

Petri Net analysis of the MASCOT Pool IDA Communication Mechanisms

Mustafa A. Jiffry, King's College, University of London, UK

Abstract

The paper is concerned with modelling and analysis of the MASCOT Pool IDA communication mechanisms. Both the Four-slot fully asynchronous mechanism and the Two-slot conditionally asynchronous mechanism are investigated. The mechanism properties are defined first, to provide the basis for Petri nets modelling. Place/Transition nets and Coloured Petri nets are used to produce the models. The Design/CPN software tool is used to verify those communication mechanisms. It is shown that the method adopted is an effective way for verification of the correctness of interprocess communication mechanisms used by parallel architectures of distributed real-time systems.

1. Introduction

The Four-slot fully asynchronous communication mechanism was first developed by Simpson, and presented publicly in the IEE Proceedings [1]. The mechanism provides a new technique for solving the problem of accurate data transfer between concurrent processes using parallel architectures within distributed real-time data processing systems. Generally the techniques which have been used to provide concurrent access to shared data, were based on the mutual exclusion principle. This new form of fully asynchronous communication mechanism is not based on the mutual exclusion principle, instead it uses co-operative access control. Co-operative access control uses control variables to steer both the reader and writer processes in order to preserve data integrity. The control variables are used as indices for the mechanism steering strategy, and are not part of any conditional statement. Because there is no existence of any form of wait protocol, the reader and writer processes are fully asynchronous and do not have to wait for each other to maintain data integrity.

2. Background

This section contains an informal introduction to MASCOT, the interprocess communication of the Pool IDA, and the Design/CPN software tool. The reader can refer to a more formal presentation in the literature [1,2,3,4,5,6,7,8,9,10].

2.1 MASCOT

MASCOT (Modular Approach to Software Construction Operation and Test) is a software design method for real-time systems, based on data flow concepts in which a key feature is the use of special symbols to represent the real time dynamics of concurrent processes

communicating through shared memory. Figure 1 shows the communication mechanism of reference data being represented by a MASCOT pool IDA.

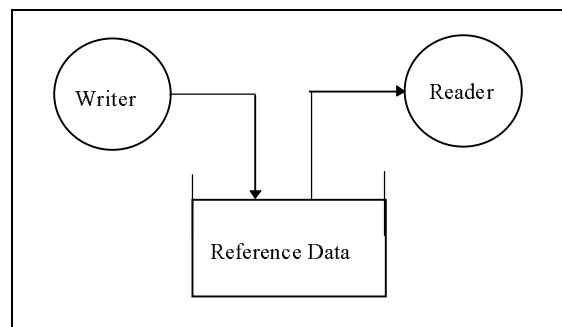


Figure 1 Shared Pool IDA

In MASCOT there are two fundamental classes of components:

1. Activity - the data processing component
An activity is a single sequential program thread that can be independently scheduled. The activities communicate through IDAs. The IDA provides the necessary synchronisation and mutual exclusion facilities. The graphical representation of activities have rounded boundary and are often shown as circles.
2. Intercommunication Data Area (IDA) - the data communication component
An IDA is a passive element which services the data communication needs of activity components. It can contain its own private data areas, and provides procedures which activities use for the transfer of data. The graphical representation of IDAs are rectangular shapes with special forms for the channel and pool.

MASCOT supports several forms of IDAs which are based on their behaviour :

- Channel IDA
The channel IDA allows message data to be passed from one process to another. It supports data communication between producers and consumers. It can contain one or more items of information. Writing to a channel adds an item without changing items already in it (a non-destructive write operation). The read operation is destructive, since it removes an item from the channel. A channel can become empty and because its capacity is finite, it can become full.
- Pool IDA
The pool IDA allows reference data (a table or dictionary) to be passed from one process to another. The reference data is retained within the pool, where it can be consulted at any time by the reader and updated at any time by the writer. The write operation is destructive and the read operation is non-destructive.

The classification of the communication models in MASCOT is based on its synchronous form. The following presents the distinction between reference and message data for a single-writer and a single-reader;

1. Reference data (Pool IDA)
 - Fully asynchronous (Four-slot mechanism).

- Conditionally asynchronous (Two-slot mechanism).
2. Message data (Channel IDA)
- Loosely synchronous (Bounded buffer).
 - Fully synchronous (Rendezvous).

2.2 Interprocess communication of the Pool IDA

The reader and the writer process co-operate dynamically by means of control variables so that concurrent access to any data record never occurs. The control variables are bit type, so their values are always guaranteed to be coherent, and where reading and writing operations can be concurrent and need not interfere with each other. They can be implemented by a single latch (a bistable latch or a D-type flip-flop).

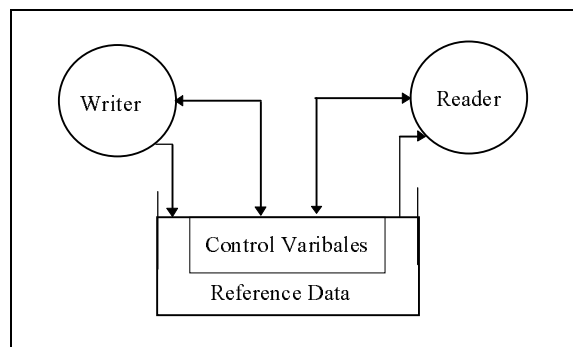


Figure 2 Co-operative access control

Therefore, the mechanism design is based entirely on co-operative access control by means of bit control variables (single binary digit of 0 or 1) and these variables are never used in conditional statements which would affect process timing. Figure 2 shows the interprocess communication mechanism which is based on co-operative access control.

2.3 The Design/CPN software tool

Petri nets allow a system to be modelled where the resulting Petri net model can be analysed to reveal information about the structure and dynamic behaviour of the modelled system. Coloured Petri nets belong to a class of high-level nets which can model complex systems in a manageable way. The practical use of Coloured Petri nets is highly dependent upon the existence of adequate computer tools in-order to assist the modeller with the following [2] :

- Storing and handling all the details of his CP-net model.
- Obtaining better results (e.g. where the CPN editor provides precision and quality drawing capabilities, and the simulator provides a computer support for complex analysis methods).
- Getting faster results (e.g. where the CPN editor is used to create and modify CP-net models, and the simulator is used to construct occurrence graphs).
- The possibility of making interactive presentation of the analysis results.
- The possibility of hiding technical aspects of the CP-net theory inside the tools.

The Design/CPN is an interactive computer tool to build and execute CP-nets [3,4,5]. It requires X-Windows/Unix, or MacOs (Macintosh) platform. The tool provides the following :

- An editor to create and manipulate CP-nets.
- Syntax checker for validating CP-nets.
- A simulator for executing and debugging CP-nets.
- Facilities for organising a CP-net into a hierarchy of modules.
- Animation and charting facilities for displaying simulation results.

3. The Mechanism Properties

The mechanism we are considering involves the case of a single writer communicating with a single reader by using a slot, which is a data area in shared memory capable of holding a single data record in transit. The reader and writer are individually represented by a process, which is an independent thread of execution defined by a series of sequential operations. Control variables are being used to regulate or direct the writing and reading of data. Both processes (reader and writer) can run in an endless loop to perform its dedicated function.

The author is trying to define the mechanism properties which will ensure through logical reasoning that the resulted Petri net model is a correct model. In other words, the mechanism software design and the Petri net model represent the same system. Defining these properties will provide the basis for Petri nets modelling. The main properties of the fully asynchronous communication mechanism have been defined by Simpson [1]. These properties relate to the behaviour of the mechanism and to how the interprocess communication is regulated. Additional properties relating to model representation will be introduced for two purposes. Firstly, to make the Petri net model a true reflection of the software design. Secondly, to help with clarity of presentation of the analysis. The properties list is as follows;

1. Asynchronism

Neither process (reader or writer) may affect the timing of the other as a direct result of its communication operations.

2. Data coherence

Data must always be passed as a coherent set. Interleaved access to any data record by the reader and the writer is not permitted.

3. Data freshness

The latest completed data record produced by the writer must always be made available for use by the reader.

4. Mutually independent cycles

Each process (reader and writer) will be represented by a mutually independent Petri net cyclic model, since each process is expected to run in an endless loop.

5. Reflecting the software design

The model must always reflect the software design forms proposed by Simpson [1]. The software module will consist of global and local control variables, and access algorithms for both the reader and writer processes. The execution of any single statement within each access algorithm will be represented by the occurrence (firing) of a single transition in the corresponding Petri net model. Global control variables will have global scoping, so they will keep their values between cycle execution (analogous to procedure or function calls). They will be represented by a place stating their current bit value (0 or 1). Local control variables will have local scoping, so their values will be set dynamically during Petri nets execution, and maintained throughout the process execution cycle as required. This property

will make sure that the dynamic execution of the Petri net model will be the same as the execution of the software design. If we break point the execution of the Petri net model at any step, we should be able to tell, for each process, what is the current statement to be executed next and what is the current value of each control variable. The results should be identical to those which would be obtained if we were to break point the execution of the software design at the corresponding statement.

6. **Reflecting the way the mechanism is using control variables**

The control variable is a bit type (single binary digit). It is used to regulate or steer the cooperative processes, in-order to maintain data integrity. The Petri net model must maintain the use of each control variable as intended by the proposed mechanism software design.

7. **Reflecting the concurrent process execution**

Petri nets execution is naturally concurrent, since the transition firing is never predetermined. When modelling the access of control variables, attempt will be made to ensure that transitions will be concurrently enabled. Therefore, if there exist any two enabled transitions having the same input place, they might be in conflict for the same enabling token. Removing all token conflicts will make all the transitions to be concurrently enabled. Here, we have two situations:

- **Both processes want to read the same control variable**

In this situation, the number of tokens will be doubled, so each transition will have its own set of enabling tokens. Therefore, we are matching the hardware design, since reading a latch is done on the rising or falling clock signal edge.

- **One process is reading and the other is updating the same control variable**

In this case, the hardware design can accommodate writing and reading of the bit variable, since the latch will lock its input (write value) on the rising edge, and deliver its output on the falling edge of the clock signal (or visa versa). Therefore, the control variable bit value will be written on the rising edge and read on the falling edge of the clock signal (or visa versa). The Petri nets execution will simulate this situation, since the firing of either transitions is never predetermined. Let us consider the following scenario:

$Tw2$, $Tr0$ are two enabled transitions which require the same enabling token from the same input place (see Figure 8). The execution sequence of $Tw2$, $Tr0$ results in writing the control variable followed by reading it. But the execution sequence of $Tr0$, $Tw2$ results in reading the control variable followed by writing it. Both of these sequences are permitted by the fair execution case of Design/CPN.

8. **Metastability**

The phenomenon of metastability occurs whenever a shared control variable is read close to the time it is being changed. However, the execution of the Petri net model does not allow it to happen because the two transitions (see Figure 8, where $Tw2$ writes and $Tr0$ reads the same shared control variable *latest*) will not fire at the same time, since they will be in conflict (requiring the same enabling token *latest*). In the case of the hardware bistable latch, the clock signal pulse width (the time duration between the rising and falling edges) can be used to prevent metastability. Therefore, the shared control variables are always stable and having a valid state of 0 or 1.

9. **Detecting error condition**

Detecting data coherence errors should not load the Petri net model by the creation of conflicts between transitions requiring the same enabling token.

10. **Naming convention for places and transitions**

For Place/Transition nets, the author has used a naming convention which reflects the software design, by assigning the software design statements as names to transitions and the control variables values as names to places. Although this has resulted in duplicate names (which is not allowed by Petri nets), it is however intended to give a clear description of the

use of those transitions and places. Of course, for Coloured Petri nets, unique descriptive names are used in accordance with the Petri nets rule.

4. The One-Slot Mechanism

This mechanism contains one possible place for data in transit. This is a null form of co-operative access control. There is no need for control variables to regulate or steer data access, since there exists only one slot to store the data in transit.

```

mechanism one-slot;
  var data : TypeData :=null;

  procedure write(item : TypeData);
    begin data := item; end;

  function reader : TypeData;
    begin read := data; end;

end.

```

Figure 3 The One-slot mechanism (in a Pascal like language)

The software design of the One-slot mechanism as in Figure 3, defines a single slot (to hold the data) and two access algorithms (write and read). The single slot is initialised to null in-order to ensure data coherence when the reader accesses the mechanism before the first write. Integrity of the One-slot mechanism is only preserved if write and read never overlap by scheduling the data access.

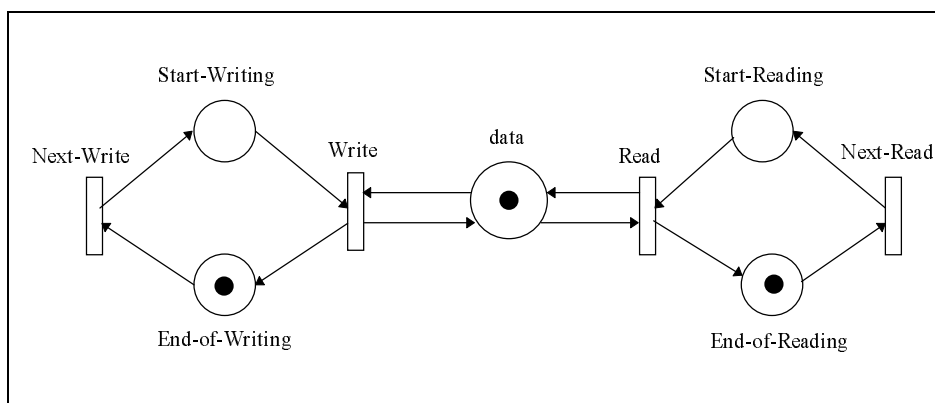


Figure 4 Place/Transition net model for the One-slot mechanism

Figure 4 shows the Place/Transition net model for the One-slot mechanism. The Coloured Petri net model is similar, since the tokens within the Coloured Petri net model would carry no information except for their presence or absence.

5. The Two-slot Communication Mechanism

The software design of the Two-slot mechanism, is shown in Figure 5. This design consists of the following;

1. A two slots array *data[bit]* to hold information in transit, and it is pre-set to *null*, in-order to ensure that a read before the first write will obtain *null* data value.
2. A control variable *latest*, which indicates the latest data written. It is pre-set to zero.
3. The write algorithm selects alternate slots for writing, and at the end of each write it indicates the latest data.
4. The read algorithm always starts to read data from the last completely written slot.

```
mechanism two-slot;
type bit = 0..1;
var data : array[bit] of TypeData := (null, null);
    latest : bit := 0;

procedure write(item : TypeData);
var wx : bit;
begin
    wx := NOT latest;
    data[wx] := item;
    latest := wx;
end;

function read : TypeData;
var rx : bit;
begin
    rx := latest;
    read := data[rx];
end;

end.
```

Figure 5 The Two-slot algorithm (in Pascal like language)

The author will be using Place/Transition nets and Coloured Petri nets to model the Two-slot mechanism. The properties established earlier would be preserved, so that the Petri net model can reflect the mechanism behaviour under investigation. Each mechanism cycle (reading and writing) would be modelled separately, in-order to reduce the complexity and provide a simple model to present each cycle on its own. Combining both into a single diagram is a trivial step. Also, when the author combines both cycles into a single Coloured Petri net model, the folding power of Coloured Petri net will be demonstrated.

5.1 Place/Transition net model of the Two-slot Mechanism

Figure 6 shows the PT-net model of the read access algorithm. *Start-Reading* is the place which establishes the entry point of the read access function. The initial marking represents the initialisation values defined by the software design as indicated by Figure 5. The

transition $rx=latest$ is the first executable statement, and $Read=data[rx]$ is the second executable statement. The places $latest=0$ and $latest=1$ represent the global control variable $latest$ having a value of 0 or 1. When a token is present at the place $latest=0$, it will mean that, the control variable $latest$ is set to have a zero (0) bit value. Since all the properties set out earlier are being encapsulated within this PT-net model, the model represents the same reading access algorithm defined by the software design form of the Two-slot mechanism.

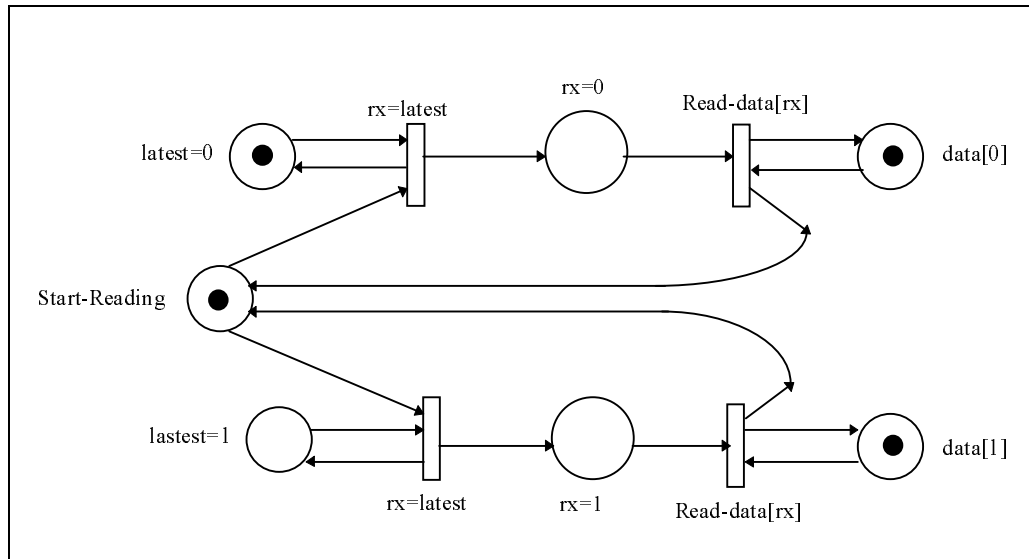


Figure 6 PT-net model for the reading cycle of the Two-slot mechanism

The model given in Figure 6 reveals how the mechanism keeps the reader process from jumping to the other slot. The reader does not have the ability to switch over until the writer indicates the presence of a new fresh data by changing the bit value of the control variable $latest$. This control variable steers the reader to the latest completely written data.

Figure 7 shows the PT-net of the write access algorithm. The write access procedure consists of an entry point and three executable statements. The place $Start-Writing$ marks the entry point, and the transitions $wx=Not(latest)$, $Write-data[wx]$, and $Latest=wx$ represent the three executable statements. The initial marking represents the initialisation values of the mechanism software design. The mechanism uses the control variable $latest$ to point to the slot containing the freshest data. Therefore, the PT-net model of the write access algorithm must reflect the use of the control variable $latest$ (as stated in the previously defined properties list). The transition $latest=wx$ will exchange the bit value of this control variable from 0 to 1, by removing the token from place $latest=0$ and depositing it in place $latest=1$, and visa versa. Since the PT-net model guarantees encapsulation of the previously defined properties, we can claim that this is a correct model.

It can be seen in Figure 7 how the writer jumps between slots, regardless whether the reader is accessing the slot or not. The writer does not have any knowledge of whether the data slot to be used next, is free and is not used by the reader. Therefore, if the writer process is faster than the reader process, a data coherence violation will arise. Simpson refers to this mechanism as the swung buffer [1], since the writer process writes alternate data items to alternate slots which are then swung into visibility (by the indication of the $latest$ control variable bit value) for use by the reader.

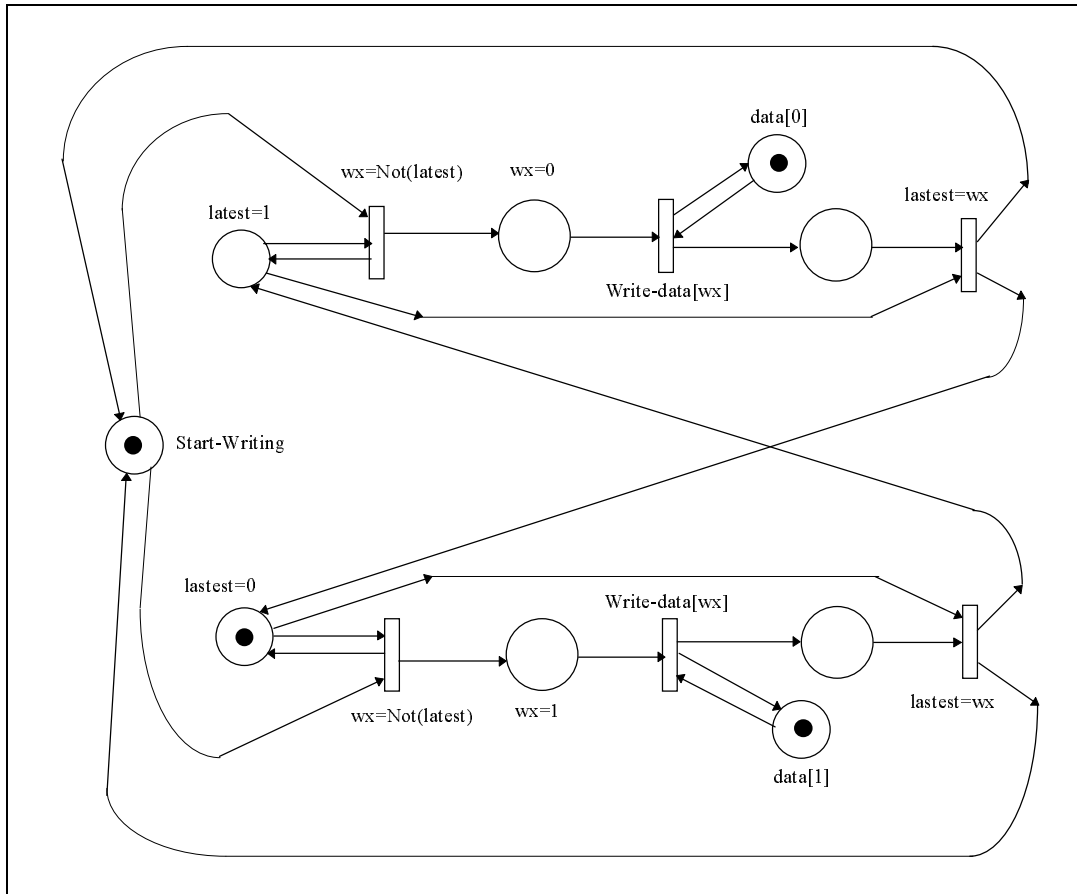


Figure 7 PT-net model for the writing cycle of the Two-slot mechanism

5.2 Coloured Petri net model of the Two-slot Mechanism

Coloured Petri net is a powerful net type, which can describe complex systems in a manageable way. In the PT-net model, we had to represent the mechanism state based on the control variable bit value, with two identical separate subnets. But now a more compact representation can be achieved, by folding the two read access function subnets into a single subnet, and the same can be done with the two subnets of the write access procedure. This is only a one level folding, and we can see the reduction of complexity by using a high level net. The original PT-net model (for both processes) consists of twelve (12) places and ten (10) transitions, whereas the equivalent CP-net model consists of seven (7) places and five (5) transitions.

The folding power of Coloured Petri net is quite a temptation for the modeller to use in-order to reduce the Petri net model size, but it can lead to complication in presentation of the analysis. It is possible to fold the Petri net model further by two levels: first by combining both processes (reader and writer) and second by combining the mechanism progression steps. If the described folding is done, there will be a violation of the previously defined properties list which would effect the clarity of presentation of the analysis. The author is aiming for a simple and precise Petri net model, in-order to reveal all the information that can be obtained towards the understanding of the mechanism behaviour.

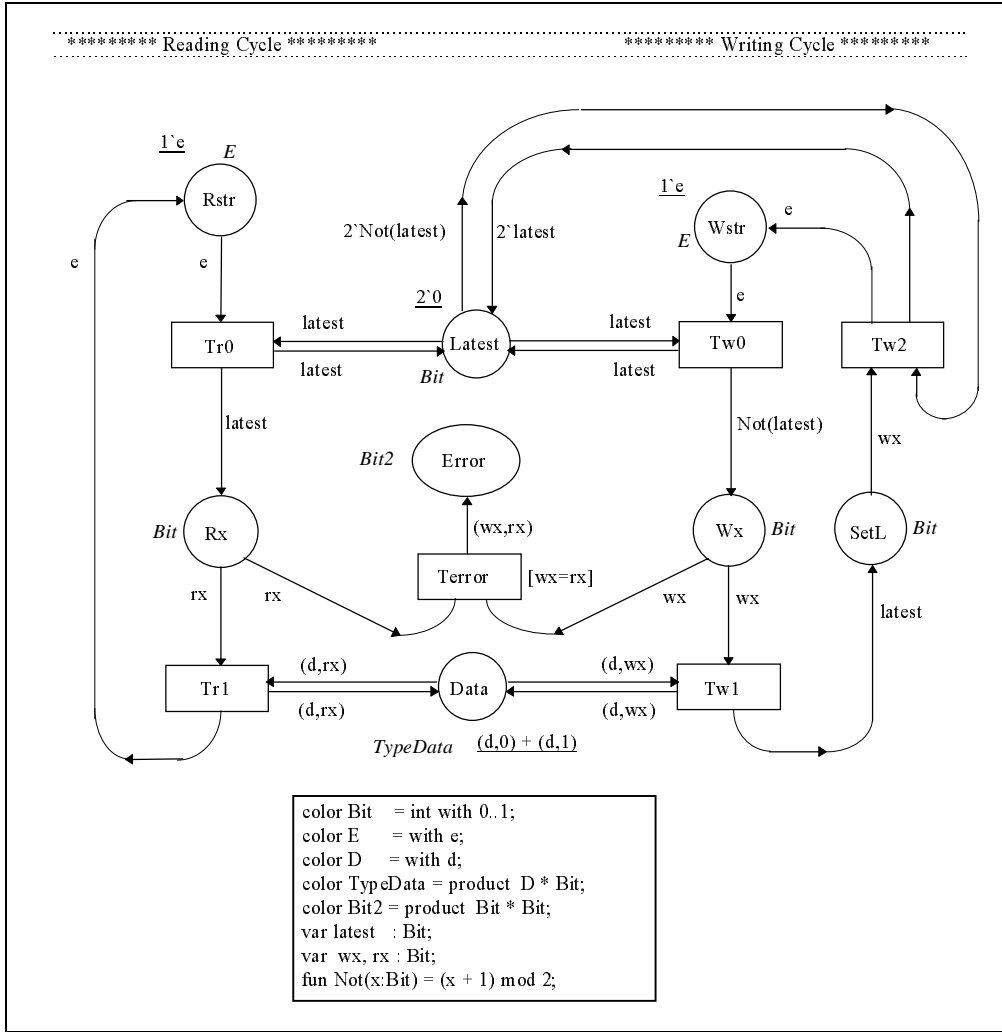


Figure 8 CP-net model for the Two-slot mechanism with error detection

The CP-net, shown in Figure 8, resulting from the one level folding of the reader and writer PT-net models, is the complete Coloured Petri net model of the Two-slot mechanism. The property of reflecting the concurrent execution of both processes is preserved by setting the *Latest* place initial marking with two tokens ($2 \cdot 0$), in-order to concurrently enable both transitions *Tr0* (represents $rx=latest$) and *Tw0* (represents $wx=Not(latest)$).

In this study, it is possible to identify locations in the CP-net model where metastability can occur. In Figure 8, for example, the simultaneous firing of *Tw2* and *Tr0* will result in metastability. However, *Tw2* and *Tr0* can not fire at the same time because the CP-net execution does not allow them since they are in conflict for the same enabling token *latest*.

5.3 Verification of the Two-slot Communication Mechanism

The Two-slot mechanism is not a fully asynchronous communication mechanism. The author needs to detect error conditions which may take place. The only property violation this mechanism may commit is that of data coherence, since data freshness is accounted for. Data coherence error arises when both processes (writer and reader) try to access the same data slot.

Design/CPN is used to verify and prove the occurrence of violation of data coherence. An error detection facility must be added to satisfy the properties list defined earlier.

Figure 8 shows the proposed error detection facility. The CP-net model will halt its execution, when the state of data coherence violation arises. There is no loading on the original CP-net model, if it executes without violating the data integrity property. When a data coherence violation takes place, then three transitions (*Terror*, *Tw1*, *Tr1*) will be in conflict by requiring the same enabling token. The guard ($[wx=rx]$) associated with the *Terror* transition makes sure that the occurrence of the error condition is the only way that will lead to enabling the *Terror* transition. When the error condition is enabled, it will mean that the three transitions (*Terror*, *Tw1*, *Tr1*) are enabled, and either *Terror* will fire or *Tw1* and or *Tr1* will fire in accordance with the Petri nets rule. But we are seeking a full occurrence graph (state space), which means we are interested in the possibility of *Terror* to occur or not.

Since we now have the complete CP-net model with error detection, we can proceed with the Design/CPN tool to generate the occurrence graph, which is the CP-net state space (some times it is referred to as reachability tree) that could be presented graphically.

Statistics	

Occurrence Graph	
Nodes:	20
Arcs:	38
Secs:	0
Status:	Full
Boundedness Properties	

Best Upper Multi-set Bounds	
NewData 1	$1^{\wedge}(d,0) + 1^{\wedge}(d,1)$
NewLatest 1	$2^{\wedge}0 + 2^{\wedge}1$
NewError 1	$1^{\wedge}(0,0) + 1^{\wedge}(1,1)$
NewRstr 1	$1^{\wedge}e$
NewRx 1	$1^{\wedge}0 + 1^{\wedge}1$
NewSetL 1	$1^{\wedge}0 + 1^{\wedge}1$
NewWx 1	$1^{\wedge}0 + 1^{\wedge}1$
NewWstr 1	$1^{\wedge}e$
Best Lower Multi-set Bounds	
NewData 1	$1^{\wedge}(d,0) + 1^{\wedge}(d,1)$
NewLatest 1	empty
NewError 1	empty
NewRstr 1	empty
NewRx 1	empty
NewSetL 1	empty
NewWx 1	empty
NewWstr 1	empty
Liveness Properties	

Dead Markings:	[15,20]
Dead Transitions Instances:	None

Figure 9 Design/CPN (partial) statistical report for the Two-slot

Figure 9 presents a partial statistical report generated by using Design/CPN to run the CP-net model of the Two-slot mechanism. The occurrence graph generated is full and small in size (20 nodes in total). Full state space of the CP-net model is required for verifying the mechanism, in-order to identify all the possible states the mechanism can reach. Data coherence error is reachable, by checking the liveness and boundedness properties of the CP-net model. The liveness property indicates that the CP-net model can have two states of dead markings, they are state number 15 and 20. These states (15 and 20) can be expanded by Design/CPN to show the marking of all the places of the CP-net model. By using the best upper multi-set bounds of the boundedness property, we can see that the place *Error* can have an upper bound marking of $1 \setminus (0,0) + 1 \setminus (1,1)$. This means, data coherence condition is reachable when both processes, the reader and writer, are either accessing *data[0]* or *data[1]*, since we have both cases of $wx=rx=0$ and $wx=rx=1$. Because the occurrence graph is small in size, an attempt is made to use Design/CPN to draw it (as seen in Figure 10).

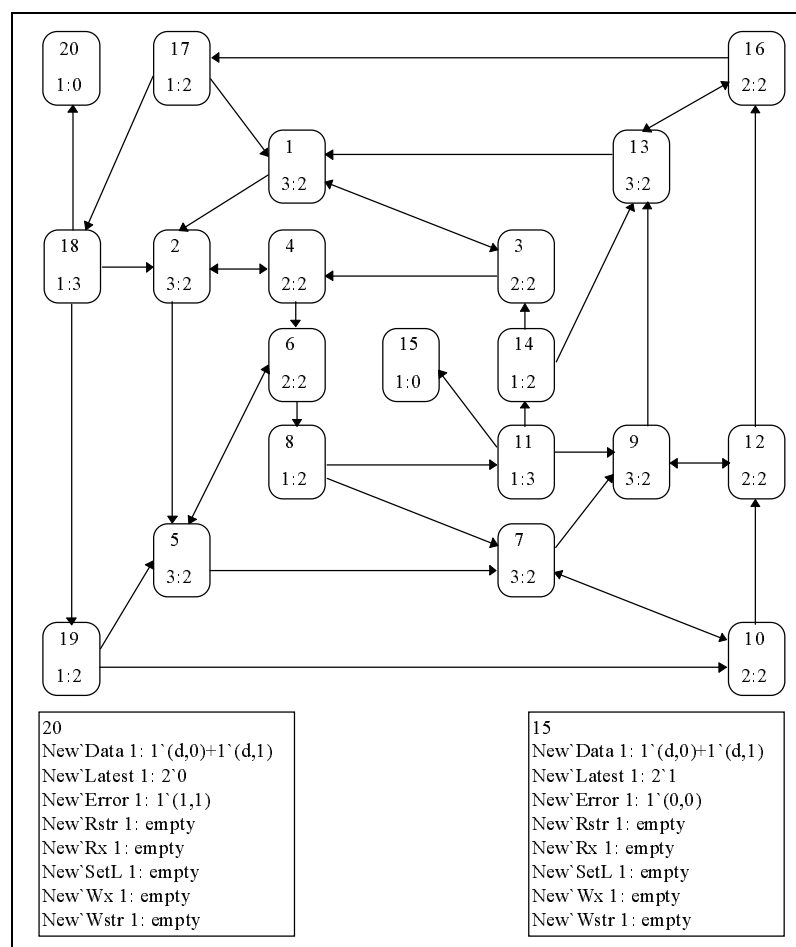


Figure 10 Occurrence graph of the Two-slot mechanism with error detection

Data coherence error is violated by the writer process. When it executes faster than the reader, it wants to write a new data record, at the same time, the reader did not finish or about to start reading the old data record. This situation is highlighted by the token values of both violation states (15 and 20), the value of the control variable *latest* is changed by the writer to have the same value as the reader was using to enter its reading cycle. For example, let us investigate the state number 20. The reader starts its cycle by accessing the control variable as $latest=1$, the write just finished writing its data record, and proceeds to exchange the bit value of the

control variable *latest* to be *latest=0*. Then it will try to write a new data record, but the reader is not through yet. This scenario can be seen from the token value of the place *Latest* (token of 2`0 means *latest=0*) and the token value of the place *Error* (token of 1`1,1) means $w_x=r_x=1$).

There is a good way to cross check the validity of the CP-net model proposed, by using the boundedness property of CP-net. As stated previously in the properties list, the CP-net model must reflect the software design. We can see that the upper multi-set bounds of all the places can never exceed the maximum number of token limits. In other words, the place *Latest* can either have 2`0 or 2`1 and the place *Rx* can either have 1`0 or 1`1. Therefore, the CP-net is precisely reflecting the control variables use.

Finally, we conclude that the Two-slot communication mechanism is conditional asynchronous, the condition would be to assure the speed of accessing and utilising the data record by the reader to be equal or faster than the writer. Let us say the time it takes the reader to access the data record is R_{access} and to utilise it is $R_{between}$ (between access), and similar for the writer. Also, we have the two functions to return the maximum and minimum time of these accessing and utilising operations. Then the condition would be :

$$\text{Min}(W_{access}) + \text{Min}(W_{between}) \geq \text{Max}(R_{access}) + \text{Max}(R_{between})$$

6. The Four-slot Communication Mechanism

```

mechanism four-slot;
type bit = 0..1;
var data : array[bit,bit] of TypeData := ( (null,null), (null,null) );
    s : array[bit] of bit := (0, 0);
    latest, r : bit := 0, 0;

procedure write(item : TypeData);
var wp, wx : bit;
begin
    wp := NOT r;
    wx := NOT s[wp];
    data[wp,wx] := item;
    s[wp] := wx;
    latest := wp;
end;

function reader : TypeData;
var rp, rx : bit;
begin
    rp := latest;
    r := rp;
    rx := s[rp];
    read := data[rp,rx];
end;

end.

```

Figure 11 The Four-slot algorithm (in Pascal like language)

Figure 11 shows the Four-slot software design. The mechanism consists of four data slots, control variables, and two access algorithms. The Four-slot software design can be described as follows;

1. A four slots array $data[bit, bit]$ to hold data in transit, organised as two pairs of two slots. It is pre-set to *null*, in-order to ensure that a read before the first write will obtain the *null* value.
2. A control variable array $s[bit]$ of two elements. Updated by the writer to indicate the index of the slot which contains the last written data, it is pre-set to zero.
3. A control variable *latest* to indicate the last written slot pair, it is pre-set to zero.
4. A control variable *r* to indicate the pair about to be, being, or last read. It is pre-set to zero.
5. Write access algorithm selects the pair of slots that are not being, or to be read. Then, it selects alternate slots within the previously selected pair. Finally it declares the position of the latest written data.
6. Read access algorithm selects the pair of slots that contain the freshest declared data, then it reads the last written data within that pair.

The methodology will follow the same pattern already established, where the author will be using Place/Transition nets, then Coloured Petri nets to model the Four-slot mechanism. The properties established earlier would be preserved, in-order for the Petri net model to reflect the mechanism behaviour under investigation. Each mechanism cycle (reading and writing) will be modelled separately.

6.1 Place/Transition net model of the Four-slot Mechanism

The Place/Transition net model of the reading cycle of the Four-slot is presented in Figure 12. The place *Start-Reading* marks its entry point, and the initial marking is based on the initialisation values defined by the software design. You can see the equivalence between the software design statement execution and the transition firing of the PT-net model.

This algorithm uses the control variable *latest* to steer it towards the slot pair which holds the freshest data. The use of the control variable *r* is more complicated, since the reader process is responsible for updating its value. In-order to reflect this use as required by the previously defined properties list, we need to set the correct bit value for the *r* control variable. When the place $latest=0$ has a token, it implies that the control variable bit value is zero (0), and the same goes with the place $r=0$. If the software design statement $r=rp$ executes (see Figure 11), then the PT-net model must set the bit value of *r* to be the same as *rp*. This means, if $rp=0$ and $r=0$ the bit value of *r* does not change. On the other hand, if $rp=0$ and $r=1$ the bit value of *r* must be exchanged with $r=0$. This way the PT-net model is reflecting the use of the control variables.

Figure 13 shows the PT-net model of the Four-slot write access algorithm, where the place *Start-Writing* sets the entry point. The mechanism use of the control variables is more complicated here. The control array *s* is used to point to the latest written slot within the corresponding pairs. This means $s[0]$ points to the pair $data[0,0]$ and $data[0,1]$, but the value of $s[0]$ points to the last slot used within those two pairs. The PT-net model is reflecting this use by exchanging the value of $s[0]=0$ and $s[0]=1$. The use of the *latest* control variable is reflected by either keeping it unchanged when $wp=latest$, and exchanging it when $wp<>latest$.

Therefore, all the properties defined previously are being encapsulated within these Petri net models, which makes them correct models for the reader and writer processes.

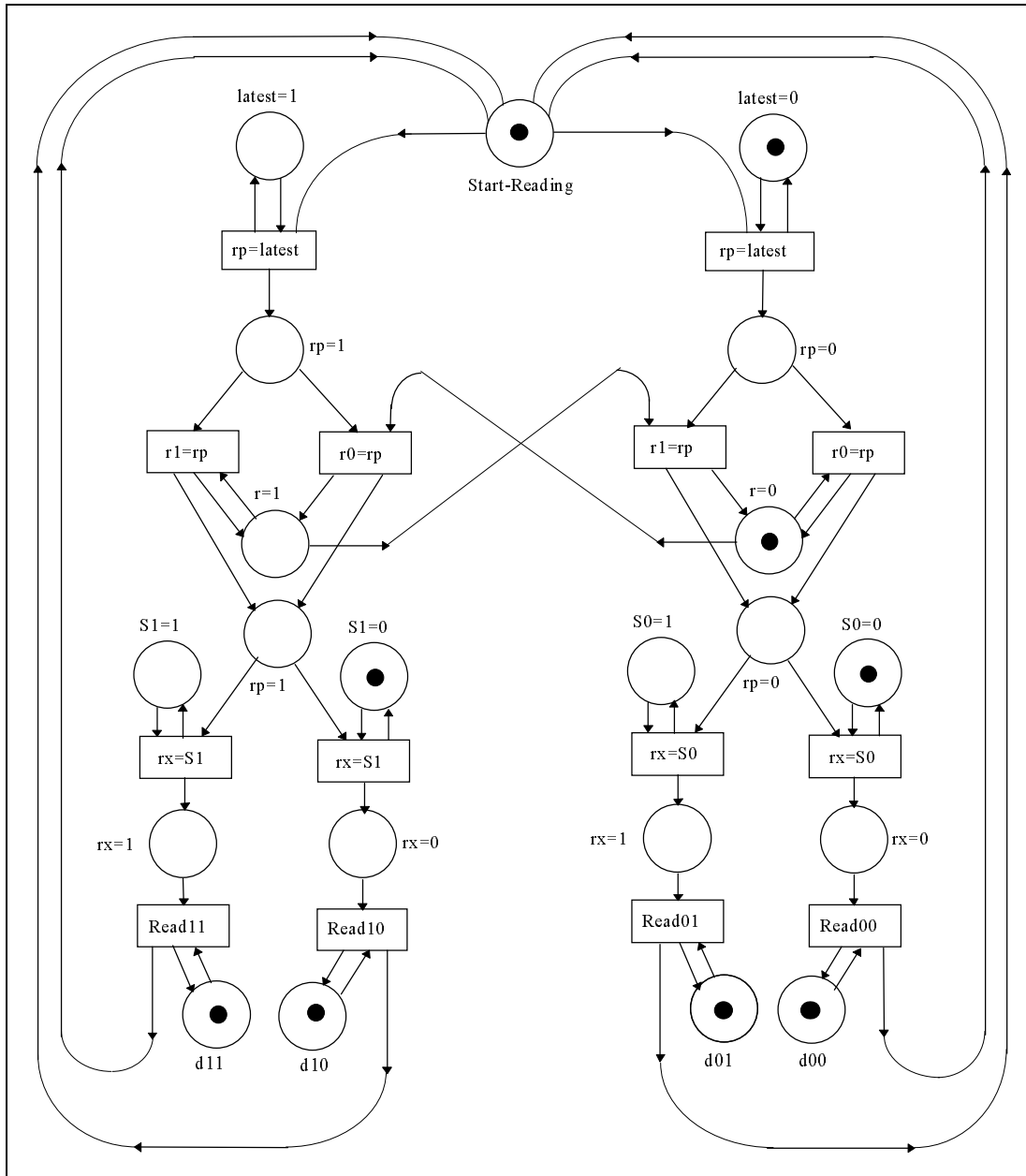


Figure 12 PT-net model for the reading cycle of the Four-slot mechanism

In Figure 12 the reading cycle steering strategy is as follows;

1. The writer steers the reader to the pair having the latest completely written data by means of the control variable *latest*. The selected pair becomes the reader currently used pair.
2. The reader steers the writer away from the pair it is currently using, by means of the control variable *r*.
3. The writer steers the reader to the slot within the reader currently used pair which has the freshest data, by means of the control variable *s[latest]*.
4. The reader access the freshest data slot, and re-starts its reading cycle

The reader will keep reading the same slot (within the currently selected pair), until the writer steers the reader to a pair having the new freshest data.

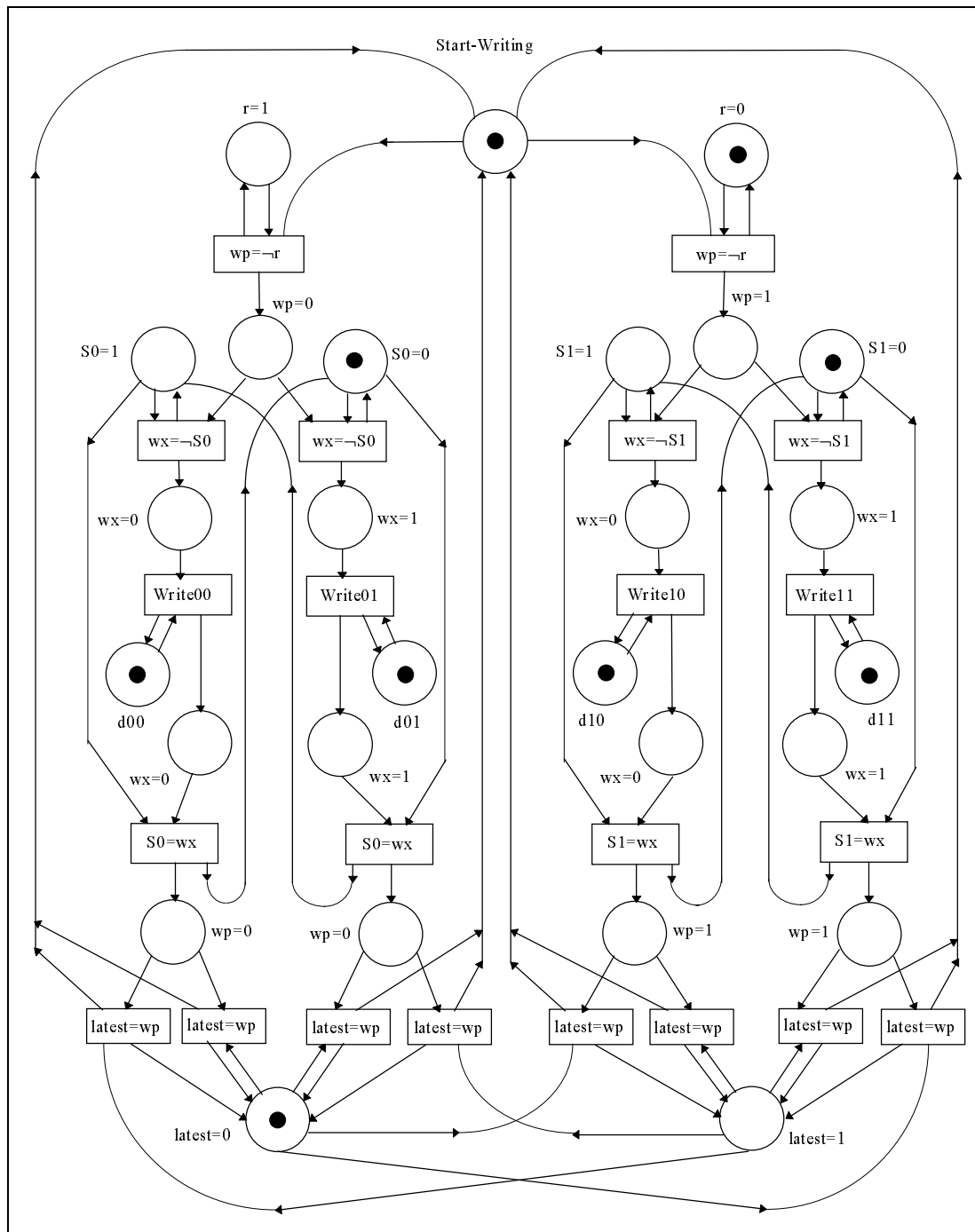


Figure 13 PT-net model for the writing cycle of the Four-slot mechanism

In Figure 13 the writing cycle steering strategy is as follows;

1. The writer avoids the reader currently used pair and chooses the free pair.
2. The writer chooses the slot having the oldest data within the free pair to write its new data.
3. When the writer has completely written its data, it indicates the new freshest data location:
 - The slot which holds the new freshest data (by the control variables $s[latest]$).
 - The pair which holds the new freshest data (by the control variable $latest$).

Afterwards, the writer re-starts its writing cycle.

The Two-slot mechanism is a one way steering strategy (where the reader keeps chasing the freshest data), but it can be seen from the previous description that the Four-slot mechanism is a two way steering strategy. The writer steers the reader towards the freshest slot, and the reader steers the writer away from the pair it is using. In other words, the reader keeps chasing the freshest data and the writer keeps avoiding the pair used by the reader.

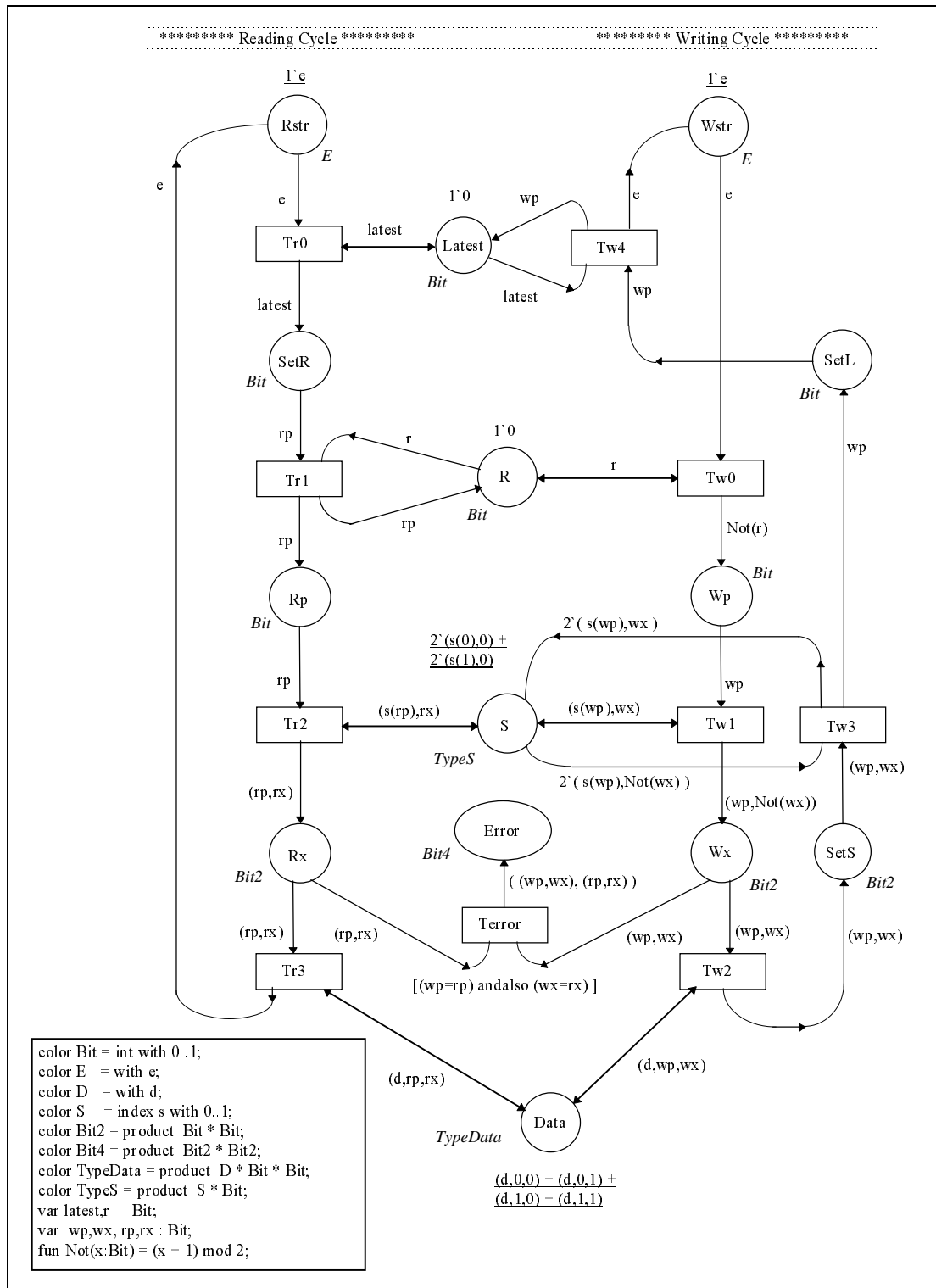


Figure 14 CP-net model of the Four-slot mechanism with error detection

6.2 Coloured Petri net model of the Four-slot Mechanism

The Four-slot mechanism could be constructed directly from the software design, but the author is trying to stress on the power of CP-net modelling and how it simplifies the resulting model into a manageable one. It also provides a good vehicle for verification.

The PT-net model has a big advantage in presenting the analysis, since it reveals all the hidden information about the mechanism behaviour. The CP-net model does not clearly describe the underlying steering strategy used by the mechanism. Therefore, both modelling techniques (Place/Transition nets and Coloured Petri nets) have their own advantages.

There exist two levels of folding within each access algorithm. The first level is the outer subnet, and the second level is the inner subnet. Figure 14 presents the resulting CP-net model.

6.3 Verification of the Four-slot Communication Mechanism

As shown in Figure 14, the proposed error detection facility will halt the execution of the CP-net model, when data coherence violation is committed.

Figure 15 shows the Design/CPN statistical report of the Four-slot mechanism. The occurrence graph generated is full, but unfortunately it is big in size (totals to 576 nodes), so no attempt will be made to draw it. Full state space is required to verify the mechanism data coherence property. By checking both the liveness and boundedness properties of the CP-net model, we can conclude that data coherence error can never occur. The liveness property indicates that the CP-net has no dead marking, so data coherence error can never be reached. Also, the best upper multi-set bounds of the boundedness property, shows that the place *Error* has a maximum marking value to be *empty*. The dead transitions instances shows that the transition *Error* can never occur. All of these are proving that the data coherence property can never be violated.

Finally, we conclude that the Four-slot communication mechanism is fully asynchronous, since both processes, the reader and writer, do not have to wait for each other or to restrict their timing to access and utilise the data in-order to maintain data integrity.

Statistics	

Occurrence Graph	
Nodes:	576
Arcs:	1152
Secs:	5
Status:	Full
Boundedness Properties	

Best Upper Multi-set Bounds	
NewData 1	$1'(d,0,0) + 1'(d,0,1) + 1'(d,1,0) + 1'(d,1,1)$
NewError 1	empty
NewLatest 1	$1'0 + 1'1$
NewR 1	$1'0 + 1'1$
NewRp 1	$1'0 + 1'1$
NewRstr 1	$1'e$
NewRx 1	$1'(0,0) + 1'(0,1) + 1'(1,0) + 1'(1,1)$
NewS 1	$2'(s(0),0) + 2'(s(0),1) + 2'(s(1),0) + 2'(s(1),1)$
NewSetL 1	$1'0 + 1'1$
NewSetR 1	$1'0 + 1'1$
NewSetS 1	$1'(0,0) + 1'(0,1) + 1'(1,0) + 1'(1,1)$
NewWp 1	$1'0 + 1'1$
NewWstr 1	$1'e$
NewWx 1	$1'(0,0) + 1'(0,1) + 1'(1,0) + 1'(1,1)$
Best Lower Multi-set Bounds	
NewData 1	$1'(d,0,0) + 1'(d,0,1) + 1'(d,1,0) + 1'(d,1,1)$
NewError 1	empty
NewLatest 1	empty
NewR 1	empty
NewRp 1	empty
NewRstr 1	empty
NewRx 1	empty
NewS 1	empty
NewSetL 1	empty
NewSetR 1	empty
NewSetS 1	empty
NewWp 1	empty
NewWstr 1	empty
NewWx 1	empty
Liveness Properties	

Dead Markings:	None
Dead Transitions Instances:	
NewTerror 1	

Figure 15 Design/CPN (partial) statistical report for the Four-slot

7. Conclusion

This paper presents a novel Petri nets modelling method for verifying the correctness of interprocess communication mechanisms of the MASCOT Pool IDA. Also, the modelling method provides a clear description of the mechanism behaviour. The Two-slot and Four-slot communication mechanisms have been analysed and verified using Place/Transition nets, Coloured Petri nets, and Design/CPN. This presents a formal correctness verification of both the Two-slot and Four-slot communication mechanisms used by the MASCOT Pool IDA.

The method developed can be used as a tool for analysis of real-time systems such as MASCOT networks. Fundamental to the method is the definition of the properties which provide the basis for Petri nets modelling. Coloured Petri nets and the support of Design/CPN make up a very powerful modelling and analysis tools. The Coloured Petri nets dynamic nature can be used to follow the path of execution of a real-time system, without the cost and effort of building a test system. Future work will aim at establishing the properties of a simple real-time system represented by a MASCOT network, and then utilising Coloured Petri nets with the support of Design/CPN to analyse the system.

8. References

1. Simpson, H., 'Four-slot fully asynchronous communication mechanism', IEE proceedings on Computers and Digital Techniques, Jan 1990, Vol 137 No 1, pp. 17-30.
2. Jensen, K., 'Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use-Volume 1', Second Edition, Springer, 1996.
3. Meta Software Corporation, 'Design/CPN Tutorial for X-Windows - Version 2.0', Cambridge MA, USA, 1993.
4. Meta Software Corporation, 'Design/CPN Reference Manual for X-Windows - Version 2.0', Cambridge MA, USA, 1993.
5. University of Aarhus, 'Design/CPN Occurrence Graph Manual Version 3.0', Aarhus, Denmark, 1996.
6. British Aerospace Dynamics, 'The Official Handbook of MASCOT (Version 3.1)', Royal Signals and Radar Establishment, June 1987.
7. Simpson, H., 'The MASCOT Method', Software Engineering Journal, May 1986, pp.103-120.
8. Simpson, H., 'A Data Interaction Architecture (DIA) for real time embedded multiprocessor systems', Proceedings of RAe Conference on Computing Techniques in Guided Flight, April 1990.
9. Bennett, S., 'Real-Time Computer Control - An introduction - Second Edition', Prentice Hall, 1994.
10. Peterson, J.L., 'Petri net theory and the modelling of systems', Prentice Hall, 1981.