Knowledge Based Support for System Verification during Requirement Analysis

Mohan R. Tanniru
School of Management, Syracuse University, Syracuse NY 13210

and

S. Sakthivel
College of Business Administration, Bowling Green State University
Bowling Green Ohio 43403

Abstract

Requirement analysis is the first step in the development of information systems and its objective is to ensure that any proposed system meets the projected requirements of the user. As a part of this requirement analysis, an analyst is asked to verify that the proposed system is representationally accurate before undertaking any other evaluation. This paper proposes a Petri Net representation of an information system and the use of a PROLOG-based knowledge base to test its representational accuracy.

Introduction

Organizations rely on accurate and timely information for making decisions and managing their operations. The availability of correct information and, in turn, the systems that provide such information, are critical to supporting the basic purposes and goals of an organization. The development of such correct information systems is dependent of how well a user's needs are specified (requirement specification) and an information system designed to meet these needs (system design and implementation). The importance of accurate specification of user requirements is obvious for a successful development of correct information systems. A system developed from wrong specifications will lead to inappropriate design and, hence, its probable rejection by its users. Also, the cost of correcting these errors at a later stage will be higher.

In general, these specifications can be expressed in terms of 'content' (what information is needed) and 'context'. The context may be related to how the

1987 ACM 0-89791-222-5/87/0003/0163 75¢

system is presented to the user (when, how, and in what form the information is presented to the user), how much resource is consumed to generate the needed information (system efficiency considerations), how well the system is adapted to its environment (implementation strategies of the system), or any combination of these three. Figure 1 shows the major steps in the requirement analysis phase: a static representation of the current or proposed system to ensure that it can meet the 'content' needs and a dynamic representation of this system to test its feasibility in meeting the context requirements defined by the user.

```
--------------------------------------------------------------------------
              CURRENT OR PRO-      GENERATION OF
REQUIREMENT   POSED SYSTEM         FEASIBLE             REQUIREMENT
DEFINITION    REPRESENTATION       ALTERNATIVES         SPECIFICATION

--------------------------------------------------------------------------
<DEFINITION>  <-----REQUIREMENT ANALYSIS----------->    <SPECIFICATION>
```

Figure 1: A Framework for Requirement Analysis

The objective of this paper is to extend the well developed modelling capabilities of Petri Nets to represent the information system of an organization and to use PROLOG-based knowledge base to verity the information system representation. The next section will discuss some earlier research with regard to requirement verification. The third section will discuss the basic features of Petri Nets as they are used to model the information system and how this modelling can be represented as factual knowledge in PROLOG. The fourth section will use the analytical properties of Petri Nets to define rules in the knowledge base and their use in the verification process.

In verifying for the content, one needs to test for the 'derivability of the output' requested by the user using 'logic' and 'precedence' information of each process in the static representation. Successive iterations of this step on each output eventually should lead to the definition of the 'boundary conditions' (where the system meets its external environment). When a complex system is represented in a hierarchical manner, the successive iteration of the output derivability step should also ensure consistency between levels in the hierarchy. In addition, one needs to ensure that all the process and data flow/file definitions are consistent, i.e. there exists no redundant or overlapping definition of either the process or data.

The methods that have been used to verify a proposed information systems range from a simple manual technique to complicated automated techniques. One simple method is to have the system reviewed by users and other analysts using techniques such as structured walk-throughs and do's and don't's associated with documentation conventions. Most of these methods are manual and, thus, are time consuming when analysing large complex systems [2].

A number of automated methods are discussed in the literature to assist the analysis and design phases of system development. These methods usually use some type of mechanism to ensure accuracy in the system representation. Software Requirements Engineering Methodology/Requirements Engineering and Validation Systems (SREM/REVS) [1], Systems ARchitects Apprentice (SARA) [27], Systems Descriptor and Logical Analyzer (SDLA) [9], PSL/PSA [25,26], SODA [11] are software products that use such methods. Refer to Olle [12] and Schneider [24] for additional references. Many of these check for inconsistent definition of inputs and outputs, and output derivability. More importantly, many of these tend to be oriented to program verification rather than system verification. SREM/REVS has been primarily used as a tool to test the software of weapon systems in a dynamic environment [4].

PSL/PSA and SREM use automated cross referencing for verification, or provide data flow, control flow and data structure diagrams for manual cross referencing. These aids are good for checking consistency and closure properties such as missing functions. PSL/PSA is shown to be weak, however, in hierarchical representation and dynamic analysis [4]. SREM is shown to be a good methodology but is not a good verification tool [20,23].

Structured Analysis and Design Technique (SADT) [19,21] and DFD [5] are commonly used in business today. Structured analysis techniques use data flow diagrams (DFD) to represent a system. These methods are excellent for user/analyst communication, but these methodologies are primarily manual and have no analytical basis for system verification.

Methodologies such as the Structured Requirements Definition of Orr [13,14] and Jackson System Development (which has a time dimension) of Jackson [7] suffer from inadequacies such as lack of a theoretical basis for verification.

The Petri Net, developed by C.A.Petri [17], is an abstract tool for modelling information flow. It has become a powerful tool for modelling and analysis of computer hardware and software systems [3,6,8,15,16,18]. Petri Nets are found to be useful to model systems that exhibit events that are asynchronous, concurrent or sequential, constrained by precedence relationships, and have varying execution time and frequency. These are based on sound mathematical principles and can be computerized for efficiency in the verification process. For additional discussion of Petri Nets see [10,15]. In this paper only the static properties of Petri Nets are used for system verification within a knowledge base environment.

Petri Net representation for defining the knowledge base

A Petri Net has two types of nodes called places (P) and transitions (T). Places can be the input (I) or the output (O) of transitions. The state of the system is defined by the tokens associated with each place. Tokens are indicated by dots inside the places and the execution of a Petri Net is controlled by the position and movement of these tokens. The placement of these tokens in a Petri Net at any given instant of time is defined by vector M. The structure of a Petri Net is, thus, represented by these three elements T, I(T) and O(T), and M represents the state of the system.

The Petri Nets defined above can be represented as a Petri Net graph. The places are represented as circles, transitions as vertical bars, and the functional relation between places and transitions as directed arcs. The arcs can be directed only between an element of one set (places or transitions) and an element of the other set (transitions or places) . The static properties of a system are represented by such a Petri Net graph and the dynamic properties of a system result from its execution.

An example of a Petri Net representation is shown in Figure 2. Here 'order' and 'catalog' represent input places (or conditions) and 'sales order' represents an output place for the 'prepare sales order' transition. This Petri Net is executed by firing or enabling the transition T1. The transition is fired when the two input places have at least one token in them, i.e. both the order and catalog information are available. The result of such a firing is a token in the output place as shown in Figure 3. For more discussion on the execution of a Petri Net see Sakthivel and Tanniru [22].

The information about the places, transitions, and their relationship can be represented in the knowledge base as shown below. For illustration, a simple order validation system will be used here. The structured system representation of this system is shown in Figure 4 and its equivalent Petri Net representation is shown in Figure 5.

Table 1. Places, Transitions, and their relationships

```
places ([catalog, order, sales order1, sales order2, credit available,
   credit not available, accepted orders, rejected orders, sales log,
   product changes]).

transitions ([prepare sales order, update product changes, update sales
   log, process orders for acceptance, process orders for rejection]).

input (prepare sales order, [order, catalog]).
output (prepare sales order, [sales order1, sales order2, catalog]).
input ( update product changes, [product changes, catalog]).
output (update product changes, [catalog]).
input (process orders for acceptance, [sales order1, credit available]).
output (process orders for acceptance, [accepted order]).
input (process orders for rejection, [sales order1, credit not
                    available]).
output (process orders for rejection, [rejected order]).
input (update sales log, [sales order2, sales log]).
output (update sales log, [sales log]).
```

1. Token Conservation: Since the firing of a transition removes the token from its input, certain places that have no input to them must contain an infinite number of tokens. We can conserve such places, i.e. have it maintain a certain

number of tokens all the time, by making it a part of both the input and output sets of that transition. Table 1 shows how the places such as 'catalog' and 'sales log' are conserved by representing them both as input and output of transitions T5 and T4 respectively. All files are represented typically in this manner.

2. Conflict in transition inputs: If a single place is an input to two transitions, then only one of these transitions can be fired at any given point of time. This type of structure allows for the representation of operational conflicts (two events that cannot occur at the same time). In Table 1, the 'catalog' place is input to transitions T1 and T5. If each of these has to be fired immediately upon receipt of the appropriate inputs, then only one of these two transitions can be fired at any given point of time. These conflicts can be identitied in PROLOG using the following rule:

```
conflict (TR1, TR2):- input(TR1,X), input(TR2,Y), diff(TR1,TR2), shares(X,Y).
```

Note that the predicate 'diff' ensures that the system does not pick the same transitions for conflict evaluation. The 'share' predicate tests for overlap among the inputs.

3. Interpretative transitions: Conditional logic can be explicitly expressed in a Petri Net rather than buried in the structured English associated with many structured methodologies (see Figure 4). For example, if acceptance of a sales order is based on the availability of credit, then two possible output places (accepted order and rejected order) can result from this evaluation. This is illustrated in Figure 2 by having these two output places generated by the firing of appropriate transitions T2 and T3. Such conflict in the transitions due to

conditional logic can be determined in a manner that is similar to file based conflict discussed above except for the following change:

```
files ([catalog, sales log]).
conflict (TR1, TR2):- input (TR1, X), input (TR2, Y), diff (TR1, TR2),
                      shared set(X,Y,Z), files(F), member (Z,F).
```

Note that the variable 'Z' extracts all the common elements between X and Y, and member predicate checks to see if any of these are a subset of (permanent) files.

4. Hierarchical representation: The tokens in places 'credit available' and 'credit not-available' are derived from the sales order value and credit limit. This can be shown in a lower level Petri Net as in Figure 6. This type of hierarchical structure in Petri Nets allows for a top-down, modular representation of an information system consistently. Note that the places 'sales order' and 'catalog' are both input links to the lower level Petri Net and the places 'credit available' and 'credit not-available' become the output links to the top level Petri Net. These places will be referred to as 'link nodes'. The knowledge base representation for this lower level net is shown below:

```
link nodes ([catalog, sales order1, credit available, credit not available]).
input (prepare order value, [catalog, sales order]).
output (prepare order value, [catalog, order value]).
input (process credit availability, [order value, credit limit]).
output (process credit availability, [credit available, credit limit]).
input (process credit nonavailability, [order value, credit limit]).
output (process credit nonavailability, [credit not available, credit
           limit]).
```

Note that these are structurally similar to the knowledge base of the higher level net, except that these are referred to as 'interpretative' transitions, since the execution of these is based on an interpretation of the inputs rather than the existence of a token.

5. Precedence in transition firing: Precedence relationships in a Petri Net are explicit as they include both the triggering mechanism (when each process is acted upon) and informational dependency in their representation. This is contrary to many structured methodologies which need a separate sequence diagram to illustrate process interdependencies. In addition, if a sales log is updated whenever an order is validated, it should be represented primarily as a parallel process to order validation. However, a data flow diagram representation of this situation may obscure such dependency information. The Petri Net representation of sales log update, as shown in Figure 5, treats this parallelism explicitly. To accomplish this parallelism of (T2,T3) and T4, sales order is named as sales order1 and sales order2, even though these represent same document. By adding a synonym relationship as shown below, one can capture such parallelism.

synonym (sales order, [sales order1, sales order2]).

In summary, the Petri Net representation of an information system can be used to create the facts about the problem domain (information system features). It is in fact possible to derive the Petri Net representation from structured documentation such as data flow diagrams. The next section will illustrate how the representation of a Petri Net can be used to verify the information system characteristics.

Petri Net Representation for Information System Verification

Information systems, as demonstrated in the previous section, can be represented as a Petri Net, where places correspond to the availability of data and transitions correspond to the processes that operate on these data. This section will use such a Petri Net representation to perform the checks described in section 2: input and output consistency, boundary definition, output derivability, hierarchical consistency, and precedence relationships.

Input and Output Consistency: Here we are concerned with proper definition of the input conditions that are needed to enable a transition and output conditions that result from such a transition. Inconsistent definition of output places (same place output by two different transitions) is often a result of either improper naming of these output places or incorrect definition of the associated transitions. For example, in Figure 5, if 'sales order' is defined as an output of both 'prepare sales order' (T1) and 'process orders for acceptance' (T2), then the role of each transition is left ambiguous. However, if the output place associated with 'process orders for acceptance' as 'accepted order', then this ambiguity is eliminated. Alternately, the 'sales order' may be an input to multiple transitions such as 'process orders for acceptance' and 'process orders for rejection' as long as the conflicts are recognized. This consistency check can be performed by using the following rule:

```
inconsistent  (X):- places (P), check consistency (P, X).

check inconsistency ([],[]).
check inconsistency ([H,T], X):- multiple member (H,Headl),
      check inconsistency (T, Taill), append (Headl, Taill, X).

multiple member (X,Y):- output (Z,List), output(Z1, List1), diff (Z,Z1),
                 member (X, List), member (X, List1), Y=X.
multiple member (X,[]).
```

These set of rules are used to select each place from the set of all places, send them to multiple member function to see if this place is an output of more than one transition, and return either the name of the place or a null value to the calling routine 'check inconsistency'. In this calling routine, these results are appended. Null value for this operation ensures input/output consistency.

An exception to this rule, however, exists. For example, sales order may be an output of 'prepare sales order' and 'process back order'. Such cases have to be reconciled on a case by case basis.

Boundary conditions: In this example, 'order' and 'product changes' are external inputs to the system, while 'credit available' and 'credit not-available' are the link nodes. Note that the catalog and sales order are the link nodes in the lower level net. In addition, 'rejected order' and 'approved order' are boundaries to the external system, and the 'credit available' and 'credit not available' are the link nodes in the lower level net. Also, places that appear both as an input and output of a transition represent files. These nodes can be identified using the following rule:

```
boundary nodes(X):- places (P), get candidate (P,X).

get candidate ([],[]).
get candidate ([H|T],X):- boundary (H,Head), get candidate(T,Tail),
                 append (Head,Tail,X).

boundary (L,M):- Transitions (T), check transition (T,L,M).
boundary (L,M):- files (F), member (L,F), M=[].
boundary (L,M):- link nodes (K), member (L,K), M=[].
boundary (L,L).

check transition ([],_,_).
check transition([H|T],L,_):- output (H,List), not member (L,List),
                check transition (T,L,[]).
check transition (_,L,L).
```

Here the knowledge based system first selects a place, checks to see if it is a member of any of the transition outputs, a member of files, or a member of link nodes. If it is not any of these, it will then append to a list. Otherwise, it will return a null value. This process is repeated for each place in the place list.

Output derivability: This is primarily concerned with the ability of the system to generate the needed outputs from the defined input conditions. Even though the highest level Petri Net corresponds to information flows in their aggregate form, it is necessary to ensure that the conditional logic associated with each of these output flows is clearly identified, i.e. what input conditions generate each of these outputs, what conditions, in turn, are needed to generate these input conditions, etc., until all the input conditions are shown to be derived, external, conserved, or extracted from a lower/higher level Petri Net. The rules related to precedence can be used to find out how each of the output places is derived.

```
precedes (TR1, TR2) :- output (TR1, O1), input (TR2, I2), diff (TR1,TR2),
          shares (O1, I2), output (TR2, O2), input (TR1, I1),
          not(shares(O2, I1)).
```

Here this precedence rule is applied for each of the output nodes until all the input places of the preceding transition are boundary nodes.

Hierarchical Consistency: All the places that are identified as links to lower/higher level Petri Nets (called 'link nodes') should have appropriate entries in these nets. This derivation is implicit if all the nets are in the same knowledge base since there is no representational difference between the higher level or lower level nets. However, if these are in distinctly different data (facts) bases, then one need to test for their internal consistency.

Each of these nets should be consistent, well defined in terms of boundaries, and have output derivability. One should be able to collapse, if needed, these nets without any effect on the rest of the Petri Net. The collapse should result in the extraction of only the 'boundary nodes' as the free (not an output of any transition) nodes.

Precedence information: Given that the representation requires the identification of what input conditions are needed to generate an output place, the sequential nature of the processes is explicitly represented in a Petri Net. The path derived in the output derivability will also ensure that all the precedence information is incorporated in the representation. The synonym relationship is used here to identify any parallel paths.

In summary, the knowledge base derived from the Petri Net representation and its static properties can be used to effectively verify the system before it is used for performance evaluation and, possible, design.

Conclusions

Requirement analysis in systems development plays a critical role in allowing the analyst and user develop an accurate requirement specification. This requirement specification (both the content of the information to be presented and the context under which this presentation is to occur) is used for an eventual system design. This paper illustrates how a Petri Net representation can be used to capture the knowledge about the system, and use the analytical properties of these Petri Nets to verify the system representation. The structural properties of the Petri Nets are amenable for automation and such automation using PROLOG-based knowledge base is illustrated.

Since one can derive the Petri Net from other structured tools such as data flow diagrams, structured English, and process sequence diagram, the incremental effort to transform a system from these representations to Petri Net is minimal. This transformation allows one to use data flow diagrams for effective user communication and knowledge base, built from Petri Net representation, for effective verification.

References

[1] T.E.Bell, D.C.Bixler and M.E.Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering," IEEE Transactions on Software Engineering , Vol SE-3,No1,Jan 1977, pp 49-59.

[2] B.W.Boehm, "Validating and Verifying Software Requirements and Design Specifications," IEEE Software , Vol 1, No 1, Jan 1984, pp 75-88.

[3] W.Brauer, Net Theory and Applications , Springer Verlag, NewYork, 1980.

[4] G.B.Davis and C.R.Vick, "The Software Development System," IEEE Transactions on Software Engineering , Vol SE-3, No 1, Jan 1977, pp 69-84.

[5] T.DeMarco, Structural Analysis and Systems Specification , Yourdon Press, NewYork, 1978.

[6] C.Girault and W.Reisig, Application and Theory of Petri Nets , Springer Verlag, NewYork, 1982.

[7] M.A.Jackson, Systems Development , Prentice Hall International, London, 1983.

[8] IEEE International Workshop on Timed Petri Nets , Torino, Italy, 1985, IEEE Catalog Number 85CH2 187-3.

[9] E.Knuth, F.Halasz and P.Rado, "System Descriptor and Logical Analyzer," in T.W.Olle, H.G.Sol and A.A.Verrijn Stuart, Information System Design and Methodologies-A Comparative Review ,Proceedings of IFIP WG 8.1 Conference, North Holland Publishing Co., NewYork, 1982, pp 143-172.

[10] T.Murata, "Petri Nets and Their Application - An Introduction," Management and Office Information Systems , edited by Shi-Kuo Chang, Plenum Publishing Corporation, 1984.

[11] J.F.Nunamaker, " A Methodology for the Design and Optimisation of Information Processing Systems," AFIPS conference proceedings , Vol 38, 1971.

[12] T.W.Olle, H.G.Sol and A.A.Verrijn Stuart, Information System Design and Methodologies-A Comparative Review ,Proceedings of IFIP WG 8.1 Conference, North Holland Publishing Co., NewYork, 1982

[13] K.T.Orr, Structured Systems Development , Yourdon Press, NewYork, 1977.

[14] K.T.Orr, Structured Requirements Definition , Ken Orr and Associates Inc, Topeka, KS, 1981.

[15] J.L.Peterson, "Petri Nets," Computing Surveys , ACM, Vol 9, No 3, Sep 1977, pp 223-252.

[16] J.L.Peterson, Petri Net Theory and Modelling of Systems , Prentice Hall Inc., NewYork, 1981.

[17] C.A.Petri, Kommunikation mit Automaten , University of Bonn, 1962; English translation by C.F.Greene Jr."Communication with Automata", supplement 1 to Tech. Report RADC-TR-65-377, Vol I, Rome Air Development Center, Griffis Air Base, Rome, NY. 1965.

[18] W.Reisig Petri Nets-An Introduction , Springer Verlag, NewYork,1982.

[19] D.T.Ross, "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering , Vol SE-3, No 1, Jan 1977 pp 16-33.

[20] P.A.Scheffer, A.H.Stone, W.E.Rzepka, "A Case Study of SREM," Computer , Vol 18, No 4, April 1985, pp 47-54.

[21] D.T.Ross and K.E.Scheman, "Structured Analysis for Requirements Definition," IEEE Transactions on Software Engineering , Vol SE-3, No 1, Jan 1977, pp 6-15.

[22] S.Sakthivel and Mohan R. Tanniru, 'Information System Verification and Validation during Requirement Analysis.using Petri. Nets', School of Management, Syracuse University, Syracuse, New York, 13210.

[23] A.E.Salwin, "A Test Case Comparison of URL/URA and RSL/REVS," Tech. report FS-77-161, Fleet Systems department, The John Hopkins University, Applied Physics Laboratory, Laurel, Maryland.

[24] H.J.Schneider, Formal Models and Practical Tools for Information Systems Design - Proceedings of IFIP TC.8 Conference North Holland Publishing Co., NewYork, 1979. Verlag, NewYork, 1980, pp 307-320.

[25] D.Teichroew, "Problem Statement Analysis: Requirements for the Problem Statement Analyzer [PSA]-ISDOS", Department of IE, Univ. of Michigan, AnnArbor, 1977.

[26] D.Teichroew and E.A.Hershey, " PSL/PSA-A Computer-Aided Technique for Structured Documentation and Analysis of Information Systems," IEEE Transactions on Software Engineering , Vol SE-3, No 1, Jan 1977, pp 41-48.

[27] J.W.Winchester and G.Estrin, "Requirements Definition and its Interface to SARA Design Methodology for Computer Based Systems," National Computer Conference , 1982, pp 369-379.

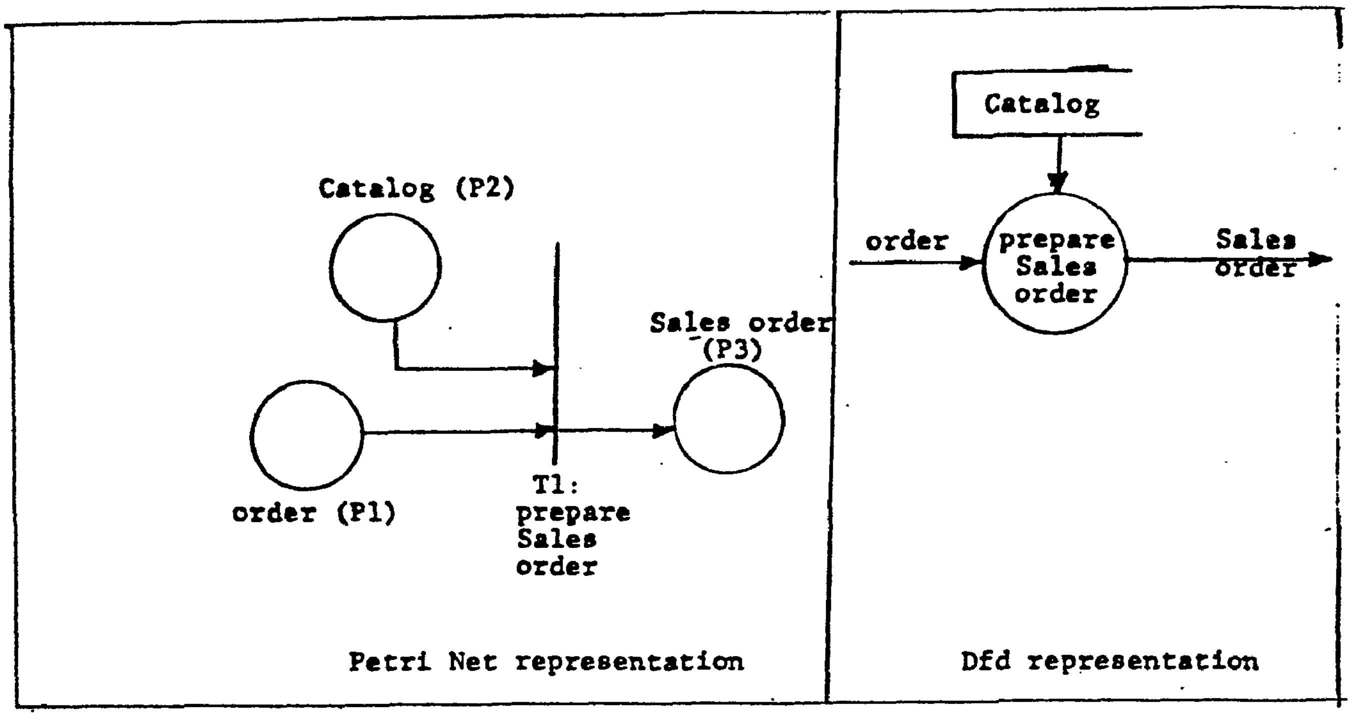


Figure 2: Petrinet and Dfd representation of order preparation process

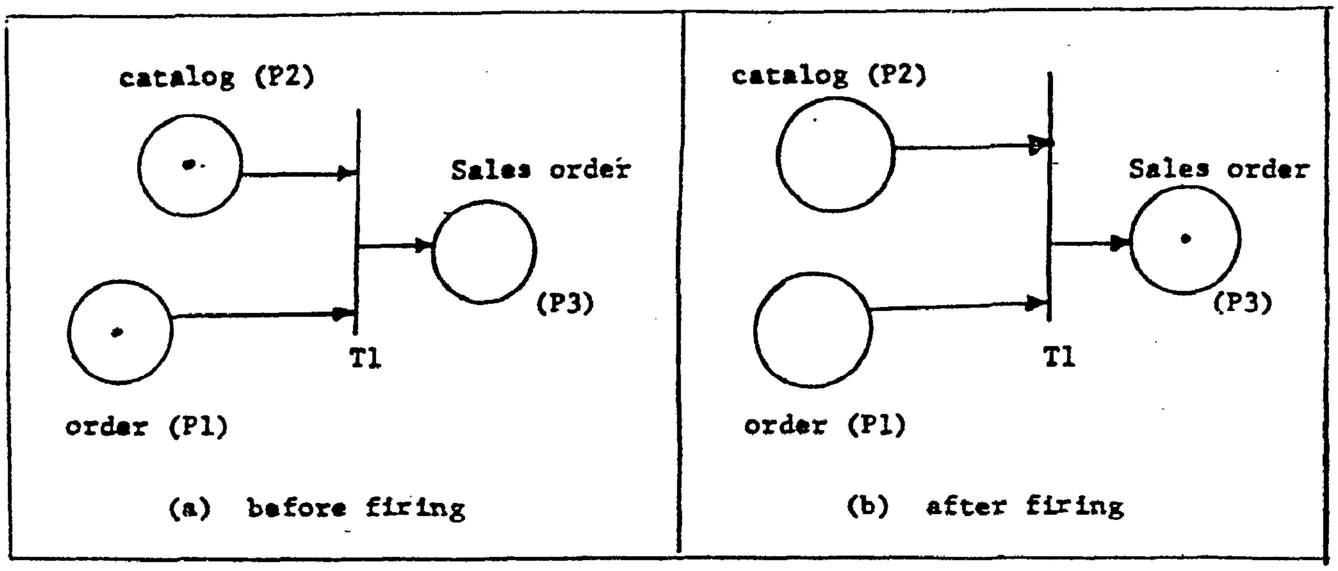
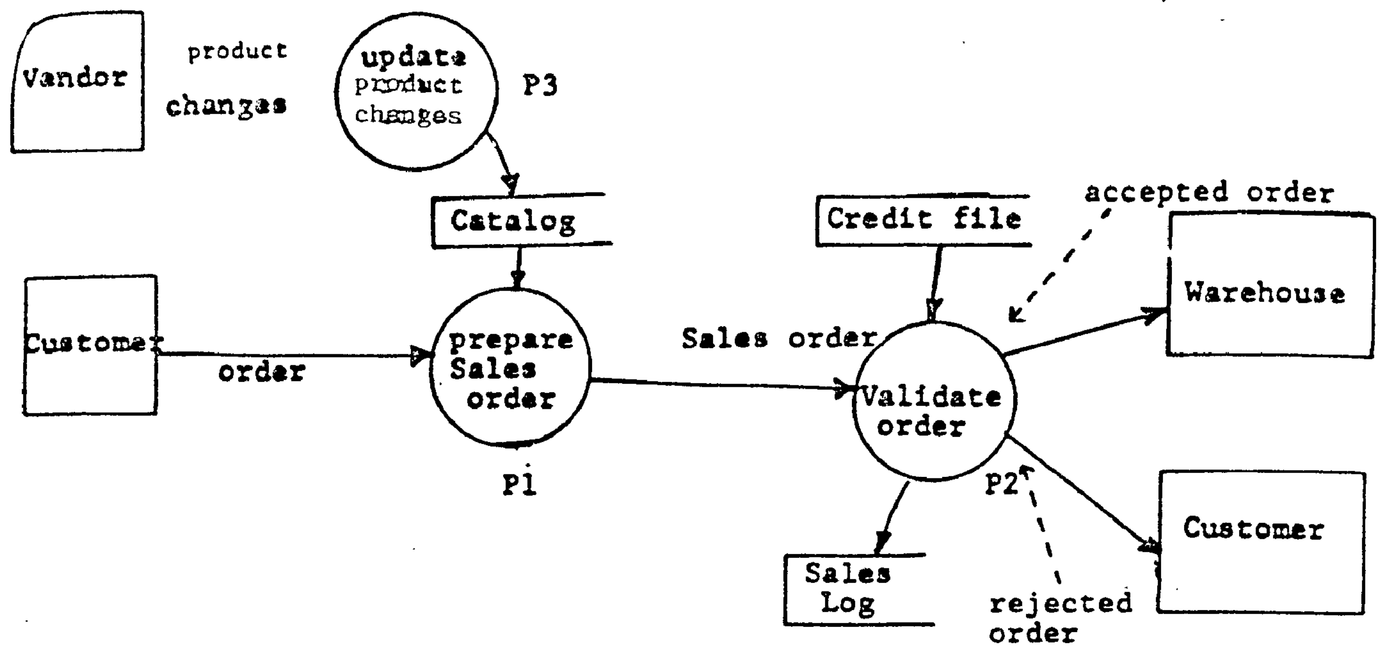


Figure 3: Execution of a Petrinet



P1: Upon receiving an order from a customer
- get product price from catalog;
- prepare a salesorder using product price information.

P2: Upon receiving a sales order,
- record the order arrival in a Sales log;
- obtain credit limit for the customer from the credit file;
- compute order value;
- if order value $\leq$ credit limit send the accepted order to warehouse,
- else (order value $>$ credit limit) send the rejected order to the customer.

P3: Upon receiving price changes from a vendor,
- record these changes in the product Catalog.

Sequence Diagram:

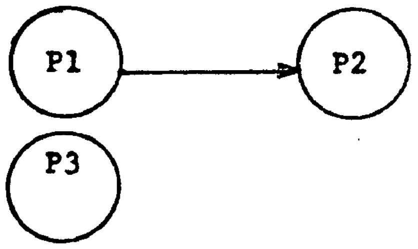


Figure 4: Structured representation (data flow diagram, structured English and sequence diagram) of an "order validation" system.

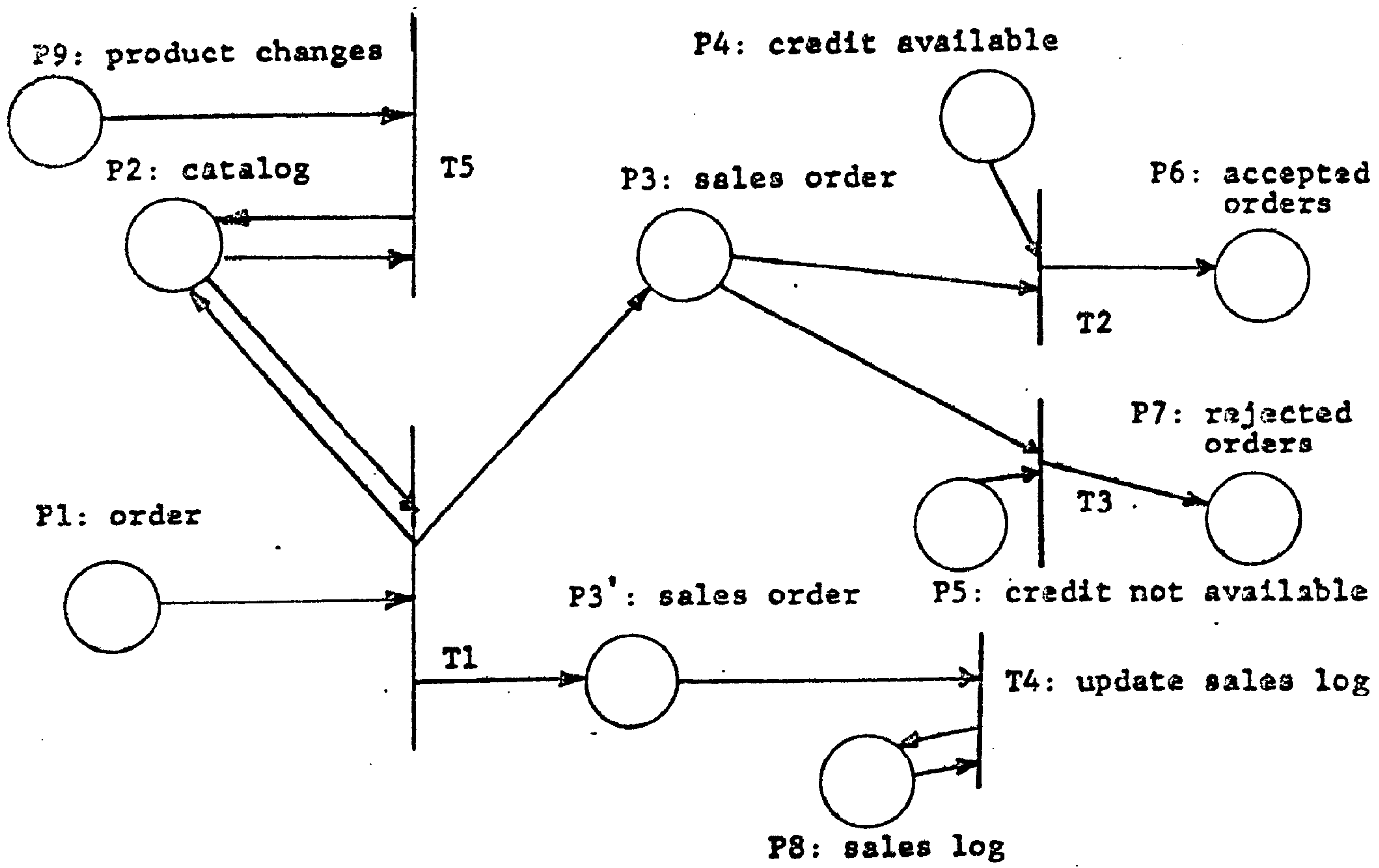


Figure 5: Order Processing System

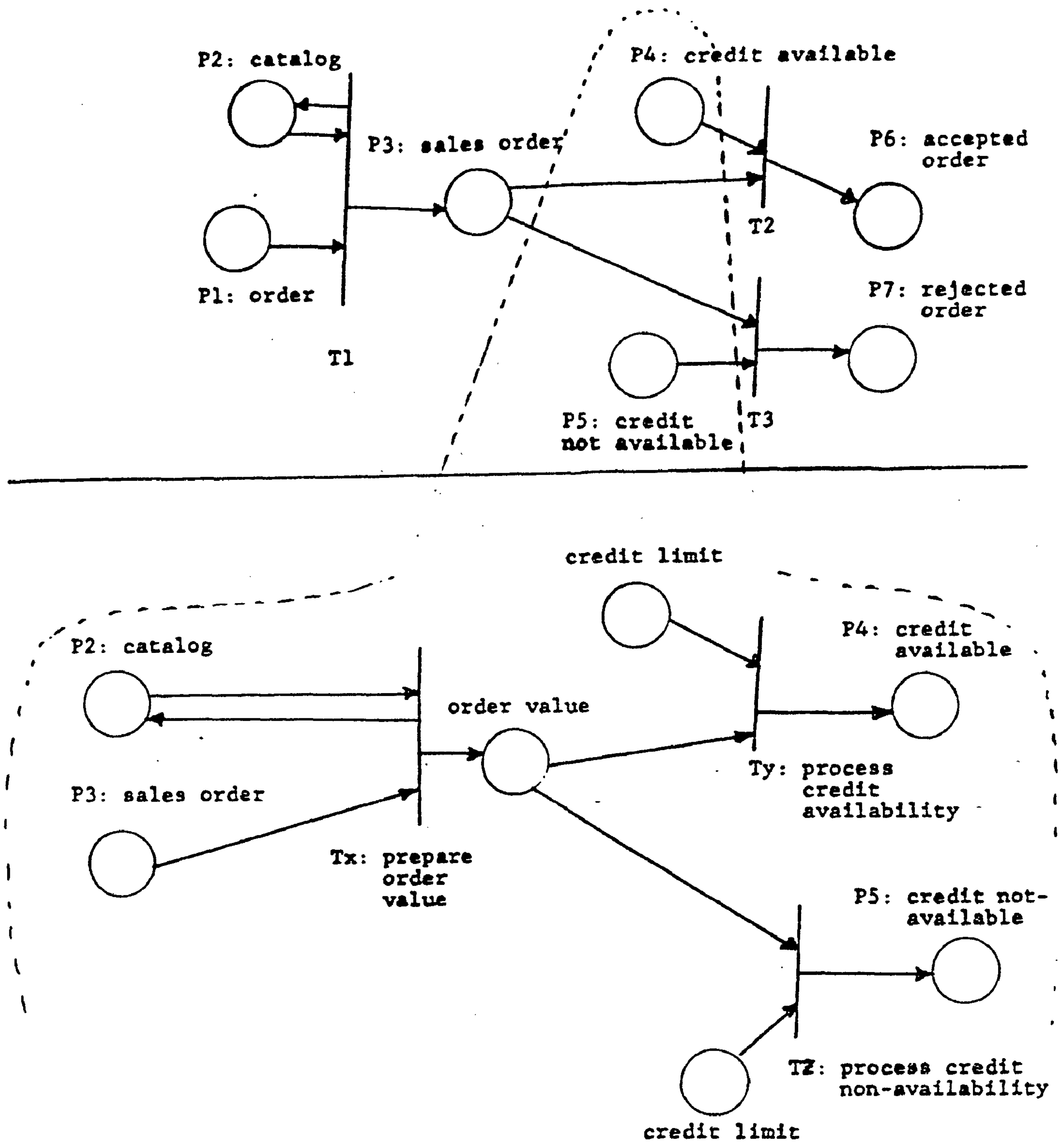


Figure 6: Hierarchial decomposition of a Petrinet