attribute \( t \). Each time a token is generated by a transition, the delay of the transition will be added onto its time attribute. After firing, the time attribute changes to \( t + T_{di} \).

Figure 10 shows a simple case in which the delays of the two transitions are \( T_{d1} \) and \( T_{d2} \), respectively. When a token with time attribute \( t_s \) is processed by the first transition, its time attribute becomes \( t_s + T_{d1} \), and when it is processed by the second transition, it becomes \( t_s + T_{d1} + T_{d2} \).
In Figure 11, two places converge to one transition. The rule for incrementing the time attribute is: $\text{Max}(t_1, t_2) + T_d$. 

Figure 10  Time attribute changes in a marked graph
The calculation of the values of the time attribute is valid (i.e., this temporal coordination functions as if there was a clock in this discrete event system) only if the following conditions stand: (a) For each simple path, only one token can be involved; and (b) the net is a marked graph (each place has only one input and one output transition). If the net is not a marked graph, there might be conflict in the net. However, the conflict can be resolved on the basis of time. In order to implement this feature, a subnet must be added to coordinate and control the whole net.

THE CPN MODEL OF THE COORDINATION LAYER

The implementation of the task layer was shown in Figure 3. However, the CPN does not show how the rules that control the information flow are implemented. For example, Figure 3 shows that DM2 may generate tokens into the place Suggestion2, but does not specify what rules DM1 should follow regarding the use of the suggestions. In this section, the coordination layer for the Colored Petri Net model of the Naval Outer Air Battle is constructed. The following coordination rules are implemented: (a) priority rules for inputs to DMs, and (b) temporal coordination rules. It is assumed that if a suggestion is available at the time a DM may use it, he will always take the suggestion, while if the suggestion is not available when a DM is ready to use it, the DM will proceed without waiting for the suggestion to arrive.

The execution of the model, i.e., the simulation, consists of two stages. The initial marking of place Input is the set of tokens representing all the threats (one token represents one threat). At the data preparation stage, transition Datapre is enabled and keeps on firing input tokens for DM1 and DM2. DM1 and DM2 will not process a token until the place Input is empty. The second stage of the simulation, called the information processing stage, begins when all the tokens in Input have been removed and tokens have been generated in places Input1 and Input2. This corresponds to the experimental condition, that all threats arrive simultaneously and are immediately visible to the DMs.

In the model, priority rules for places Input1 and Input2 need to be implemented, which correspond to each DM's strategy for determining the sequence of dealing with the threats. Figure 12 show the implementation of the priority rule for DM1. The one for DM2 is symmetric. The implementation of priority here is slightly different from that described in the previous section.

FIGURE 12 - old 5.1
Place **Input1** can hold three classes of tokens r1, r2, and r3. A place **Sinput1** (for status of Input1) has been added to the net; It has a three-tuple (ro1, ro2, ro3) with each component corresponding to the number of tokens in **Input1** belong to class r1, r2, and r3 respectively. The code segment for **Datapre** is:

```plaintext
input (ro1,ro2,ro3,ro4,r);
output (ri1,ri2,ri3,ri4);

action

case r of
  r1 => (ro1+1,ro2,ro3,ro4)
  r2 => (ro1,ro2+1,ro3,ro4)
  r3 => (ro1,ro2,ro3+1,ro4)
  r4 => (ro1,ro2,ro3,ro4+1);
```

At the data preparation stage, the initial marking of **Sinput1** is (0, 0, 0). After all the tokens in **Input** are fired, the marking of **Sinput1** becomes (n1, n2, n3). The transition **Tp1** generates tokens in place **prires1** according to the priority rule for place **Input1**. For example, if the required priority rule for tokens to be fired in place **Input1** is “priority [r3] > priority [r2] > priority [r1]”, the code segment for **Tp1** can be written as:

```plaintext
input (ro1,ro2,ro3);
output (ri1,ri2,ri3,rr1);

action

if ro3>0 then (ro1,ro2,ro3-1,r3)
else if ro2>0 then (ro1,ro2-1,ro3,r2)
else (ro1-1,ro2,ro3,r1);
```

The guard function of **Tp1** is: ro1 > 0 or else ro2 > 0 or else ro3 > 0. This means that if there is no token in **Input1**, **Tp1** cannot fire.

The guard function for transition **Div1** is r = rr1, which means that if a token with attribute rr1 is in place **prires1**, only a token with the same attribute rr1 in **Input1** will be removed when **Div1** fires. Each time a resource token is available in **Res1**, and the guard function is satisfied (there is at least one token in **Input1**), **Tp1** would check the token in **Sinput1**, and put a token with attribute r1, r2 or r3 into **prires1**.
The first step in implementing the temporal aspect of this model is to add appropriate time attributes to the tokens. The subnet of DM1 which deals with the time attribute has shown in Figure 13. The color set and variables defined for time attributes are as follows:

```plaintext
color TS = int;
var t, t1s, t1ci, t1co, t1f, t2s, t2ci, t2co, t2f, tf1, tf2: TS;
```

**FIGURE 13 - old 5.2**

Two time attributes have been added to the tokens in the DM1 subnet: t1s (the starting time of a token) and t1c (the current time, denoted by t1co, t1ci, t1f in the arc expressions). The resource token also takes a time attribute, t1r (the ending time of last token and the starting time for next token which is denoted by t1f in arc expressions). The time stamp t1s remains unchanged while a token is in the DM1 subnet, while t1c changes as described in the previous section. Figure 14 depicts the changes in t1s and t1c when a token is processed by a transition.

![Figure 14 Change in time attributes](image)

The process for changing the time attribute is summarized as follows:
- When an input token enters the DM process, t1s and t1c are both set to t1r
  \[ t1s \leftarrow t1r, t1c \leftarrow t1r. \]
- t1s remains unchanged during the whole process
- t1c increases by the amount of the processing delays.
- When the processing of a threat/token has been completed, t1r is set to t1c:
  \[ t1r \leftarrow t1c. \]

At the beginning of the simulation, t1r is set to zero, which means that the starting time of the whole system is zero.
The model for DM2 is symmetric to DM1. Figure 15 shows the complete net with time attributes. If the net is a marked graph, and each simple path involves only one token at a time, the above method can be used directly. However, inspection of the task layer reveals that while each DM can process one token at a time, this net is not a marked graph. Consequently, to implement the time coordination function, a subnet (functioning as time controller) is required.

FIGURE 15 - old 5.4

The organizational structure of this team is a parallel one: the two DMs are functioning at the same time with each one processing one token. The time controller which is implemented in CPN is used to coordinate this parallel process (Figures 16 and 17). The subnet in Figure 17 works like a scheduler to coordinate the behavior of the model.

FIGURE 16 - old 5.6a

FIGURE 17 - old 5.6b

There are still some technical modeling problems associated with the implementation of the net that occur because of the software implementation. For example, if all the tokens in \textcolor{red}{\textbf{Input1}} have been processed, while there are still some tokens left in \textcolor{red}{\textbf{Input2}}, deadlock will occur. This problem was resolved in the actual simulation net (see next section), but it is mentioned here to point out that there is still much to be done to develop CPN modules that model specific logical operations.

RESULTS

The CPN model for the Naval Outer Air Battle consists of two layers: the task layer and the coordination layer. To obtain the actual executable net that can be used to obtain simulation results, a subnet must be constructed to automate the switching between the two stages (data preparation stage and information processing stage) and to deal with the remaining input tokens when one of the input place of a DM is empty.

The actual CPN net used in the simulations is shown in Figures 18 and 19. The initial marking of the net is:

- For \textcolor{red}{\textbf{Input}}: All the tokens represent all incoming threats
- For NI: A token whose color is the number of the tokens in \textcolor{red}{\textbf{Input}}
- For Sinput1 and Sinput2: A token with color (0, 0, 0)
For **SSug1** and **SSug2**: A token with null attributes (1000, 1000, 1000)

For example, if the initial marking for **Input** is two token with colors (30, r1, v720, m5) and (45, r1, v360, m2), then the initial marking for **NI** would be 2.

**FIGURE 18 - old A.1a**

**FIGURE 19 - old A.1b**

When **Datapre** fires for the first time, it puts a token in **Last** and decreases the value of the token in **NI** by one. Then **Datapre** keeps on firing, decreasing each time the value of the token in **NI** by one until **Input** is empty. At that time the token in **NI** has the value zero. Transition **Sw** is enabled and fires to put one resource token with time attribute 0 in **Res1** and **Res2**. The process continues until either **Input1** or **Input2** is empty. Let **Input1** be empty (the token in **Sinput1** is (0, 0, 0)). When the resource token in **Res1** is available again, transition **Finish1** fires to put a token with a very large finishing time value (in the net, this large value is 500) in place **Co1**. Therefore, the tokens in **Input2** can continue to be processed.

The guard function for **Compare1** is \( t_{1f} < t_{2f} \); for **Compare2** is \( t_{2f} < t_{1f} \) in Figure 17. This is not symmetric. To make the net symmetric, the guard functions for **Compare1** and **Compare2** are modified so that when the time a suggestion becomes available and the time the token goes to **Pdec1** or **Pdec2** are equal, the suggestion is always taken. The guard function for **Compare1** is: \( t_{1f} < t_{2f} \) or else \( t_{1f} = t_{2f} \) and also \( r = r_2 \) or else \( r = r_3 \)). The guard function for **Compare2** is: \( t_{2f} < t_{1f} \) or else \( t_{1f} = t_{2f} \) and also \( r = r_2 \) and also \( r = r_3 \)).

Places **CSuggestion1** and **CSuggestion2** have been added onto the net to collect the tokens that enter **Suggestion1** and **Suggestion2**. One additional binary attribute has been included that takes the value 1 when communication occurs, and 0 when no communication occurs.

The time delays for each transition are shown on Figure 20. The input data for the simulation is listed in Table 6.

**FIGURE 20 - old with time delays**
Table 6 The input data for the simulation

<table>
<thead>
<tr>
<th>Bearing</th>
<th>Region</th>
<th>Speed</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>r1</td>
<td>v1200</td>
<td>m5</td>
</tr>
<tr>
<td>99</td>
<td>r2</td>
<td>v720</td>
<td>m5</td>
</tr>
<tr>
<td>123</td>
<td>r3</td>
<td>v600</td>
<td>m2</td>
</tr>
<tr>
<td>130</td>
<td>r3</td>
<td>v900</td>
<td>m4</td>
</tr>
<tr>
<td>135</td>
<td>r3</td>
<td>v450</td>
<td>m5</td>
</tr>
<tr>
<td>163</td>
<td>r4</td>
<td>v360</td>
<td>m2</td>
</tr>
<tr>
<td>181</td>
<td>r4</td>
<td>v1200</td>
<td>m4</td>
</tr>
<tr>
<td>202</td>
<td>r4</td>
<td>v300</td>
<td>m5</td>
</tr>
</tbody>
</table>

The priority rule implemented for this simulation is a random one, i.e., there is no explicit, a priori rule setting priorities among threats. To implement random inputs for the decision maker, the code segment for Tp1 was written as follows:

```plaintext
input (ro1,ro2,ro3,t1f);
output (ri1,ri2,ri3,t,rr1);
action
let val ii=random(ro1`r1+ro2`r2+ro3`r3)
in
  case ii of
    r1 => (ro1-1,ro2,ro3,t1f,r1)
    | r2 => (ro1,ro2-1,ro3,t1f,r2)
    | r3 => (ro1,ro2,ro3-1,t1f,r3)
end;
```

Table 7 contains the simulation results for random inputs (no priority). Ts stands for the starting time of processing a particular threat; Tc stands for the time the processing of that threat. COMM1 or COMM2 show whether communication occurred (1 stands for yes, 0 for no). Figure 21 is the graphic interpretation of the simulation result in Table 7.
The delay for each information flow path in the task layer is as follows:
- For threats in r1 or r4 (non-overlap areas), the total delay is 6 units of time.
- For a threat in r2 processed by DM2 or in r3 processed by DM1 (suggestion),
  the time delay is 6
- For a threat in r2 processed by DM1 or r3 processed by DM2, the time delay is 6, if a suggestion is not available, and 8 if suggestion is used.

<table>
<thead>
<tr>
<th>B</th>
<th>Region</th>
<th>Ts DM1</th>
<th>Tc DM1</th>
<th>COMM1</th>
<th>Ts DM2</th>
<th>Tc DM2</th>
<th>COMM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>R1</td>
<td>24</td>
<td>30</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>99</td>
<td>R2</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>24</td>
<td>30</td>
<td>-</td>
</tr>
<tr>
<td>123</td>
<td>R3</td>
<td>18</td>
<td>24</td>
<td>-</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>130</td>
<td>R3</td>
<td>12</td>
<td>18</td>
<td>-</td>
<td>38</td>
<td>46</td>
<td>1</td>
</tr>
<tr>
<td>135</td>
<td>R3</td>
<td>0</td>
<td>6</td>
<td>-</td>
<td>30</td>
<td>38</td>
<td>1</td>
</tr>
<tr>
<td>163</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>181</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12</td>
<td>18</td>
<td>-</td>
</tr>
<tr>
<td>202</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>18</td>
<td>24</td>
<td>-</td>
</tr>
</tbody>
</table>

FIGURE 21 - old timeline

In Figure 21, the token with bearing 135° was processed by DM1 during the time interval 0-6 on the time line, and by DM2 at time 30 to 38. When DM2 begins to process this token the suggestion has already been made. Communication occurs and the processing time is 8.

The token with bearing 99° has been processed by DM1 at time 6-12, and processed by DM2 at time 24-30. When DM1 is processing the token, a suggestion is not been produced yet by DM2. Communication does not occur and the processing time is 6. The same situation arises for threats with bearing 123° and 130°.

A second simulation was executed but this time with priorities assigned to the various threats. The priority rules implemented for DM1 was r3 > r2 > r1, and for DM2 r2 > r3 > r4. The code segment for $T_{p1}$ is as follows:
input (ro1,ro2,ro3,t1f);
output (ri1,ri2,ri3,t,rr1);

action
if ro3>0 then (ro1,ro2,ro3-1,t1f,r3)
else if ro2>0 then (ro1,ro2-1,ro3,t1f,r2)
else (ro1-1,ro2,ro3,t1f,r1);

Table 8 is shows the simulation results for inputs with priority. Figure 22 is the graphic interpretation of these simulation results.

Table 8  Simulation results for inputs with priority

<table>
<thead>
<tr>
<th>B</th>
<th>Region</th>
<th>Ts DM1</th>
<th>Tc DM1</th>
<th>COMM1</th>
<th>Ts DM2</th>
<th>Tc DM2</th>
<th>COMM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>R1</td>
<td>26</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>99</td>
<td>R2</td>
<td>18</td>
<td>26</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>123</td>
<td>R3</td>
<td>6</td>
<td>12</td>
<td>-</td>
<td>22</td>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>130</td>
<td>R3</td>
<td>0</td>
<td>6</td>
<td>-</td>
<td>6</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>135</td>
<td>R3</td>
<td>12</td>
<td>18</td>
<td>-</td>
<td>14</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>163</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>36</td>
<td>42</td>
<td>-</td>
</tr>
<tr>
<td>181</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>30</td>
<td>36</td>
<td>-</td>
</tr>
<tr>
<td>202</td>
<td>R4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>42</td>
<td>48</td>
<td>-</td>
</tr>
</tbody>
</table>

FIGURE 22- old timeline

In Figure 22, DM1 processes threats in r3 first, then in r2, and lastly in r1. DM2 processes threats in r2 first, then in r3, and then in r4. For the threat with bearing 135˚, DM2 begins to process at time 14; when the token arrives to Pdec2, the time is 19, and at that time the suggestion is ready. Communication occurs and the time delay for DM2 is 8.

CONCLUSIONS
REFERENCES


