ASSESSING THE GRAPHICAL AND ALGORITHMIC STRUCTURE OF HIERARCHICAL COLOURED PETRI NET MODELS

Dr George L. Benwell and Dr Stephen G. MacDonell
Department of Information Science
University of Otago
Dunedin, New Zealand

ABSTRACT

Petri nets, as a modelling formalism, are utilised for the analysis of processes, whether for explicit understanding, database design or business process re-engineering. The formalism, however, can be represented on a virtual continuum from highly graphical to largely algorithmic. The use and understanding of the formalism will, in part, therefore depend on the resultant complexity and power of the representation and, on the graphical or algorithmic preference of the user. This paper develops a metric which will indicate the graphical or algorithmic tendency of hierarchical coloured Petri nets.

INTRODUCTION TO PETRI NETS

Since the 1960s when Petri nets were first devised (Petri, 1962) they have been modified and advanced. As a result their power and useability have increased. The application domain has also increased to reflect these improvements. Originally a Petri net was defined as a bipartite graph consisting of places, transitions, arcs and tokens, such that;

$P = \{p_1, p_2, ... p_n\}$ is the set of n places (graphically represented as circles)
$T = \{t_1, t_2, ... t_m\}$ is the set of m transitions (graphically represented by rectangles)
A = {the set of directed arcs connecting places and transitions}, and
M = {the set of tokens resident in places at a given instant} (tokens are represented by dots).

In addition to these constructs a net has associated sets of enabling and firing rules. These rules control the actions relating to transitions. The enabling rules determine under what conditions (or particular marking) a transition is enabled and therefore may fire. The firing rules determine what action will occur as a transition is fired. Originally, these rules were simplistic, but nonetheless appropriate. They were; a transition, T_i, is enabled iff all its input places have a token and none of its output places have a token (Symons 1982), that is;

T_i is enabled when $P_I = 1$ AND $P_O = 0$
where
P_I is the set of input places for T_i, and
P_O is the set of output places for T_i.

The result of this early theory was that the nets were complex even when modelling relatively simple processes. This was considered to be a disadvantage, and in part, was responsible for a reluctance to use the nets. There was little point in producing a model that was potentially more complex than the phenomenon being modelled. It is important to emphasise here that complexity refers to structure of the graphical formalism. As the rules were simplistic a process had to be decomposed almost to an atomic level. Simple rules led to the creation of complex graphics. Again, complexity needs to be clarified. The formalism has only four constructs so complexity arises not from that number, but rather from their excessive and repetitive use.
These deficiencies were addressed with the introduction of coloured Petri nets and also hierarchical coloured Petri nets (Reisig 1991). The extension of the theory allowed for coloured tokens and complex algorithmic rules (op. cit.) to control the flow structure of the net. The rules supplemented and extended the notion of enabling and firing rules. Rules could now be far more complex. The result was that nets were no longer rampantly complex in terms of graphics. It is now possible to represent a complex process with minimal graphics, where the graphical structure is transformed into coloured tokens and complex rules. So Petri nets can be now described in the following manner (Purvis and Purvis, 1993);

- a net structure, which is just like that of ordinary Petri nets,

- a set of data declarations, and
- a set of net inscriptions.

The net structure is like that of Place/Transition nets: nodes consisting of places and transitions, and directed arcs that always connect nodes of different types (either places or transitions).
The declarations component, defines the colours, number and types of variables in the net. Each place, instead of having a capacity attached to it, now will have a colour set attached to it. Colour sets are analogous to abstract data types in programming languages, and just as with abstract data types, each colour set can have an associated set of operations and functions that can be applied to tokens of that colour. Because of this analogy with abstract data types, the declarations component could be expressed in terms of any of a number of programming notations. For example, Jensen's (Jensen, 1990) original notation of Standard ML could be used as it is in a commercial implementation of coloured Petri nets (MetaSoft Corp., 1992).
The net inscriptions, are expressions which can be attached to a place, a transition, or an arc;

- Places can have colour sets and initialisation expressions (analogous to the capacities and initial markings of Place/Transition nets).
- Transitions can have guards, which are boolean expressions that may contain operations on objects, such as constants, variables, and functions that have been defined in the declarations component. Guards must evaluate to TRUE in order for the transition to fire. If a guard always evaluates to TRUE, then it is not shown.
- Arcs may contain arc expressions, which also contain constants and items that have been defined in the declarations component. When the arc expressions are evaluated, their variables are bound to the appropriate colours. The value of the expression must be equal to a multi-set (bag) of the colour that is attached to the place at one end of the arc. The arcs associated with a single transition (whether incoming or outgoing) have a common scope: any variable that appears more than once must be bound to the same colour.

The end result is that there is now a continuum of combinations. A net could be highly graphical with little more than simple rules or at the other end of the spectrum, it could be hardly graphical and highly algorithmic. Figure 1 presents this concept.
It is contended that it is appropriate to derive a measurement to assess the relative percentages of graphics and algorithmic rules of a net. In the short term, this metric could be simply to determine if there are classes of nets. In the long term, it could be possible to match 'net type' with 'end user type' to maximise the outcome of the application of the Petri net formalism. This paper uses the term 'structure' as a measure of the relative percentages shown in Figure 1. It may be argued that it would have been better to use 'complexity'. This latter term is related here more to the number of constructs in use rather than their alternate representations (graphical or algorithmic).
This definition is supported by Symons (1982) where he states;

> 'The structure of a net is defined by the interconnection pattern of the places and transitions' Symons (1982, p 3)

To be absolutely sure then, this paper deals with measuring the two forms of representation - graphical and algorithmic and at this moment is not concerned with comparing two different nets for complexity (number constructs). The latter would be important when measurements of efficiency and clarity are required.

PETRI NETS AND INFORMATION SYSTEMS

Petri nets have been utilised in information science since they were devised in the 1960s. It is true to say, however, that the use has not been wide spread. It is claimed that this is due to their complexity, though this objection is surely diminishing with the advent of coloured and hiearachical nets. In fact Petri nets have been seen as a rigorous formalism and as such an appropriate substitute for data flow diagrams (Tse et al, 1989, p1). In that vision the complexity was not denied and it should not be regarded as totally negative nor insurmountable. A synergy is required between the 'simplicity and informality' of data flow diagrams and the 'complexity and rigour' of Petri nets. Benwell et al. (1991) have proposed an extended use of Petri nets in the system development life cycle.

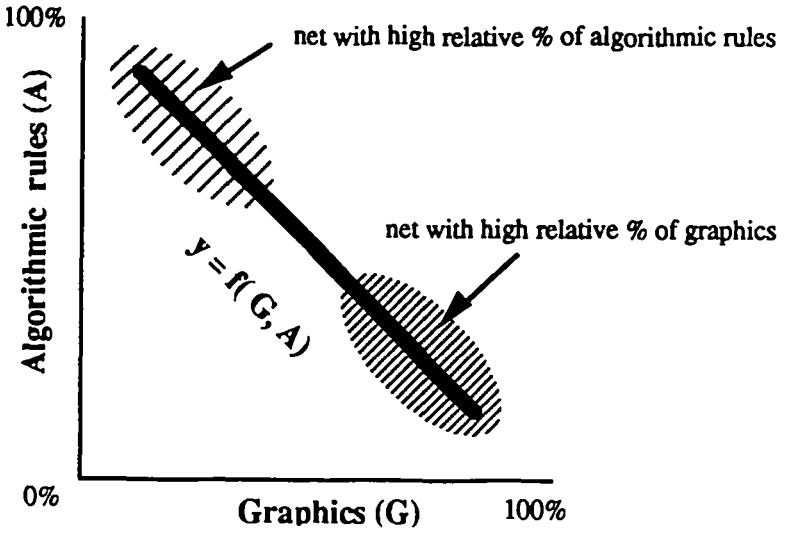


Figure 1

When an information system is to be created it is necessary to understand the phenomena to be modelled. Any representation of the reality depends upon some method for categorising and defining what is being observed in a given situation. For example, for any situation being observed it may be desirable to categorise items as entity types or processes. During the observation, all phenomena and their most obvious situation-related processes are then classified. For instance, in a VEHICLE RENTAL situation (such as Figure 5a), a CUSTOMER and a VEHICLE would be classified as entity types. This method, with some variations, has been described as an object-oriented approach to systems analysis (Bailin, 1989). In entity relationship models entity types may be people, objects or events. In structured process modelling the emphasis is more on processes which relate to the dynamics of information in reality. Using this method, the reality of the situation being observed can be described in terms of its data and processes. The next step is to produce an entity relationship model and data flow diagrams. So a variance on existing sytsem delevolpment methods is one that is seen to model dynamic processes rather than static models of data.

INTRODUCTION TO METRICS

To assess the structure of a Petri net model it may be appropriate substantially to adopt techniques from the field of software metrics, particularly those investigations that have attempted to assess and analyse graphical product representations. Software metrics is the common name for the branch of software engineering that is concerned with the (normally quantitative) assessment of software project effort, quality assurance and the like. Underlying all of these major goals is the fundamental attribute measurement of products, resources and processes (Fenton 1991). As a Petri net is a tangible system representation, this paper is most interested in methods of product assessment. Some previously proposed product measures are very simple, for example, the number of lines of code in a program, or the number of modules on a structure chart. Others, such as Basili and Hutchens' SynC family of measures (1983) and the Macro/Micro measures proposed by Harrison and Cook (1987), are more complicated combinations of several product attributes.

Software product measures such as these have been used in a wide variety of functions. Some have been used, for example, in the estimation of software development costs, or to assist in the prediction of maintenance requirements, or in the evaluation of particular development strategies (MacDonell 1992). In some cases, however, the validity of these investigations has been subsequently challenged, as the measures used have been chosen inappropriately and have therefore failed to actually assess the characteristic(s) of interest (Fenton 1991). This has occurred most especially when assessments of poorly-defined attributes such as 'understandability' or 'useability' have been the goal. The concern in this paper, however, is only with the direct assessment of Petri net structure. It is therefore likely that problems of validity can be largely avoided through the use of relatively rigorous measurement definitions. Once this assessment has been shown to be valid, it may then be possible to determine useful relationships between the structure metrics and other aspects of the software process. In addition, discussion here is not concerned with functionality as this should appear in both (or any) representations

of the same system. The core theme is to only consider the structure of the chosen representation in terms of path/decision metrics. Figures 2a and b provide a simple example. While being simple it must be said that the example also displays some relaxed theoretical rules. That is, the sink or counter at P4 in (a) and P3 in (b), once marked by a token would bar the transition T1 from being enabled again. This relaxed rule matters little in terms of the measurement of structure being discussed here. Figures 2a and 2b are equivalent. The functionality represented graphically in Figure 2a is distributed between graphics and explicit algorithmic rules in Figure 2b.

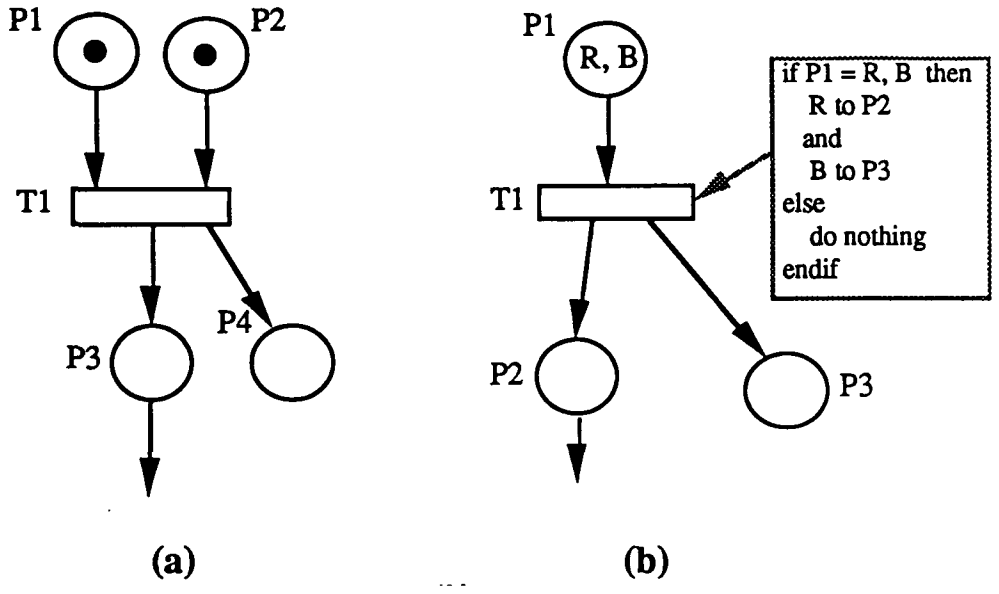


Figure 2

As described earlier, a Petri net is a system representation formalism that incorporates both graphical and algorithmic components. Structural assessment of the whole would intuitively require separate assessment of each aspect, as both contribute to overall net size and structure. Given that the algorithmic component is concerned with processing control (Jensen, 1990), which can also be depicted graphically, it would seem appropriate to use graphical analysis methods for both Petri net components.

Metrics that are derivable from graphic product representations are widespread in the software engineering literature; for example, the methods described by Chen (1978), Benyon-Tinker (1979) and Szulewski et al. (1981). Some are based on conceptual or functional system descriptions, such as DeMarco's Bang method (1982). Others are derived from design documents, and are therefore generally based on a product's calling structure. Of most interest here, however, are those assessment techniques that may be applied to lower level product abstractions as represented in graph form, as these will be directly applicable to the graphical structure of Petri nets. Although many of these methods were initially proposed as measures of software product complexity (for example, see McCabe (1976), Woodward et al. (1979) and Chen (1978)), the underlying rationale of the methods is appropriate for the less ambitious assessment of Petri net structure.

MEASUREMENT OF PETRI NET STRUCTURE

A graphical structure assessment method adapted directly from software metrics (but which, interestingly was originally derived from graph theory) is that which determines the 'cyclomatic complexity' of a program flowgraph (McCabe 1976). As a Petri net can be considered in terms of a directed graph C (Peterson 1977) with nodes p and t (places and transitions), edges a (arcs), and connected components c, the formulation of the graph cyclomatic number v is as follows (Berge 1973):

$$v(C) = a - p - t + c \qquad (1)$$

Equation (1) applies when the Petri net graph is strongly connected (for strong connections see, Symons, 1982, p7); that is, a directed path may be traced between any pair of nodes in the graph.

Some graphs, however, are not strongly connected. For example, a given model may have logical starting or ending points, or both. This can be observed in the form of 'start' nodes s or 'finish' nodes f, places that only have inputs or outputs respectively. For Petri net graphs of this type the above formulation must be adjusted accordingly (Henderson-Sellers 1992). This is important as, at least intuitively, they are two basic net forms. The first may be called cyclic, having no end or beginning, and is a net that models the ongoing logic of a system. The other may be called episodic, which clearly have start and finish nodes. The fact that the difference may only be the level of abstraction or the 'view' of the system or process does not remove the need for the classification. Thus for nets that include a start place and/or a finish place the equation for the cyclomatic number becomes:

$$v(C) = (a + 1) - p - t + c \qquad (2)$$

Finally for nets that have two or more start places or two or more finish places the calculation of the cyclomatic number is of the form:

$$v(C) = ((a + 1) + (s + f)) - (p + t + 2) + c \qquad (3)$$

Figure 3 is an example of how equations (1) to (3) may be applied to small nets.

Values of v (as calculated from equations (1) and (2)) were used by McCabe (1976) as measures of graph 'complexity', and subsequently as a basis for determining software testing strategies. This approach was based on the underlying assumption that a greater number of decision constructs meant a greater number of paths, and that this in turn contributed significantly to higher product complexity (also see (Peterson, 1981, p118)). Since it was first proposed, McCabe's method has been the subject of extensive investigation, with somewhat mixed results. Some empirical support has appeared - for example, see the studies reported by Curtis et al. (1979) and Hartman (1982) - and, on the whole, the measure is viewed as one of the more effective software metrics (Boehm 1981; Arthur 1985; Li and Cheung 1987). In contrast, however, it has also been the subject of some criticism in relation to its treatment of path control constructs. This has led in some cases to metric refinements being proposed based on notions of intuitive complexity (Myers 1977; Hansen 1978; Ramamurthy and Melton 1988). In the current study, however, the issue of varying construct complexity is of minimal concern, as the interest lies only in the underlying path structure of a Petri net (which is unaltered by construct variations), not how this relates to complexity or understanding (Shepperd 1988). Thus the measures are used here simply to provide standardised indicators of the graphical and algorithmic path structure of Petri nets.
The c term in equations (1) to (3) relates to the number of connected components. McCabe (1976) adopted this term as an adjustment factor for the use of sub-programs, so that the number of components equalled the main module plus all called modules. Thus when determining the value of v for a single module the value of c was equal to 1. In the current context it seems natural to take the same approach - the value of c for each distinct self-contained Petri net will therefore also be set equal to 1. In addition there may be some need to consider the hierarchical nature of coloured Petri nets as a contributing factor to the structure of a net. This is believed to have more to do with functionality than structure (as already defined). So those places and transitions which link layers in the hierarchy will not be given any higher weight than they do as ordinary nodes. It has also been shown that transitions and places can be transposed into places and transitions to produce an equivalent net (Peterson, 1981, p13). It would therefore seem logical that places and transitions are all nodes of equal significance. Given these conditions the three equations above may be reduced (Stetter, 1984; Henderson-Sellers, 1992):
Equation 1 becomes,

$$v(C) = a - p - t + 1 \qquad (4)$$

Equation 2 becomes,

$$v(C) = a - p - t + 2 \qquad (5)$$

Equation 3 becomes,

$$v(C) = a - p - t + s + f \qquad (6)$$

Algorithmic structure assessment is likely to follow a very similar approach, as the coded rules in a Petri net perform essentially the same function as the graphical form; that is, they control the sequence of functions through the use of conditional tests (Purvis and Benwell, 1993). Thus the graphical assessment methods

discussed above will also be adapted for use with the algorithmic net component. This clearly requires the development and measurement of a flowgraph for each place, transition or arc in which rules are used. The total structure measure for the algorithmic component would be the sum of the individual block measures adjusted for the number of blocks (Henderson-Sellers, 1992).

The Petri net shown in Figure 2b will serve as an initial example. Figure 4 shows the concept of forming a graphic equivalent of the algorithmic component for assessment purposes. The net originally shown in Figure 2(b) is therefore now represented as two graphs, with the numbers on each indicating the counts of places, arcs and transitions. The derivation of the algorithm-based graph (on the right of Figure 4) is intuitive - place 1 (P1) represents the IF statement, arc 1 is equivalent to the THEN branch, arc 2 the ELSE branch, the transitions represent the two "actions", and arcs 3 and 4 represent the implicit return of control to the next sequential statement, the ENDIF in P2.

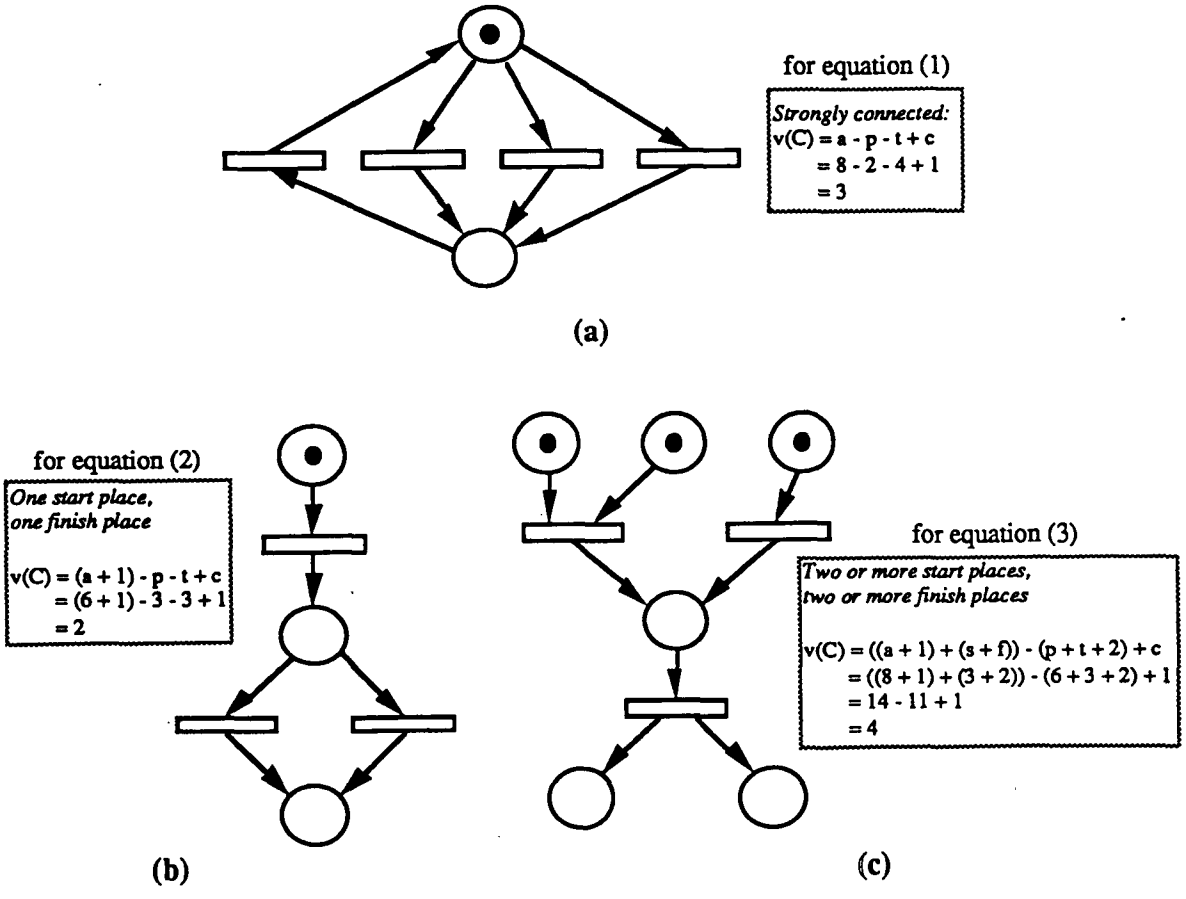


Figure 3

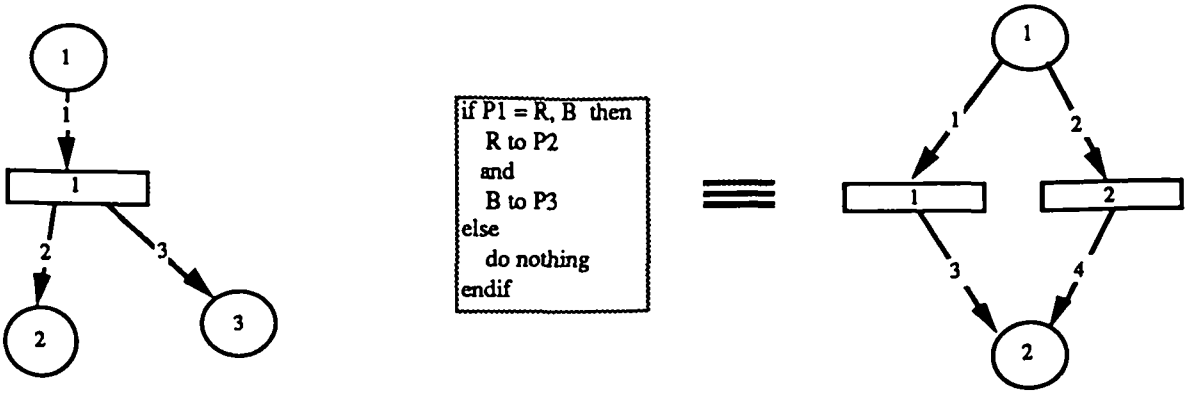


Figure 4

Using equations (5) and (6) the metrics for graphical structure (S_g) and algorithmic structure (S_a) for Figure 4 can be expressed as;

$$S_g = a - p - t + s + f \qquad (7)$$

$$S_a = a - p - t + 2 \qquad (8)$$

So, from Figure 4 it is determined that;

$$S_g = (3 - 3 - 1 + 1 + 2) = 2$$

and

$$S_a = (4 - 2 - 2 + 2) = 2$$

It may be held that it is necessary to determine the number of tokens in the net as a contributing factor toward structure. While this may be so, tokens are not considered separately as;

1. the number of types of tokens is the significant factor not the number of tokens,
2. the types of tokens are considered indirectly in the assessment of the algorithmics,
3. the number of tokens relates more to net complexity than net structure. Safeness and conservational aspects of a net, directly concern tokens but relate to complexity and are not assessed here.

As an example consider a Petri net modelling a car hire process (Benwell, Firns and Sallis, 1991) (see Figure 5). In a part of the net, cars are represented by tokens; the number of cars will dictate the number of coloured tokens (but not necessarily the number of colours). This will not inherently change the structure of the net nor in fact complexity until such time as the processes to handle the number of cars change. Then a new process is being modelled so it should be expected that structure (and maybe complexity too) will be different. Therefore a net to model 10 cars or a 100 cars will have the same structure, provided all cars are treated equally and the process to handle the number of cars is the same.

The measure of overall Petri net structure S_p, as given by equation (9), could be solely used as a tool for net evaluation. Notwithstanding that possibility, the thesis of this paper is to derive a metric that will classify nets as predominantly graphical or algorithmic and then, and only then, to draw some conclusions as to any differences between the two types.

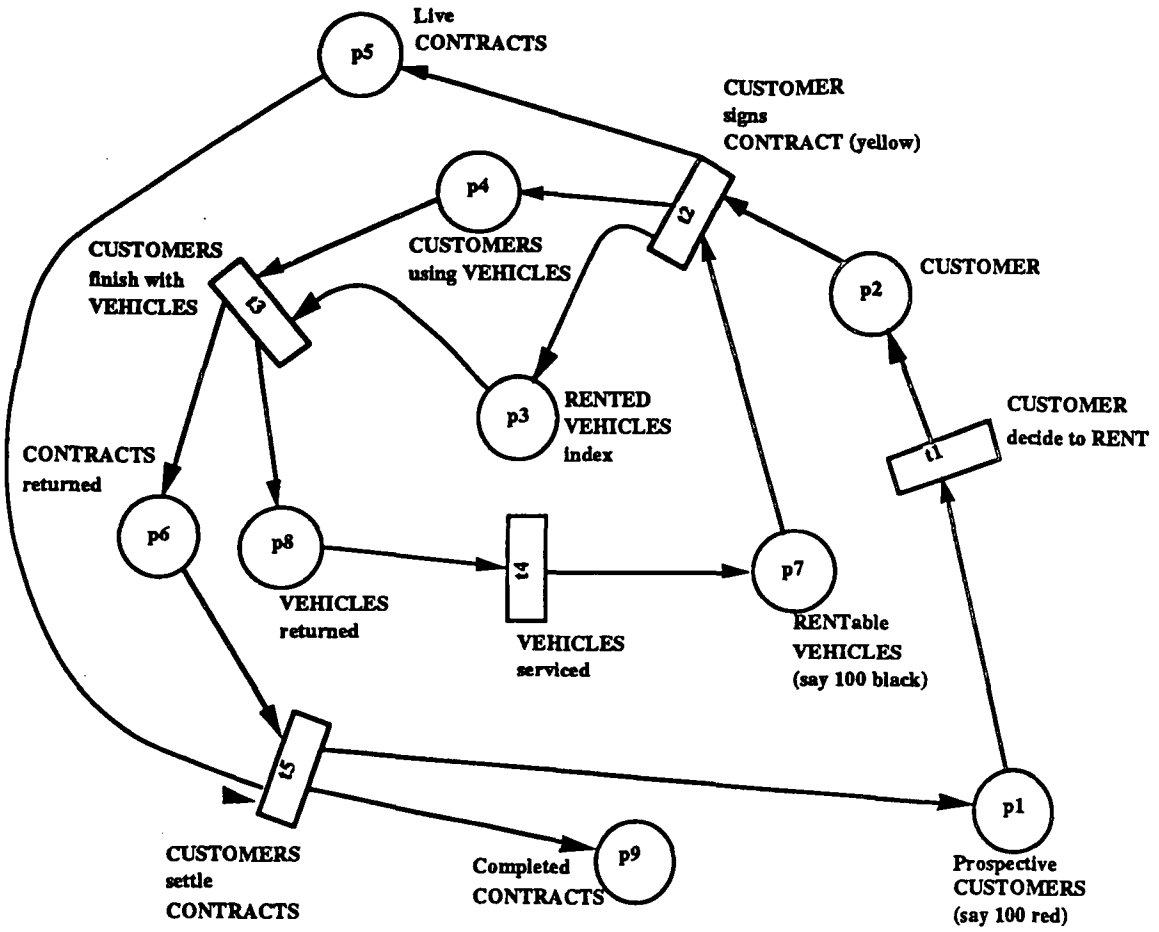
Structural dominance can be expressed in terms of graphics or algorithmics according to the following;

$$D_{pg} = S_g/S_p \qquad (10)$$

$$D_{pa} = S_a/S_p \qquad (11)$$

but, in cases in which S_g and S_a contribute to the structure:

$$D_{pg} + D_{pa} \_ 1 \qquad (12)$$



Car Hire Net (adapted from (Benwell et. al. 1991))

Figure 5

So, it is not possible to directly compare the influences and sum them to 100%. It is desirable to have the sum of equation (12) equal to one. This can be achieved by adjusting S_p. Therefore,

$$S_{p'} = S_p + (c - 1) \quad (13)$$

and

$$D_{pg} = S_g/S_{p'} \quad (14)$$

$$D_{pa} = S_a/S_{p'} \quad (15)$$

and, therefore,

$$D_{pg} + D_{pa} = 1 \quad (16)$$

Now the two components may be compared directly as complementary percentages (equation (16)). The outcome is a direct comparison, but $S_{p'}$ no longer represents the total count of all paths through the net.

Equations (14) and (15) enable the assessment of the graphical or algorithmic dominance in a given Petri net. Some 'classes' of Petri net user may prefer graphically or algorithmically dominated representations. The consequential matching of a graphical (/algorithmic) net with a graphical (/algorithmic) user has not escaped attention; though it is not the subject of this paper. For example, project managers as a group, may prefer a higher graphic component, as this may enable more effective communication. Conversely, programmers might be better served by having algorithmic dominated nets, so that the implementation of the functions described by the net are made more straightforward. In addition, there may be classes of problems that are better portrayed in either representation. At this stage, however, the paper is not concerned with why different preferences exist.

Given the ratio form of the Petri net dominance indicators, the value of $D_{p(i)}$ will fall somewhere along the continuum from graphically dominated to algorithmic dominated. When D_{pg} equals D_{pa} the structure is evenly distributed between the graphical and algorithmic components. A different rendition of Figure 1 conveys the concept in Figure 6.

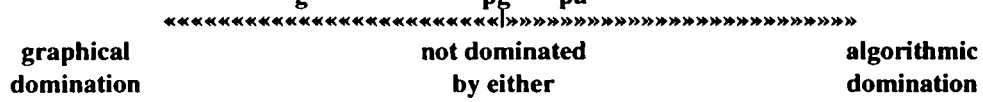


Figure 6

THE DINING PHILOSOPHERS PROBLEM

The dining philosophers problems is now universally established (Peterson, 1981, p65 and Krishnamurthy, 1989, p54) and is a good practical example for comparing the structural dominance of the graphics and algorithmics. The Petri net for the philosophers (adopted from Peterson (op. cit.)) in given in Figure 7. The problem is that there are five diners and five eating utensils, the latter can be assumed to be forks (a, b, c, d, e in Figure 7). A diner needs two forks to eat. If two forks are not available the diner must be a true philosopher and forego eating for thinking. It is only possible for two opposite diners (say '1' and '3' or '1' and '4' for example) to eat at the same time. Figure 7 represents this situation where diner '1' is eating, the immediate left ('5') and right ('2') dinner partners must think, while either of the upper two ('3' or '4') may choose to eat. If one chooses to eat, the other cannot. This situation is cyclic and continues *ad nauseam*. The net configuration, as presented, makes a deadlock impossible. In the more general case it would be possible where any diner can select *a* fork (as distinct from *a pair*) at a given time.
The graphical measure of this net is given by;

$$S_g = a - p - t + c = 40 - 15 - 10 + 1 = 16$$

and $S_a = 0$, so

$$S_p = (S_a + S_g) - (c - 1) = 0 + 16 - (1 - 1) = 16$$

$$S_{p'} = S_p + (c - 1) = 16 + (1 - 1) = 16$$

Therefore,

$$D_{pg} = S_g/S_{p'} = 16/16 = 1 \gg 100\%$$

$$D_{pa} = S_a/S_{p'} = 0 \gg 0\%$$

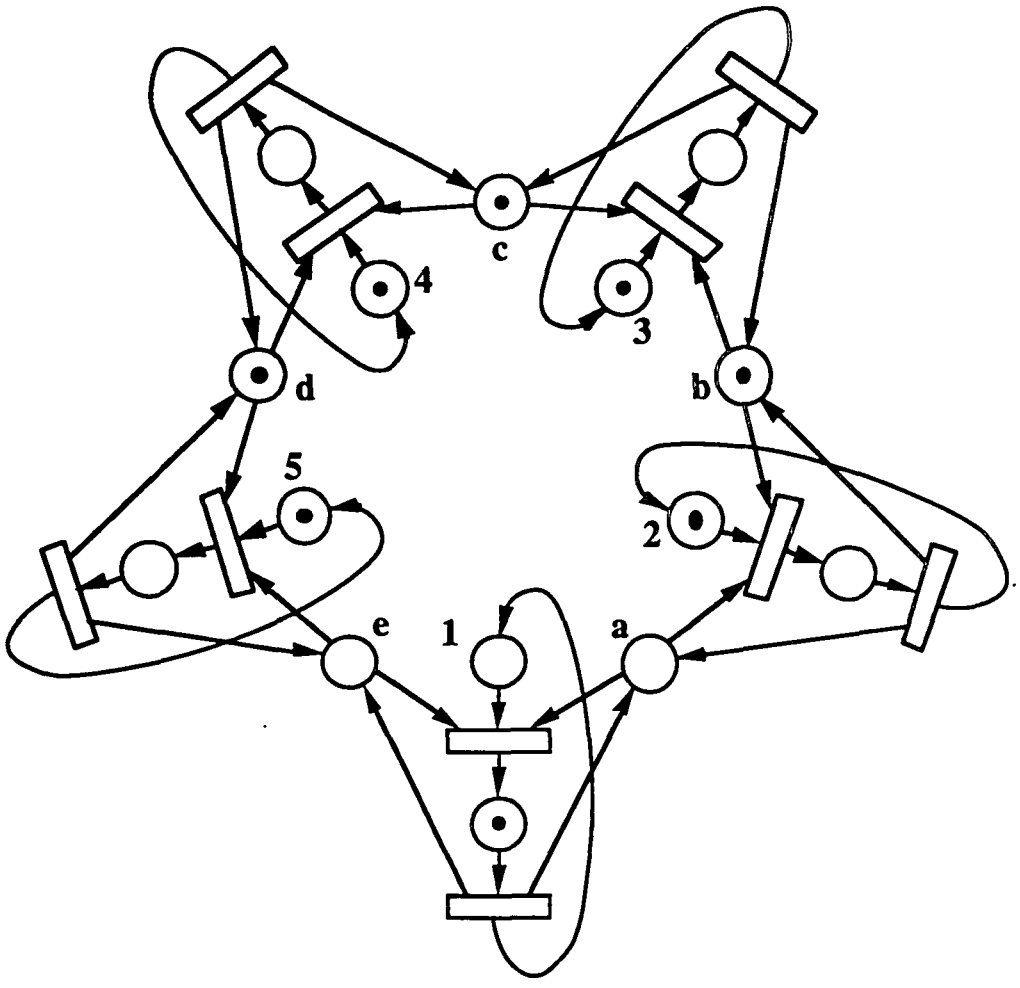
which leads to the obvious conclusion that the net is graphically dominated. Now a new net can be created where some or all of the graphical logic is transposed to algorithmics. This is presented in Figure 8. Calculation of the graphical structure of the Figure 8 representation is relatively straight forward;

$$\begin{aligned} S_g &= a - p - t + c \\ &= 4 - 2 - 2 + 1 \\ &= 1 \end{aligned}$$

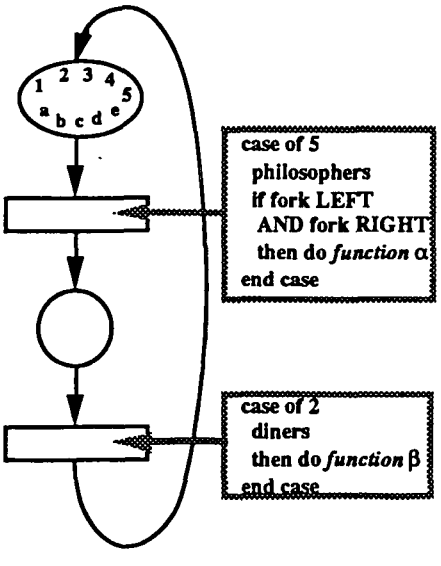
The assessment of the algorithmic structure, S_a, however, requires some preliminary calculations. Given that there are two blocks of algorithmic code in the representation, the value for each block must be determined first;

$$\begin{aligned} S_{a1} &= (a + 1) - p + c \\ &= (30 + 1) - 17 + 1 \\ &= 15 \end{aligned}$$

$$\begin{aligned} S_{a2} &= (a + 1) - p + c \\ &= (4 + 1) - 4 + 1 \\ &= 2 \end{aligned}$$



The Dining Philosophers
(adapted from, Peterson, 1981, p65)
Figure 7



Collapsed Dining Philosophers' Net
Figure 8

As stated previously, the total algorithmic structure value is the sum of the individual block measures adjusted for the number of blocks. Hence;

$$\begin{aligned} S_a &= (S_{a1} + S_{a2}) - (c - 1) \\ &= (15 + 2) - (2 - 1) \\ &= 16 \end{aligned}$$

Now that values for S_g and S_a have been obtained the remaining indicators of overall structure, S_p, $S_{p'}$, D_{pg} and D_{pa} can be evaluated;

$$\begin{aligned} S_p &= (S_g + S_a) - (c - 1) \\ &= (1 + 16) - (2 - 1) \\ &= 16 \end{aligned}$$

(as can be expected, the overall structure value for this representation is equal to that of the representation in Figure 7).

$$\begin{aligned} S_{p'} &= S_p + (c - 1) \\ &= 16 + (2 - 1) \\ &= 17 \end{aligned}$$

Therefore,

$$D_{pg} = S_g/S_{p'} = 1/17 = 0.06 \text{ » } 6\%$$

$$D_{pa} = S_a/S_{p'} = 16/17 = 0.94 \text{ » } 94\%$$

These values now show that the net, as represented in Figure 8, is dominated heavily by algorithmic structure. An intuitive assessment would no doubt have generated the same conclusion; the calculations above, however, provide quantitative relative indicators of this fact.

CONCLUSION

This paper has adopted and derived a numerical technique to measure the relative percentages of graphical and algorithmic structure of a Petri Net. The basis for the calculation has been the research and development undertaken in the area of software metrics. The metric so derived can be utilised to indicate the influence each component has on the structure of a Petri Net. This has fulfilled the initial goal of this research. In the longer term it may be possible to match the attributes of a net to the characteristics of a user and thereby enhance the functionality of Petri Nets as a formalism.

REFERENCES

Arthur, L.J. (1985) **Measuring Programmer Productivity And Software Quality**, John Wiley & Sons, New York.

Bailin, S.C. (1989) "An Object-Orientated Requirements Specification Method", **Communications of the ACM**, 32, 5.

Basili, V.R. and Hutchens, D.H. (1983) "An Empirical Study of a Syntactic Complexity Family", **IEEE Transactions on Software Engineering** 9 (6), pp 664-672.

Benwell, G.L., Firns, P.G. and Sallis, P.J. (1991) "Deriving Semantic Data Models from Structured Process Descriptions of Reality", **Journal of Information Technology**, No. 6, pp 15-25.

Benyon-Tinker, G. (1979) **Complexity Measures In An Evolving Large System, Conf. Proc., Workshop on Quantitative Software Models for Reliability, Complexity and Cost**, New York NY, USA (1979, pp 117-127.

Berge, C. (1973) **Graphs and Hypergraphs**, North-Holland, Amsterdam.

Boehm, B.W. (1981) **Software Engineering Economics**, Prentice-Hall, Englewood Cliffs NJ.

Chen, E.T. (1978) "Program Complexity and Programmer Productivity", **IEEE Transactions on Software Engineering** 4 (3), pp 187-194.

Curtis, B., Sheppard, S.B., Milliman, P., Borst, M.A. and Love, T. (1979) "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics", **IEEE Transactions on Software Engineering** 5 (2), pp 96-104.

DeMarco, T. (1982) "Controlling Software Projects", Yourdon, New York.

Fenton, N.E. (1991) **Software Metrics**, Chapman & Hall, London.

Hansen, W.J. (1978) "Measurement of Program Complexity by the Pair (Cyclomatic Number, Operator Count)", **ACM SIGPLan Notices** 13 (3), pp 29-33.

Harrison, W.A. and Cook, C. (1987) "A Micro/Macro Measure of Software Complexity", **Journal of Systems and Software** 7, pp 213-219.

Hartman, S.D. (1982) "A Counting Tool for RPG", **ACM SIGMetrics PER** 11 (3), Fall, pp 86-100.

Henderson-Sellers, B.H. (1992) "Modularization and McCabe's Cyclomatic Complexity", **Communications of the ACM** 35 (12), pp 17-19.

Jensen (1990,) **Coloured Petri Nets: High Level Language for System Design and Analysis**, in Advances in Petri Nets 1990, Springer-Verlag, Berlin.

Krishnamurthy, E.V. (1989) **Parallel Processing, Principles and Practice**, Addison-Wesley, Sydney, 332pp

Li, H.F. and Cheung, W.K. (1987) "An Empirical Study of Software Metrics", **IEEE Transactions on Software Engineering** 13 (6), pp 697-708.

MacDonell, S G. (1992) **Quantitative Functional Complexity Analysis of Commercial Software Systems**, unpublished PhD thesis, University of Cambridge.

McCabe, T.J. (1976) "A Complexity Measure", **IEEE Transactions on Software Engineering** 2 (4), pp 308-320.

MetaSoft Corp. (1992) **MetaSoft Corporation Manual, Design/CPN Version 9**, MetaSoft Corporation, Cambridge, Mass.

Myers, G.J. (1977) "An Extension to the Cyclomatic Measure of Program Complexity", **ACM SIGPLan Notices** 12 (10), pp 61-64.

Peterson, J.L. (1977) "Petri Nets", **Computing Surveys**, Vol. 9, No. 3, pp 223-252.

Peterson, J.L. (1981) **Petri Net Theory and The Modelling of Systems**, Prentice-Hall, Eaglewood Cliffs, New Jersey, 290 pp

Petri, C. A. (1962) **Communication with Automata**, Supplement 1 to RADc-TR-65-377 Vol. 1, Griffiss Air Force Base, New York, - Originally published in German 'Kommunikation mit Automaten', University of Bonn, cited in (Peterson, 1981, p 3).

Purvis, M.K. and Benwell, G.L. (1993) "A Causal Agent Approach for Modelling Dynamic Systems", New Zealand Computer Society, Conference Proc., **35th NZCS Conference**, Hamilton, New Zealand, pp 592-604.

Purvis M.A. and Purvis, M.K. (1993) "Dynamic Modelling of the Resource Management Act", Conf. Proc., **5th Annual Colloquium, Spatial Information Research Centre**, University of Otago, New Zealand, pp 225-240.

Ramamurthy, B. and Melton, A. (1988) "A Synthesis of Software Science Measures and the Cyclomatic Number", **IEEE Transactions on Software Engineering** 14 (8), pp 1116-1121.

Reisig, W. (1991) **A Primer in Petri Net Design**, Springer-Verlag, Berlin.

Shepperd, M. (1988) "A Critique of Cyclomatic Complexity as a Software Metric", **Software Engineering Journal**, pp 30-36.

Stetter, F. (1984) "A Measure of Program Complexity", **Computer Language (UK)** 9 (3/4), pp203-208.

Symons, F.J.W. (1982) "The Application of Petri Nets and Numerical Petri Nets", **Research Laboratories Report 7520, Telecom**, Australia.

Szulewski, P.A., Whitworth, M.H., Buchan, P., and DeWolf, J.B. (1981) "The Measurement of Software Science Parameters in Software Designs", **ACM SIGMetrics** *PER* 10 (1), pp89-94.

Woodward, M.R., Hennell, M.A., and Hedley, D. (1979) "A Measure of Control Flow Complexity in Program Text", **IEEE Transactions on Software Engineering** 5 (1), pp45-50.