Complementing role models with Petri nets in studying asynchronous data communications

Fei Xia[1] and Ian Clark[2]

[1] Department of Computing Science, University of Newcastle upon Tyne, Claremont Tower, Claremont Road, Newcastle upon Tyne, NE1 7RU, UK
fei.xia@ncl.ac.uk
[2] Department of Electronic Engineering, King's College, University of London, The Strand, London, WC2R 2LS, UK
ian.clark@kcl.ac.uk

Abstract. Simpson's role model method was designed for the analysis of synchronisation-free data communication mechanisms employing shared memory and has been shown to be especially useful for the representation and analyses of data freshness properties. Previously published analyses using the role model method have employed proprietary state space search techniques developed by Simpson. In this paper, a formal definition of role models is given and a way of representing role models using Petri nets is presented. Potential advantages of analysing systems using the role model method complemented with Petri net techniques are demonstrated with a case study of data freshness properties of a data communication algorithm.

Introduction

The use of fully asynchronous processes is advantageous in many hard real-time distributed computer systems. For instance, the complete elimination of time interference in data communications between concurrent processes makes it possible to accurately predict the temporal progress of each process in the system because the timing of each one is completely independent. In certain safety critical systems it may also be required that a process cannot be temporally connected to any other processes and must progress on its own pace. In the real time software design method MASCOT, the pool type IDA accommodates the possibility of a complete lack of synchronisation between the reading and writing processes [1, 2].

The class of data communication mechanisms described and studied by Simpson [3] are designed for transferring data between completely synchronisation-free processes. The mechanisms employ from one to four data areas (called *slots*), implied to be in memory shared between the writer and the reader. Some of the mechanisms have logic in the form of control variables to steer the writer and reader to prevent them from accessing the same slot simultaneously and to cause the reader to obtain the most up-to-date data provided by the writer. The reader and writer are not subject to any relative timing constraints and thus are completely temporally independent without the need for synchronisation, wait states or arbitration in the conventional, request/grant, sense. The general organisation of these types of mechanisms is shown in Fig 1. They will be referred to as *slot mechanisms* in this paper.

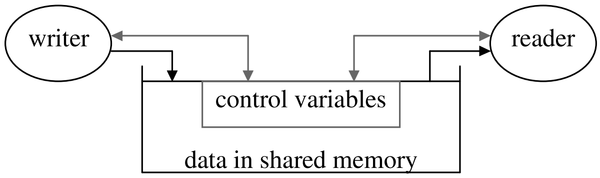

Fig 1 Schematic of slot mechanisms using shared memory and possibly control variables.

It is vital, especially in view of the lack of synchronisation, for the slot mechanisms to be proved to pass on data that is both coherent and fresh. Although other solutions to the problem exist [4, 5], Simpson was the first to show a solution using four slots arranged in two pairs to maintain data coherence and data freshness, having found that up to three slots arranged linearly cannot maintain data coherence [3, 6]. In order to prove the data coherence and data freshness properties of the slot mechanisms, Simpson introduced the role model method [7, 8], which is a novel notation describing discrete state transitions to assist a process of exhaustive reasoning and state space search. A proprietary technique, employing what are called *transition diagrams*, which is similar to reachability analysis in Petri nets [9], was proposed by Simpson for the analyses of role models [7].

As a result of the novel nature of the role model method there have been no supporting studies using it for analysis published by other researchers. To date, there has been no attempt to formally define role models. In addition, although it has been claimed that automated, computerised analyses have been carried out based on transition diagrams [8], no software packages by other sources exist by which any analysis result using Simpson's software may be verified.

It has been shown that Petri net models of the slot mechanisms can be obtained and that the loss of data coherence, as it is signified by a relatively simple state (that of the reader and writer accessing the same slot at the same time), can be checked for with the help of the Petri net models in a straight forward manner [10, 11]. However, as data freshness is affected not only by simple states but also by the trajectories of states, the role model method is more useful for its analysis [10, 12, 13].

In this paper, the role model method is defined in a formal manner. A method of deriving Petri net representations for role models is then given. The intention is to replace or supplement the transition diagram analysis technique with the analysis techniques available for Petri nets to realise more automated and verifiable analyses of asynchronous communication systems. An example is provided where the data freshness properties of a four-slot mechanism algorithm is investigated.

This paper should be read in conjunction with [3], [7] and [8].

Role models

Although the role model method was "developed specifically to deal with the [slot mechanisms] in which small shared control variables are used to co-ordinate access to shared data" [8], it may be extended to cover fairly general modelling of bounded discrete event systems.

A role model consists of a set of agents A, a set of events E, and a set of roles R. The agents are capable of assuming roles. A complete description of the role(s) each agent is assuming as well as a specification of which events are allowed to occur next constitute the current state. States may change only when events occur.

Definition 1 Role model basics

A role model is a tuple, $\mathrm{M} = (\mathrm{A}, \mathrm{E}, \mathrm{R}, \mathrm{S}, \mathrm{s}_0)$.

$\mathrm{A} = \{\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_n\}$ is a finite set of *agents* (implied but not formally described in [7] and [8]), $n \geq 0$. $\mathrm{R} = \{\mathrm{r}_1, \mathrm{r}_2, \ldots, \mathrm{r}_m\}$ is a finite set of *roles*, $m \geq 0$. $\mathrm{E} = \{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_l\}$ is a finite set of *events*, $l \geq 0$. The sets of agents, roles and events are disjoint with one another, $\mathrm{A} \cap \mathrm{R} = \varnothing$, $\mathrm{A} \cap \mathrm{E} = \varnothing$, $\mathrm{E} \cap \mathrm{R} = \varnothing$.

An agent may *assume* a number of roles (known as the *role pattern* of the agent [7]). An event may or may not be permitted to *occur*. A *state*, $\mathrm{s}_i \in \mathrm{S}$, is a complete description of the current role pattern of each agent together with an *event pattern* that describes the events permitted to occur next (implied but not formally described in [7] and [8]). Such sets of patterns are generally known as *pattern expressions* or in short *expressions*.

An event's occurrence may *modify* a state to produce another one. An event e_j, $j \in [1, l]$, is therefore a *state transition function*, a mapping from states to states, $\mathrm{e}_j \in \mathrm{E}: \mathrm{S} \times \mathrm{S}$, and is the only provision by which states may be changed.

If multiple events are permitted to occur by a state, which one occurs next is not deterministic. Events occur according to interleaving semantics, in other words, at most one event occurs at any time, because "events are regarded as distinct" [8].

The dynamic progression of the model consists of the occurrences of events from the *initial state* $\mathrm{s}_0 \in \mathrm{S}$. ▲▲

Definition 2 Role and event patterns, static description

A *static description role pattern*, $p_\mathrm{R} \subseteq 2^\mathrm{R}$, is a subset or a group of subsets of R. A simple way to express it is by writing the names of the *member roles* together in a continuous string. For instance, $r = \mathrm{r}_j\mathrm{r}_k$ is a static description role pattern, if r_j and r_k are role names and $\mathrm{r}_j \neq \mathrm{r}_k$. Agent a_i assuming the roles of r_j and r_k may be written as $\mathrm{a}_i\mathrm{r}_j\mathrm{r}_k$ or $\mathrm{a}_i r$ if $r = \mathrm{r}_j\mathrm{r}_k$.

A *static description event pattern*, $p_\mathrm{E} \subseteq 2^\mathrm{E}$, is a subset or a group of subsets of E. It is expressed in the same way as a role pattern but is not associated with an agent.

Sometimes it is not necessary to specify the presence and absence of every possible role or event in a pattern. In this case pattern operators may be employed.

The *asterisk operator* '*' is used in static description patterns to denote any roles or any events. In this case, *a** means "*a* with any roles, including no role", *ar** means "*a* with *r* and any (including no) other roles", and *e** means "an event pattern including *e* with any other events".

The *null operator* '0' is used in static description patterns to denote an empty pattern. Thus *a*0 means that *a* assumes no role, while 0 denotes an empty event pattern. The null operator always forms a complete pattern by itself. ▲▲

Definition 3 Role and event patterns, modification description

Let a be an agent, r_j be a role, and e_k be an event.

A *modification description role pattern* is a number of distinct *role modifiers* written in a continuous string. A role modifier is either an asterisk operator, a null operator, a *bar operator* with a role *operand*, or an *addition operator* with a role operand.

A *modification description event pattern* is a set of distinct *event modifiers* written in a continuous string. Event modifiers are similar to role modifiers but pertain to events, rather than roles.

In modification description patterns the asterisk operator indicates no change. Therefore, $a*$ means "no change to a's roles" and * in place of an event pattern means "no change to the event pattern".

In modification description patterns the null operator indicates the "removal of all roles" or the "removal of all events" from the relevant pattern.

The bar operator '$^{-}$' takes a single operand which is a role or event written below it. It indicates the action of "removing if present" on the operand. Thus $a\,\overline{r_j}$ means that the role r_j is to be removed from a and $\overline{e_k}$ means that the event e_k is to be removed from the event pattern. No action is taken if the operand specified to be removed by the bar operator is not in the current pattern.

The addition operator '+' also takes one operand which is a role or event written to its right. It indicates the action of "adding if not present" on its operand. Thus a+r_j means that r_j is to be added to the role pattern of a and +e_k means that e_k is to be added to the event pattern. No action is taken if the role or event specified to be added is already in the current pattern. When it is not confusing to do so, the addition operator may be omitted from the notation, leaving its operand to describe the modification, i.e. in a modification description pattern, ar_j is the same as a+r_j.

The same role or event name may not appear as the operand of both the bar and addition operators in the same modification description pattern. For instance, $a+r_j\overline{r_j}$ and $ar_j\overline{r_j}$ are not valid patterns. ▲▲

Definition 4 Pattern equality and compatibility

Two static description patterns p_x and p_y are said to be equal to each other iff they are both role or event patterns and every role or event is specified present or absent the same way in both p_x and p_y.

Two static description patterns p_x and p_y are said to be compatible with each other iff they are both role or event patterns and no role or event is specified present or absent in conflicting ways in p_x and p_y. This relation is denoted by the symbol '~'. The complement relation, incompatibility, is denoted by '/~'.

For instance, $ar* = ar_jr_k*$, iff $r = r_jr_k$, and $a* \sim ar$ for any r. ▲▲

Obviously, both compatibility and equality are symmetric relations. Some examples of pattern compatibility are given below:

> $[r_1r_2r_5]$ is compatible with {*}, $[r_1*]$, $[r_2*]$, $[r_5*]$, $[r_1r_2*]$, $[r_1r_5*]$, $[r_2r_5*]$ and $[r_1r_2r_5]$ but no other pattern.

$[r_2r_5*]$ is compatible with $\{*\}$, $[r_2*]$, $[r_5*]$, $[r_2r_5]$ and $[r_2r_5*]$ only.
$\{*\}$ is compatible with any pattern.
$[0]$ is compatible with $[0]$ and $\{*\}$ only.

The significance of different types of brackets is given in Definition 5.

Definition 5 Pattern expressions

A pattern expression (or expression in short) is a set of patterns. Expressions are written in the conventional list style, i.e. member patterns separated by comas.

There is at most one role pattern for each agent of the model, as well as at most one event pattern, in any expression. The simplest non-empty expression is a single pattern with one member.

An expression and/or some of its member patterns may be enclosed in (), [], or {}. In general, square brackets indicate that the expression describes a subset of all possible patterns and curly braces indicate that the expression describes all possible patterns within its scope. These and parentheses are used to discriminate between parts of expressions.

Member patterns of any single expression may either be static description ones or modification description ones. They may not be a mixture of both. An expression is therefore either a *static description expression* or a *modification description expression*, but not partly both. The following are example expressions:

$$[a_1 0, (a_2\,\overline{r_4 r_5}\,, a_3{+}r_4, a_4*), a_5{+}r_1, +e_3\,\overline{e_1 e_2}\,]$$
$$[a_1 r_4 r_5*, \{a_2*, a_3*, a_4*\}, a_5 0, e_1 e_3]$$
$$\{*\}$$

where a_i, $i \in [1,5]$, are agents, r_j, $j \in \{1,4,5\}$, are roles, and e_k, $k \in [1,3]$, are events. The parentheses and braces indicate that agents a_2, a_3 and a_4 may represent something distinctive, or are being highlighted for some reason.

A pattern in one expression is said to *correspond* to a pattern in another expression if both are role patterns of the same agent or if both are event patterns.

A static description expression X_x is said to be compatible with another static description expression X_y, denoted by $X_x \sim X_y$, iff every member pattern of X_x is compatible with its corresponding member pattern of X_y. ▲▲

Definition 6 Transition statements, basics

A transition statement consists of an *input expression*, which is a static description expression, followed by an $\rightarrow$ followed by an *output expression*, which is a modification description expression. It describes an *action*, which, when *carried out*, results in a *modification* of the patterns of the input expression as specified by the output expression.

For a role model M = (A, E, R, S) in the form given in Definition 1, any transition statement takes the form

$$T: I_T \rightarrow O_T$$

where $I_T = [p_{ITi},\ i \in [1, n+1]]$ is the input expression which consist of one role pattern for each agent and one event pattern, and $O_T = [p_{OTj}, j \in [1, c_{OT}]]$ is the output expression which consists of a number, $c_{OT} \in [1, n+1]$, of modification description patterns.

For any transition statement T in the form given above, $\forall p_{\mathrm{OT}j} \in \mathrm{O}_{\mathrm{T}}$, $\exists p_{\mathrm{IT}j} \in \mathrm{I}_{\mathrm{T}}$ such that $p_{\mathrm{OT}j}$ and $p_{\mathrm{IT}j}$ correspond to each other. In other words, for any transition statement, every member pattern of its output expression must correspond to a member pattern of its input expression. ▲▲

Definition 7 Transition statements, basic actions

The simplest possible type of transition statements has a single role or event modifier in their output expressions. Such transition statements are known as *basic transition statements* and the actions they describe are known as *basic actions*. The modifications of all possible basic transition statements are given below. These are known as *basic modifications*.

Let *a* be an agent, *r*” be a role pattern, *r*’ and *r* be roles, *e*” be an event pattern and *e* and *e*’ be events.

The transition statement

[*ar*” with any other pattern(s) not relating to *a*] → [*a*+*r*’] (**1**)

describes the modification of adding *r*’ to the role pattern of *a* if $r' \notin r''$. Other patterns in the input expression are not modified (This applies to all (**1**) ~ (**8**)).

The transition statements

[*ar*” with any other pattern(s) not relating to *a*] → [$a\,\bar{r}$] (**2**)
[*ar*” with any other pattern(s) not relating to *a*] → [*a**] (**3**)
[*ar*” with any other pattern(s) not relating to *a*] → [*a*0] (**4**)
[*e*” with any role pattern(s)] → [+*e*’] (**5**)
[*e*” with any role pattern(s)] → [$\bar{e}$] (**6**)
[*e*” with any role pattern(s)] → [*] (**7**)
[*e*” with any role pattern(s)] → [0] (**8**)

describe similar modifications in the same way as (**1**) above, according to the specifications of pattern modifiers given in Definition 3. ▲▲

Definition 8 Transition statement sets and their rationalisation

A finite number *k*, $k > 1$, of transition statements may be collected into a set:

$\{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_k\} = \mathrm{T}$

where T_i, $i \in [1, k]$ is a transition statement, if the input expressions of the *k* transition statements involved are all compatible with one another and the actions of the transition statements involved are considered simultaneous.

A set of transition statements is said to be *rational* if no more than one basic modification is specified for any pattern member. For a rational transition statement set, the combined modification of the set is obtained by carrying out the action of each member statement once, one at a time, i.e. according to interleaving semantics.

Any set of transition statements may be *rationalised* by applying the following operator precedence rule on every pattern member concerned:

The precedence of the four modification pattern operators is defined to be: *, +, ¯, and 0, i.e. the asterisk operator takes effect once if it is specified one or more times, if it is not the addition operator takes effect once if it is specified one or more times, if neither of the previous operators is specified the bar operator takes effect once if it is specified one or more times, and the null operator, if specified one or more times, takes effect once only if no other operator is specified. ▲▲

Definition 9 Transition statements, completion

A transition statement T is defined as follows

$$\mathrm{T}: \mathrm{I_T} \rightarrow \mathrm{O_T} \equiv \{\mathrm{T_1}, \mathrm{T_2}, \ldots, \mathrm{T}_{c\mathrm{OT}}\} = \{\mathrm{I_T} \rightarrow [p_{\mathrm{OT}1}], \mathrm{I_T} \rightarrow [p_{\mathrm{OT}2}], \ldots, \mathrm{I_T} \rightarrow [p_{\mathrm{OT}c\mathrm{OT}}]\}$$

In other words, any transition statement T consists of a set of transition statements T_i, $i \in [1, c_{\mathrm{OT}}]$, c_{OT} being the cardinality of the output expression of T, each of which specifies the modification to one of the member patterns of its input expression.

The action of any transition statement T_i above is defined as follows

$$\mathrm{T}_i: \mathrm{I} \rightarrow [p_{\mathrm{OT}i}] \equiv \{\mathrm{I_T} \rightarrow [\mathrm{m}_{p\mathrm{OT}i1}], \mathrm{I_T} \rightarrow [\mathrm{m}_{p\mathrm{OT}i2}], \ldots, \mathrm{I_T} \rightarrow [\mathrm{m}_{p\mathrm{OT}ik}]\}$$

where every $\mathrm{m}_{p\mathrm{OT}ij}$, $j \in [1, k]$ is the jth member modifier of $p_{\mathrm{OT}i}$, and k is the number of member modifiers in $p_{\mathrm{OT}i}$. In other words, any transition statement on a single pattern consists of a set of basic transition statements, each of which specifies the basic modification to one of the pattern members of its input expression.

Given the above, the overall modification of any transition statement can be obtained by applying the rules on transition statement sets given in Definition 8 and rules on basic modifications given in Definition 7. ▲▲

The action described by one transition statement is therefore considered atomic, i.e. all pattern members of the input expression are modified simultaneously.

Lemma 1 Transition statement decomposition

In general, the modification of any transition statement T is obtained by carrying out the action of each of the modifiers in its output expression once, one at a time.

Proof: From Definition 8, the above is true if T is rational.

According to Definition 5, "there is at most one role pattern for each agent of the model, as well as at most one event pattern, in any expression". It therefore follows from Definition 8 that any transition statement T of the following form

$$\mathrm{T}: \mathrm{I_T} \rightarrow \mathrm{O_T} \equiv \{\mathrm{T_1}, \mathrm{T_2}, \ldots, \mathrm{T}_{c\mathrm{OT}}\} = \{\mathrm{I_T} \rightarrow [p_{\mathrm{OT}1}], \mathrm{I_T} \rightarrow [p_{\mathrm{OT}2}], \ldots, \mathrm{I_T} \rightarrow [p_{\mathrm{OT}c\mathrm{OT}}]\}$$

is rational if every T_i, $i \in [1, c_{\mathrm{OT}}]$, is rational.

By reviewing Definition 2 and Definition 3, it is obvious that any T_i of the above form is indeed rational. ▲▲

The following example transition statement

$$[\mathrm{a_1}*, \mathrm{a_2 r_1}, \mathrm{a_3}0, \mathrm{a_4 r_5}*, \mathrm{e_1}] \rightarrow [\mathrm{a_1}\,\overline{\mathrm{r_4 r_5}}\, +\mathrm{r_1}, \mathrm{a_2}*, \mathrm{a_3}+\mathrm{r_4}+\mathrm{r_5}, \mathrm{a_4}0, 0]$$

describes an action which, when carried out, removes the $\mathrm{r_4}$ and $\mathrm{r_5}$ roles from and adds the $\mathrm{r_1}$ role to $\mathrm{a_1}$, leaves the role pattern of $\mathrm{a_2}$ (which was $\mathrm{r_1}$) unchanged, adds the $\mathrm{r_4}$ and $\mathrm{r_5}$ roles to $\mathrm{a_3}$ (which originally assumed no role), removes all roles from $\mathrm{a_4}$, and removes all events from the event pattern (which was $\mathrm{e_1}$). The resulting patterns are [$\mathrm{a_1}$(with $\mathrm{r_1}$ but without $\mathrm{r_4}$ and $\mathrm{r_5}$, otherwise no change), $\mathrm{a_2 r_1}$, $\mathrm{a_3}$ $\mathrm{r_4 r_5}$, $\mathrm{a_4}0$, 0].

Definition 10 States

A state is described by a *state expression*, which is a static description expression whose role patterns describe explicitly the current roles assumed by all agents in the model and whose event pattern describes explicitly the events permitted to occur next. An example is

$$s = [a_1r_4r_5,\ a_2r_1,\ a_3r_4,\ a_40,\ a_50,\ e_1e_3]$$

for a model M with five agents ($n = 5$).

If a state has the null operator as its event pattern it is a *terminal* one (i.e. no further events may occur) for the model. ▲▲

Since a state expression must be explicit, i.e. with all roles and events specified, it may not contain the asterisk operator.

Definition 11 Event functions

An event e_i is a state transition function, $e_i \in E: S \times S$ which describes the state change when e_i occurs. It consists of p transition statements, p being a finite number, $p \geq 1$. The description of any event e_i is written as

$$e_i\text{: } I_{ei1} \rightarrow O_{ei1},\ I_{ei2} \rightarrow O_{ei2},\ \ldots,\ I_{eip} \rightarrow O_{eip}$$

where I_{eij} is the jth input and O_{eij} is the jth output expression of e_i, $j \in [1, p]$.

All input expressions of e_i must have non-empty event patterns which are compatible with e_i, because the evenp pattern of an expression specifies the events allowed to occur next.

If $e_i \sim s$, and $s \sim I_{eij}$ for some $j \in [1, p]$, s is said to be a valid pre-state of e_i.

An event may occur if the current state of the model is a valid pre-state of it. When an event e_i occurs under a valid pre-state s, the action described by its jth transition statement, $j \in [1, p]$, is carried out iff $s \sim I_{eij}$.

The occurrence of an event is atomic, i.e. the actions described by multiple transition statements, if carried out during one occurrence of an event, are simultaneous, i.e. they may be collected into a set. This is consistent with Definition 4, Definition 5 and Definition 8. ▲▲

Lemma 2 Elements of state transition functions

The effect of any event may be obtained by carrying out a finite number of basic transition statements in the form of (**1**) ~ (**8**) following interleaving semantics.

Proof: The above conclusion follows from Definition 6, Definition 7, Definition 8, Definition 9, Definition 10, Definition 11 and Lemma 1. ▲▲

Lemma 2 does not imply that multiple events in a role model can all be decomposed into basic transition statements whose actions are carried out during analysis in an uncontrolled manner. In fact, because events are atomic, no interleaving is allowed between them. Care must be taken during state space analysis to ensure that once an event has started, it must be finished (i.e. the actions of all of its basic transition statements carried out) before another one can start.

The specific operator precedence rule of Definition 8 makes an event e_i a function, i.e. there is only one possible new state as a result of its occurrence from any pre-state. However, e_i is not assumed to be injective, i.e. the same state may be the outcome of different pre-states if e_i occurs. According to Definition 1, the representation of indeterminism is provided at the event level, i.e., if multiple events are permitted to occur by a state, which one occurs is not deterministic.

Certain simplifications of the conventions of presenting role models are allowed once the physical significance of agents, role patterns and event patterns are clearly understood. Examples of shorthand notation resulting from such simplifications can be found later in the role models for the case study below. It must be noted, however, that any simplified model must be convertible to the full form of Definition 1 ~ Definition 11.

Petri net representations of role models

Since a role model represents a discrete system with a bounded number of potential states (A, R and E being bounded for any known model) whose actions follow interleaving semantics, a subset of what classical Petri nets (P/T nets) can represent [9], it should be possible to find an equivalent P/T net for any role model. Here techniques are developed whereby a safe (1-safe) P/T net representation of any given role model may be found.

Proposition 1 Petri net representation of agents and their role patterns

It is proposed to represent the notion of an agent and its possible role patterns with 1-safe places. An agent assuming a particular role in a pattern is represented by a particular place having the marking 1. An agent not assuming a particular role in a pattern is represented by its corresponding place having the marking 0. Complementary places are employed to avoid the need for inhibitor arcs. This necessitates a potential $2m$ such places for each agent in the model and a maximum of $2m \times n$ such places for all agents. From [7] and [8], it is clear that sometimes not all agents may assume all roles, depending on the physical entities and phenomena they represent. In such cases the number of places needed is reduced from the maximum.

The role pattern of $a_1r_4r_5$, assuming that m=5, is thus represented by the P/T net fragment and marking in Fig 2. The role pattern of a_10 is represented by marking all the (not) places and unmarking all the other places in Fig 2. ▲▲

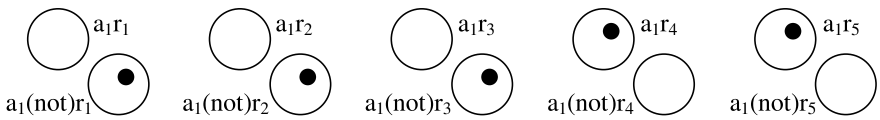

Fig 2 P/T net representation of $a_1r_4r_5$.

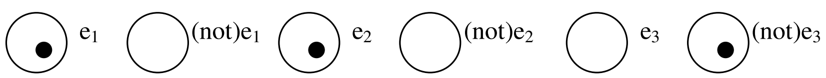

Fig 3 P/T net representation of e_1e_2.

Proposition 2 Petri net representation of event patterns

It is proposed to represent all possible event patterns for a system model using 1-safe places in the same way as in Proposition 1. A total of $2l$ places are required.

Assuming that l=3, the event pattern of e_1e_2 is thus represented by the P/T net fragment and marking in Fig 3 while the event pattern of 0 is represented by marking all the (not) places and unmarking all the other places in Fig 3. ▲▲

Since state expressions may not include the asterisk, bar and addition operators the two propositions above provide for the representation of all possible role model state expressions.

The active elements of a role model (or where things happen) are the events with their state transition statements. The active elements of a Petri net are the transitions. It is then natural to try and use Petri net transitions to represent events.

According to Lemma 2, any event may be decomposed into a number of basic transition statements. It is therefore important to find the correct representations of all possible basic modifications.

Proposition 3 Petri net representation of basic transition statements

Rules are provided here for the representation of statements (**1**) ~ (**8**). Markings in the figures of this proposition generally represent the terminal condition of the transition statements (i.e. after the modifications have been carried out).

The transition statement [*ar*" with any other pattern(s) not relating to *a*] → [*a*+*r*'] is represented by the P/T net fragment in Fig 4. In one action of this statement only one of the transitions in the P/T net fires.

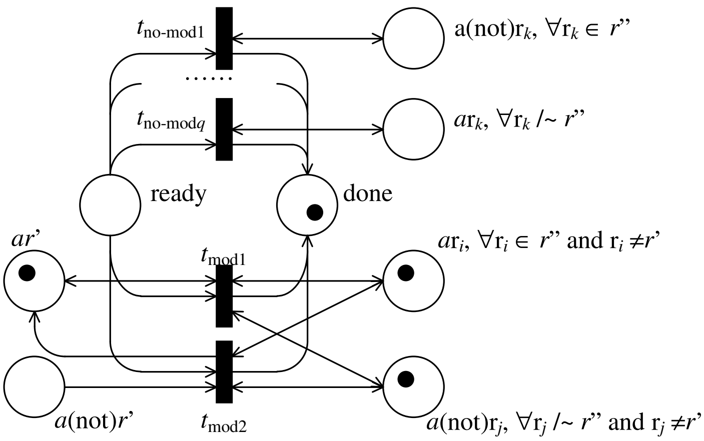

Fig 4 P/T net representation of (1)**.**

There are two types of transitions in this net fragment. One type represents the proper modification according to the basic transition statement and include the transitions t_{mod1} and t_{mod2}. The roles of *a* other than *r*' need to be referenced by these transitions, depending on if and in what way they are compatible with the pattern *r*". If a role is a member of *r*", the place representing it must be marked for t_{mod1} or t_{mod2} to fire. If a role is not compatible with *r*" the place representing its complement must be marked for t_{mod1} or t_{mod2} to fire. If a role is compatible with but not a member of *r*", i.e. it is compatible with *r*" solely because *r*" contains an *, places representing it are not connected with t_{mod1} and t_{mod2} in any way. These connections represent the fact that the input expression of the transition statement must be satisfied (i.e. it is compatible with the present state) for its action to be carried out. The connections between transitions t_{mod1} and t_{mod2} and the place representing *ar*' and its complementary place represent the modification of this transition statement. The markings of these places are either modified or retained depending on whether the role *r*' is assumed by the agent *a* or not before the action.

The "ready" and "done" places are used to control the actions of the net whose significance will become clear later. These places are not explicitly drawn in the other P/T net fragments given below but are assumed to be present. One of the transitions firing in this net fragment takes a token away from place ready and deposits it in place done, and the net fragment only becomes active when ready is marked.

The number of transitions named $t_{\text{no-mod}x}$, $x \in [1, q]$ where q is a finite integer are needed to ensure that the net fragment does not introduce deadlocks artificially, i.e. if place ready is marked, at least one transition between it and place done is enabled. At least one of these "no modification" transitions is enabled if *a* does not assume the pattern *r*". These transitions are also assumed to be present in all the other net fragments in this proposition but are schematically shown here only once.

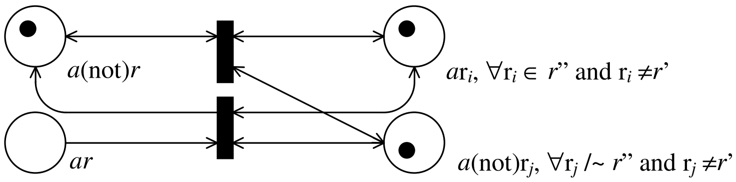

Fig 5 P/T net representation of (2)**.**

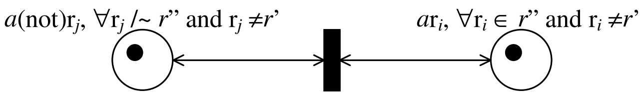

Fig 6 P/T net representation of (3)**.**

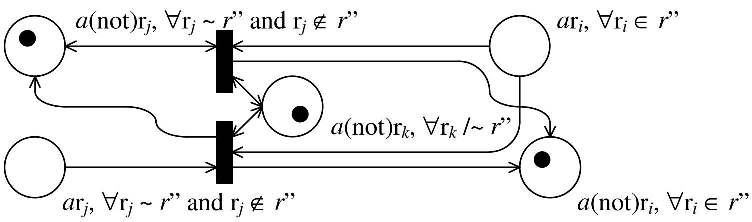

Fig 7 P/T net representation of (4)**.**

The transition statements (**2**) ~ (**4**) are represented by the P/T net fragments in Fig 5 ~ Fig 7. They are obtained using the same techniques outlined above. No no-mod transitions are shown in these figures to save space.

P/T net representations for transition statements (**5**) ~ (**8**) can be obtained in the same way. ▲▲

It is clear from the discussions and definitions presented so far that if a role model consists of a single event, it can be decomposed using the rules given in Lemma 1 and Lemma 2 into basic transition statements, which can in turn be represented by P/T net fragments given in Proposition 3. A full net model consisting of such fragments assembled together, even allowing random firing order but ensuring that each transition is fired once, would produce the correct modification of roles and events of the role model.

For role models with multiple events, however, a P/T net directly assembled from parts built according to the rules given so far would usually not consist of a correct representation of a role model, because events are atomic actions in role models. A technique that helps to capture the notion of event atomicity needs to be developed before events themselves can be decomposed and the resulting basic transition statements represented by the net fragments given in Proposition 3.

Proposition 4 Petri net representation of events

In this proposition, it is assumed that *e* is the event being represented. It is also assumed that the event *e* is rational or has been rationalised, with a finite number *h*, $h \geq 1$ basic transition statements.

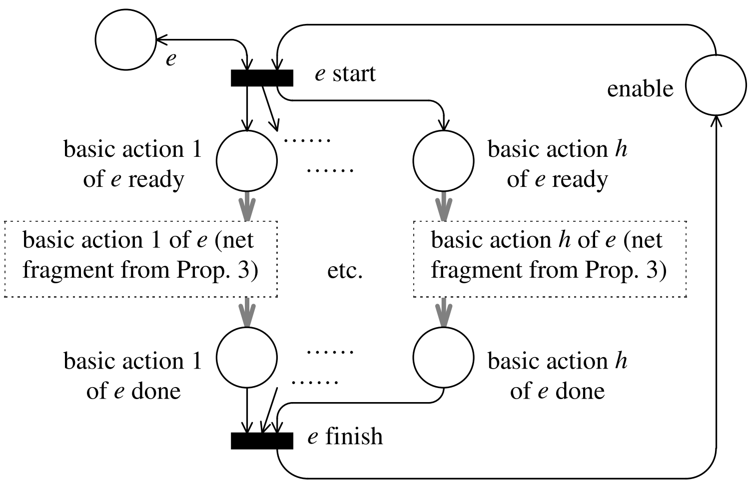

Fig 8 Basic transition statements within an event.

It is proposed that a single place with 1-safe markings be used to manage the atomicity of events on a global basis. The place labelled “enable” in Fig 8 serves this purpose.

The schematic shown in Fig 8 indicates that the start transition for event *e* may fire only when both places *e* and “enable” are marked. The basic actions of any event are represented by net fragments in the style of Proposition 3 and are organised in parallel within the sub-net representing the event, in the way shown in Fig 8. If the event start transition for *e* fires, a token is deposited into the “ready” place of every net fragment representing a basic transition statement of *e*. The transition “*e* finish” may not fire until every basic transition statements of *e* have been carried out once, signified by a token in each of the “done” places involved. The place enable is connected in the same way to all event start and finish transitions, thereby ensuring the atomicity of events within the overall net, i.e. when any event is in progress (between its start and finish transitions), no other event may start. The marking in Fig

8 represent the situation that some other event is in progress and event e is waiting for the places e and enable to be marked. ▲▲

The no-modification transitions (Proposition 3) in the net fragments representing the basic transition statements ensure that there are no artificially introduced deadlocks in a P/T net of the type of Fig 8.

With all the representation techniques introduced so far, it is now possible to find a P/T net representation of any role model by following these steps: 1) Rationalise all events; 2) lay out places representing all possible pattern members using the rules given in Proposition 1and Proposition 2; 3) construct a sub-net representing each event by the technique given in Proposition 4, making use of the rules given in Proposition 3 to obtain representations of the basic transition statements involved; 4) assemble the sub-nets together.

Case study

The most recently proposed four-slot mechanism algorithm in [8] employs one fewer control variable than the one found in [7]. It also has a more streamlined organisation. This new algorithm is used here in a case study to demonstrate the finer points of the role model method and the use of Petri net techniques to supplement the analysis of role models. The algorithm itself, for a data communication mechanism of the type of Fig 1, is reproduced here in Fig 9. The algorithm includes two access procedures, one each for the writer and the reader. Each access procedure is assumed to be imbedded within a larger cycle loop in which data to be communicated is produced or consumed. Depending on the arrangement of the larger writer and reader cycles, the access may be open or limited. Taking the writer process as an example, the writer access procedure is assumed to be part of a cycle loop of "loop, produce data, write access procedure, end loop" if it has limited access. If it has open access, the statement *wr* in the access procedure would include the actions of preparation/production of data, i.e. the data is written as it is produced. In this algorithm, the reader and writer procedures are regarded as sequential within themselves, but arbitrary interleaving is allowed between the two processes. Thus it provides for complete temporal independence of the two sides and has been described as "fully asynchronous" or "synchronisation free".

Writer	Reader
wr: $d[ip, w[ip]] := input$	$r0$: $r := w$
$w0$: $ip := \overline{ip}$	$r1$: $op := \overline{ip}$
$w1$: $w[ip] := \overline{r[ip]}$	rd: $output := d[op, r[op]]$

Fig 9 Four-slot data communication mechanism algorithm.

Simpson applied role model based analysis on the algorithm in Fig 9 and verified its data coherence, data freshness and data sequencing properties [8]. Definitions of these properties, obtained from [7] and [8], are given below.

Definition 12 Data coherence

Data coherence is maintained if the reader and writer never access the same slot at the same time, i.e. whenever *wr* and *rd* overlap in time, $[ip, w[ip]] \neq [op, r[op]]$. ▲▲

The statements *w*0 and *w*1 constitute the writer post sequence while the statements *r*0 and *r*1 constitute the reader pre-sequence. During these sequences the writer and the reader provide information to and receive information from each other. Both following (so that the reader tends to read the latest data the writer has just provided) and avoidance (so that they tend to not access the same slot at the same time) are performed in these sequences with the help of control variables such as *w*, *r*, and *ip*.

Definition 13 Latest and previous latest data

At any time, the latest piece of data is found in the slot accessed by the last completed *wr* statement, and the previous latest piece of data is found in the slot accessed by the last but one completed *wr* statement. ▲▲

Definition 14 Data freshness

Data freshness is maintained if the reader always accesses an acceptable slot. Acceptable slots are:

1. the slot containing the piece of data which was the latest at the beginning of the pre-sequence of the current reader cycle, $slot_L$;
2. the slot containing the piece of data which is the previous latest at the beginning of the pre-sequence of the current reader cycle, $slot_P$, only if the pre-sequence of this reader cycle started while a writer post sequence is in progress;
3. any slot accessed by a completed *wr* statement that happened later than the one that accessed $slot_L$ before the beginning of the current reader pre-sequence. ▲▲

In Definition 14, 1. is intuitive, as the reader access procedure uses the pre-sequence to determine where the latest (called “freshest” by Simpson [7, 8]) piece of available data is, and a successful attempt at obtaining this piece of data, even in the event of newer data becoming available during the reader pre-sequence, cannot be regarded as a data freshness failure. In comparison, 3. is meant to cover the case in which the reader is after all able to obtain a piece of data that is newer than the piece of data that was the latest at the beginning of the reader pre-sequence, by design or by accident. It is recognised in 2. that if a writer post sequence and the pre-sequence of the current reader cycle overlap in time, the reader should not be expected to obtain the location of the latest piece of data available at the beginning of its pre-sequence because its location may not have been completely indicated by the writer post sequence yet. However the reader should be expected to at least obtain the location of the piece of data that is the previous latest at the start of its pre-sequence in this case, as that should have been indicated by the post sequence of the previous writer cycle.

Definition 15 Data sequencing

Data sequencing is correct (correctly maintained) iff the reader does not access any data older than the piece of data it accessed during its previous cycle. This implies that for any two pieces of data d_1 and d_2 obtained by the reader in the order of d_1, d_2, they were written into the mechanism by the writer also in the order of d_1, d_2. ▲▲

Role models were developed to analyse the data coherence, freshness, and sequencing properties for the four-slot algorithm in Fig 9 by Simpson [8]. A P/T net model of the algorithm, based directly on the algorithm statements rather than Simpson's role models, using the techniques discussed in [10] and [14], has been developed and the result of analysis based on this model confirms the findings of [8] that the algorithm maintains data coherence in all possible interleaving situations of the reader and writer. The direct P/T net model is capable of revealing all operating states and sequences of states of the algorithm via a reachability search. By definition this information covers any property of the algorithm including data freshness and data sequencing. However, the extraction from the reachability analysis of relevant information describing these two properties is not straight forward. It is more convenient to employ special role models highlighting these properties, obtain P/T net representations of these models, and reveal the properties by analysing the P/T net models.

Simpson used additional roles and transition statements to the events of the main role model which covered data coherence issues for the analyses of data freshness and sequencing. The additional data freshness transition statements for the algorithm in Fig 9 are reproduced here in Fig 10. Here the agents are the pairs of slots and the slots within each pair. From Fig 9 it can be seen that these are pairs 0 and 1 and slots 00, 01, 10, and 11, with the data item being stored in slot *ij* identified by "*data*[*i*, *j*]". The roles used in this particular model for data freshness are all relevant only to slot agents, with the positions reserved for pair agents always marked * (The pair agents and their roles are used by Simpson in the analysis of data coherence properties and are not of interest here.). Since it is understood that the writer and the reader are both cyclic sequences of statements which may interleave each other in every manner possible, event patterns are entirely omitted in the model of Fig 10, with their function being served by the arrangement of event transition statements. The writer events progress in a cycle of *wip*, *wcs*/*wis*, and the reader one in a cycle of *rcs*/*ris*, *rcp*.

Writer:

wip: $[\{(*), F*, *\}, \{*\}] \rightarrow [\{(*), P, *\}, \{*\}],$

$[\{(*), W*, *\}, \{*\}] \rightarrow [\{(*), VF, \overline{F}\}, \{(*), \overline{F}, \overline{F}\}]$

wis: $[\{*\}, \{*\}] \rightarrow [\{(*), \overline{P}, \overline{P}\}, \{(*), \overline{P}, \overline{P}\}]$

Reader:

rcs: $[\{(*), P*, *\}, \{*\}] \rightarrow [\{(*), p, \overline{p}\}, \{(*), \overline{p}, \overline{p}\}],$

$[\{(*), F*, *\}, \{*\}] \rightarrow [\{(*), f, \overline{f}\}, \{(*), \overline{f}, \overline{f}\}],$

$[\{*\}, \{*\}] \rightarrow [\{(*), \overline{V}, \overline{V}\}, \{(*), \overline{V}, \overline{V}\}]$

Fig 10 Role model transition statements for data freshness of the algorithm in Fig 9.

In other words, the semicolon ";" used in conventional programming languages to indicate sequential order, is implied here by the arrangement of event transition statements in sequential order. An explicit event pattern is only needed if more complex enabling/disabling patterns (such as some form of synchronisation between two concurrent processes) need to be represented. The correspondence of the events to the statements of the algorithm is given in Fig 11.

The events are labelled using three letters, the first being either *w* or *r*, indicating if the event is a writer one or a reader one, the second being either *i* or *c*, signifying the actions of *indicating* or *choosing* by the process using control variables, the last being either *p* or *s*, indicating if the choosing or indicating is of a pair or a slot. So *wip* stands for "writer indicates pair", which is done by the statement *w*0 in Fig 9 (The writer indicates to the reader, via the control variable *ip*, which pair of slots it is going to access next.).

Algorithm statements	Role model events
wr	no freshness events
*w*0	*wip*
*w*1	*wcs*, *wis*
*r*0	*rcs*, *ris*
*r*1	*rcp*
rd	no freshness events

Fig 11 Role model events for algorithm statements.

In Fig 10, the absolute positions of the pair and slot agents are flexible, providing for a further shorthand of presentation. If *x*, *y*, *z*, *u*, *v*, and *w* are role patterns, the expression $[\{(x), y, z\}, \{(u), v, w\}]$ represents the situation of one of the pairs assuming the *x* role pattern, with one of its slots assuming the *y* pattern and the other the *z* pattern, and the other pair assuming the *u* pattern, one of its slots assuming the *v* pattern and the other the *w* pattern. It is not, in this case, indicated which of the pairs 0 or 1 assumes the *x* pattern. Therefore such an expression describes all the possible combinations of slots and pairs assuming overall role patterns that satisfies it. For instance, the above expression describes both the state of pair 0 assuming *x*, slot 00 assuming *y*, slot 01 assuming *z*, pair 1 assuming *u*, slot 10 assuming *w* and slot 11 assuming *v* and the state of pair 0 assuming *u*, slot 00 assuming *w*, slot 01 assuming *v*, pair 1 assuming *x*, slot 10 assuming *y* and slot 11 assuming *z*. This type of shorthand presentation of a role model can only be used if the absolute information about each agent's role pattern is not of significance.

The relative positions of the role patterns are significant, however, in that they are consistent within one transition statement. In other words, once the input expression is written the output expression must correspond to it. For instance, the transition statement

$$[\{(*), F*, *\}, \{*\}] \rightarrow [\{(*), f, \overline{f}\}, \{(*), \overline{f}, \overline{f}\}] \qquad \textbf{(9)}$$

which is part of the *rcs* event in Fig 10, describes the following role modifications: The *f* role is added, if previously not assumed, to any slot that assumes the *F* role, but is removed, if previously assumed, from any slot which does not assume the *F* role. The fact that at most one slot may assume the *F* role at this time is indicated by the one *F* in the statement. Note that the flexibility obtained from not having to name agents reduced the number of statements needed to represent this action considerably. The addition operator is omitted from this shorthand.

The roles in this model and their significance are listed in Fig 12.

In [8], the freshness-related roles and their uses are explained as: "A role *F* is allocated at the start of the writer post sequence to the slot just written, but at the same time the old F is transformed to a P to capture the ambiguity which exists while the write post sequence is carrying out its task of making the new data available. P is deleted at the end of this sequence. Both P and F are regarded as acceptable [to be read] for this short interval. The reader has to acquire access to valid data within the duration of its pre-sequence. This is investigated by attaching p, f respectively to the slots carrying P, F at the start of this sequence; all other slots are set to be invalid at that time [$\overline{V}$]. It is possible for new data to become available during the read pre-sequence and this is assigned a V role. To check for freshness, it is merely necessary to establish whether the slot assigned an R at the end of a read pre-sequence also carries a p, f or V."

Roles	Representations
W	slot is being written to
R	slot is being read from
F	slot contains freshest piece of data known to the writer
f	slot contains freshest piece of data known to the reader
P	slot contains previous freshest piece of data known to the writer
p	slot contains previous freshest piece of data known to the reader
V	slot contains freshest data just written but may not have been indicated

Fig 12 Roles and their significance.

With the roles in Fig 12, Definition 14 may be rephrased as follows:

Assuming that the piece of data which is the latest at the beginning of the pre-sequence of the current reader cycle is in slot f, that the piece of data which is the previous latest after the last completed wr is in slot p only if the pre-sequence of the current reader cycle overlaps in time with a writer post sequence, and that any slot containing newer data than slot f is branded V, data freshness is maintained if the reader accesses slot V, slot f, or slot p.

The adding, removing and referencing of the F, f, and V roles are straightforward. The P and p roles are used to verify 2. in Definition 14 in the following way: The event *rcs*, which corresponds to the reader statement $r0$, adds the p role to the slot currently assuming the P role and removes the p role from any other slot. This means that the subsequent *rd* can read from this slot without failing data freshness. On the other hand, from the writer events it is clear that the P role only exists during the writer post sequence. Therefore the p role would only be assumed by any slot if the reader pre-sequence overlaps in time with a writer post sequence.

It is clear that models following the shorthand notation of the type in Fig 10 can always be converted to the basic form of Definition 1 ~ Definition 11. However the basic form will be significantly more complex to present.

It is reported in [8], after analyses using the role model in Fig 10 with the transition diagram method, that the algorithm in Fig 9 maintains data freshness. Further analysis using similar role models concluded that the same algorithm does not maintain correct data sequencing.

P/T net models based on the propositions above have been constructed for both the freshness and sequencing role models which are then coupled to the P/T net model of the mechanism algorithm used for data coherence analysis. Reachability analyses have been carried out using two different pieces of software [15, 16], the results confirm that the algorithm does fail to maintain correct sequencing but does maintain data freshness as defined in Definition 14. A P/T net fragment representing part of (**9**) above, with slot 00 assuming the role *F* (and the other slots not assuming this role), is shown in Fig 13. In order to save space, the no-modification transitions (Fig 4) are not shown in Fig 13 but are present in the models used in the study.

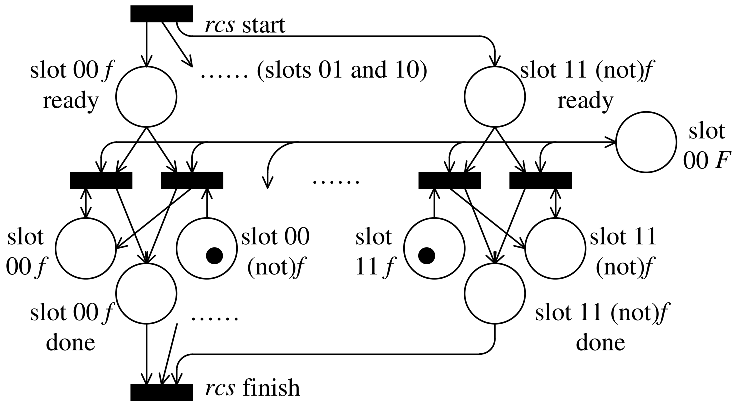

Fig 13 Part of P/T net representation of (9)**.**

There is an element of inelegance in saying that the property of data freshness is still maintained in certain situations even though data sequencing may be incorrect. However, the reason why the algorithm of Fig 9 satisfies Definition 14 but not Definition 15 can be easily found from significant fragments of the reachability graph such as the one shown in Fig 14.

The reachability graph fragment shown in Fig 14 indicates the following development: writing slot 00 (*ip* = 0 and *w*[0] = 0), writer post sequence start, writer post sequence finish, reader pre sequence start, writing slot 11, writer post sequence start, reader pre sequence finish, reading slot 11 (the most up to date data item), reader pre sequence start, writer post sequence finish, writing slot 01, writer post sequence start, reader pre sequence finish, reading slot 00 (Data in 00 is older than that in 11, the last read, hence sequence violation).

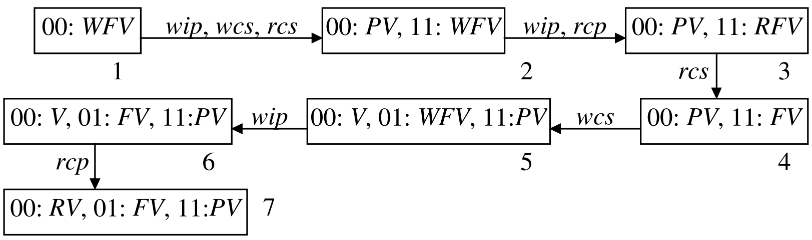

Fig 14 Part of reachability graph showing sequence violation.

From the above, the reason why Definition 14 does allow the potential loss of correct sequencing is quite clear. From marking 2 in Fig 14, the reader may access either slot 00 or slot 11 without violating Definition 14 (satisfying 1. or 3.). In the event it accesses the newer data in slot 11. The next reader pre sequence takes a long time to finish, during which the writer carries out a write and parts of two post sequences. Because of the overlap of the reader pre sequence with the writer post sequences, the reader may access slots 00 (satisfying 2.), 11 (satisfying 1.) or 01 (satisfying 3.) without violating Definition 14. However, if, as in the case of Fig 14, it accesses slot 00 which contains older data than 11, the slot it accessed during its previous cycle, data sequencing as defined in Definition 15 would be violated.

Interestingly, it has been shown, using the same techniques outlined in this paper, that the older four-slot algorithm found in [3] does maintain both data freshness in the sense of Definition 14 and data sequencing in the sense of Definition 15 [17].

Conclusions and future work

Simpson's role models are presented in a formal way for the first time here. It has been shown that role models have significant advantages when there is a need to track state transitions over unknown numbers of steps, for instance, when properties such as data freshness are studied.

A way of representing role models using Petri nets has been presented and a general method for using Petri net techniques to complement the role model method has been tried and proved to be viable. Certain finer points of using the role model method in checking slot mechanisms, especially their data freshness and sequencing properties are explored with a case study. Two software packages developed independently have been used to increase confidence in the analysis results obtained.

Since two different four-slot mechanism algorithms have been shown to behave differently where data sequencing is concerned, further analyses taking advantage of the availability of easy to use Petri net techniques may point out a way to improve data sequencing and related qualities of future slot mechanism algorithm designs.

From related work [14], it seems natural that coloured Petri nets (CPNs) [18] may be used to represent role models in a more straight forward manner, especially considering the shorthand notations of the latter. Therefore this may be a profitable technique to use in future investigations of the role model method. It would also be interesting to see if the role model method may be employed in the study of systems other than slot mechanisms.

This work is part of on-going studies at King's College London and the University of Newcastle into asynchronous communication algorithms and mechanisms. Both authors are supported by EPSRC grants — FX by grants GR/K/70715 (project HADES) and GR/L93775 (project COMFORT at NCL) and IC by grant GR/L92471 (project COMFORT at KCL).

References

1 Joint IECCA and MUF Committee on Mascot (JIMCOM), The official handbook of Mascot, Computing Division, N building, Royal Signals and Radar Establishment, St

Andrews Road, Malvern, Worcestershire, WR14 3PS, UK, June 1987. (Now available from the Defence Research Information Centre, DERA, Kentigern House, 65 Brown Street, Glasgow. G2 8EX.)

2 Simpson, H.R., "The MASCOT method", Software Eng. Journal, Vol. 1, (3), pp.103-120, 1986.

3 Simpson, H.R., "Four-slot fully asynchronous communication mechanism", IEE Proceedings, Vol. 137, Pt. E, No. 1, pp.17-30, January 1990.

4 Kirousis, L.M., "Atomic multireader register", Proc. 2nd Int. Workshop on Distributed Computing, Amsterdam, LNCS-312, pp.278-296, Springer Verlag, 1987.

5 Tromp, J., "How to construct an atomic variable, Proc. 3rd Int. Workshop on Distributed Algorithms, Nice, LNCS, Springer Verlag, pp.292-302, 1989.

6 Simpson, H.R., "Fully-asynchronous communication", IEE Colloquium on MASCOT in real-time systems, pp.2/1-2/6, May 12, 1987.

7 Simpson, H.R., "Correctness analysis of class of asynchronous communication mechanisms", IEE Proceedings, Vol. 139, Pt. E, No. 1, pp.35-49, January 1992.

8 Simpson, H.R., "New algorithms for asynchronous communication" and "Role model analysis of an asynchronous communication mechanism", IEE Proceedings on Computing and Digital Techniques, Vol. 144, No. 4, pp.227-240, July 1997.

9 Peterson, J.L., Petri net theory and the modeling of systems, Prentice-Hall, 1981.

10 Xia, F, Clark, I.G. and Davies, A.C., "Petri-net based investigation of synchronisation-free interprocess communication in shared-memory real-time systems", Proc. 2nd UK Asynchronous Forum, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, July 1-2, 1997.

11 Semenov, A. and Yakovlev, A., "Contextual net unfolding and asynchronous system verification", Technical report, No.572, Department of Computing Science, University of Newcastle, April, 1997.

12 Clark, I.G., PhD thesis, King's College, University of London, in preparation.

13 Xia, F., Combining MASCOT with Petri nets, PhD thesis, King's College, University of London, to be published in 1998.

14 Clark, I.G., Xia, F., Yakovlev, A.V. and Davies, A.C., "Petri net models of latch metastability", Electronics Letters, pp.635-636, Vol.34, No.7, April 2, 1998.

15 Clark, I.G., "Reach — a reachability search program for cyclic P/T nets", the Petri net email list, URL http://www.daimi.aau.dk:80/~petrinet/, 1995.

16 Zuberek, W.M., "Modeling using timed Petri nets -- model description and representation", Technical Report No.9601, Department of Computer Science, Memorial University, St. John's, NF, Canada A1C 5S7, August 1996.

17 Clark, I.G., Davies, A.C., "Data freshness and data sequencing in asynchronous communication: an enhancement to Simpson's role model analysis", submitted for publication in IEE Proceedings on Computing and Digital Techniques, 1998.

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