VERIFICATION OF A LOCAL AREA NETWORK PROTOCOL WITH TINA, A SOFTWARE PACKAGE FOR TIME PETRI NETS

J-L. ROUX, B. BERTHOMIEU

Centre National de la Recherche Scientifique
Laboratoire d'Automatique et d'Analyse des Systèmes
7, Avenue du Colonel-Roche
31077 TOULOUSE, FRANCE

Abstract

TINA is a software package for computer aided description and verification of time dependent systems, among such systems are communication protocols, for instance. The systems are expressed as Time Petri nets (Merlin's extension of Petri nets); TINA implements the enumerative analysis method known for these nets and provides facilities for editing nets and processing the results of analysis. The use of the package is illustrated with the specification and verification of a bus access protocol for a local area network.

INTRODUCTION

This paper focuses on analysis of concurrent systems in which time is part of the specifications. Real time systems and communication protocols belong to that class of systems; functional properties may be time dependent, and performance requirements are often expressed by means of temporal constraints. Time Petri nets /Merlin 74/ were choosen for modeling these systems because of their ability to handle these various aspects.

An enumerative analysis technique for Time Petri nets was proposed in /Berthomieu 83/. This method has been proven convenient on a number of significant examples, but it necessarily requires the help from a computer

for managing the construction of the graph of state classes, which is the basis of the analysis method. TINA is an implementation of the method; it also includes facilities for editing Time Petri nets, and a set of tools for conveniently extracting information from the graph of state classes. Evaluation of the tool is in progress, by processing meaningfull examples.

The following section recalls the necessary background on Time Petri nets and summarizes the enumerative method; section 2 describes the features of TINA; and a significant example of use is reported in section 3. The TINA session transcripts for the example in section 3 are given in appendix.

1. TIME PETRI NETS

1.1. Terminology and behavior

Time Petri nets (TPNs) are obtained from Petri nets by associating a time interval 3a,b3, with 0 <= a <= b, with each transition of the net. Assume that transition t last became enabled at time u, then it must fire between u+a and u+b, unless it is disabled by the firing of another transition.

States in TPNs are pairs (M,I) consisting of a marking M and for each transition k enabled by M, a firing interval I_k specifying the time range in which the transition is allowed to fire. It must be pointed out that these intervals may dynamically differ from the Static Firing Intervals initially assigned to the transitions. The state change function for Time Petri nets may be stated as follows:

Firing a transition t, at a time θ, from a state (M,I) is allowed iff both the transition is enabled by marking M, and time θ is comprised between the earliest firing time (EFT) of transition t and the smallest of the latest firing times (LFTs), among those of the transitions enabled by marking M.

The next state (M',I') is computed as follows:

(i) The new marking M' for place p is defined, as usual, as:

$$M'(p) = M(p) - B(t,p) + F(t,p)$$

where B and F are the Backward and Forward Incidence Functions of the net, respectively.

(ii) The new firing intervals I' for transitions are computed as follows:

- For all transitions n not enabled by marking M', I'_n is empty;
- For all transitions k enabled by marking M and not in conflict with transition t for marking M, then:

$$I'_k = 3\ Max(0,\ EFT_k - \theta),\ LFT_k - \theta\ 3$$

where EFT_k and LFT_k denote the EFT and LFT of transition k, respectively;
- All other transitions receive their static firing intervals.

More details on the firing rule and examples may be found in /Berthomieu 82/ and /Berthomieu 83/ where, in particular, solutions are presented to deal with multiple enabledness of transitions.

The above firing rule defines a reachability relation among states of a Time Petri net. A Firing Schedule is a sequence of pairs (transition, time) such that the transitions in the sequence may be successively fired, at their corresponding times.

1.2. The enumerative method for analyzing Time Petri nets

The behavior of a TPN is characterized by its set of states reachable from its initial state or, alternatively, by its set of firing schedules feasible from its initial state. Unfortunately, as time is continuous and as transitions may fire at any time in their allowed interval, states have generally an unbounded number of successors; which forbids an enumerative analysis based upon states.

State Classes have been introduced to overcome this problem. A State Class will be associated with each firing sequence, defined as the union of all states reachable from the initial state by firing schedules with this firing sequence. More formally, the firing domain of a state being defined as the product set of the firing intervals of the transitions enabled, the class associated with sequence s is the pair (M,D) in which M is the marking reached from the initial marking by firing sequence s, and D is the union of all firing domains of states reachable from the initial state by firing schedules with sequence s.

It may be shown that Firing Domains in state classes are convex sets; they may be expressed as solution sets of some systems of linear inequalities $A.\underline{t} >= \underline{b}$, in which A is a matrix of integers, $\underline{b}$ is a vector and variable $\underline{t_i}$ is associated with the i^{th} transition enabled by the marking.

A transition rule may be directly expressed for state classes:

The initial class is defined as the class containing only the initial state. For this class, the firing domain is simply the solution set of the system $a_i <= \underline{t_i} <= b_i$, i ranging over the number of enabled transitions, $\underline{t_i}$

corresponding to the i^{th} transition enabled and a_i, b_i being its Static EFT and LFT respectively.

A transition t (assume it is the i^{th} enabled) is firable from a class (M,D) iff both the following conditions hold:
- t is enabled by marking M, i.e. M(p) >= B(t,p) for all places p;
- transition t may fire the first among the transitions enabled by M, i.e. the following system of inequalities is consistent:

  $A.\underline{t} >= \underline{b}$

  $\underline{t_i} <= \underline{t_j}$ for all variables $\underline{t_j}$, $j \neq i$.

Computation of the successor class (M',D') is done as follows:
1) Compute new marking M' as for Petri nets;
2) Compute new domain D' in four steps:
   a) Augment the system $A.\underline{t} >= \underline{b}$ with the above firability conditions for transition t;
   b) Eliminate from this system the variables associated with transitions in conflict with t;
   c) Express each remaining variable $\underline{t_j}$, with $j \neq i$, as the sum of variable $\underline{t_i}$ and a new variable $\underline{t'_j}$ and eliminate $\underline{t_i}$ from the system;
   d) Add one variable for each newly enabled transition, constrained to belong to the Static Firing Interval of the transition it is associated with.

Full justification of the rule may be found in the reference. The rule allows building a tree of state classes from the initial classes. Two classes are defined equal iff both their markings and their firing domains are equal. Due to the simplicity of the inequations involved, an efficient algorithm may be found for comparing domains for equality. The graph of state classes is obtained from the tree by merging equal classes.

Further, it is known from /Berthomieu 83/ that a Time Petri net admits a bounded number of state classes iff it is bounded, which is an undecidable problem /Jones 77/. Fortunately, usable sufficient conditions can be found. The following condition has been proven adequate for most of the applications we have in mind.

A Time Petri net is Bounded if no pair of state classes C=(M,D) and C'=(M',D') reachable from its initial state class are such that:
(i) C' is reachable from C;
(ii) for all places p: M'(p) >= M(p),
  and for at least one place p: M'(p) > M(p);
(iii) D'= D.

Stronger conditions are discussed in /Berthomieu 82/. It may be pointed out that line (ii) corresponds to the necessary and sufficient condition given in /Karp 69/ for the boundedness of usual Petri nets and Vector Addition systems. Although it allows proving bounded a large class of TPNs, it appears too weak from the meaningfull examples we have treated so far.

The graph of state classes of a bounded TPN will be used, as a finite representation of its graph of states, to check the properties characterizing the correct behavior of the system represented by the net.

2. THE TINA ANALYZER

2.1. Outline

TINA (for TIme petri Net Analyzer) is a prototype software package designed for handling Time Petri nets.

The functions provided by TINA may be split into three classes: net editing compiling and loading; enumeration of state classes, using the method described in section 1; and tools for extracting information from the graph of state classes.

Nets are input in textual form, using an editor. Net definitions may be loaded into the analyzer, which in effects compiles net definitions into some internal representation. Several nets can be merged together at loading time. Compiled net definitions can be saved for later use.

The package for enumerating state classes maintains a high level of interactivity with the user. The enumeration is run after a number of options are set, including an increment on the number of classes. The user may also provide properties to be incrementally checked on state classes, such as marking invariants. When an exception is raised, the user may either abort, suspend or resume enumeration. In any case, partial or total informations about the graph of classes, the classes themselves and the schedules of the graph of classes are available upon request.

When a TPN has been proven bounded, the required properties for the model can be investigated using its graph of state classes. Liveness and Cyclic properties are defined for Time Petri nets as they are for Petri nets and proven in a similar way, using the connexity structure of the graph of classes. Finally, utilities such as path finding functions, for instance, help reading the content of the graph of classes.

2.2. Packages

Access:

The TINA system is a single-user interactive software tool. An information menu can be displayed, giving the main functions available at top-level; more specialized functions are proposed where useful. TINA is written in APL, the user working space contains a copy of the system, together with its private library of net definitions and result of analysis. The whole workspace can be saved in a disk file and later reloaded in the APL environment for further work. Access to TINA consist of calling the APL interpreter, loading the desired working space, and eventually saving the work done before closing the session.

Editing, compiling and loading nets:

Nets are edited using the APL function editor; designing a specialized editor did not appear necessary since the APL editor is convenient enough. The data structure describing the net is input as an APL function, whose name is the name of the net. A net is described as a set of transition declarations and an initial marking assignement. An example of net definition is given in table 1 of appendix.

Each transition declaration must contain a transition identifier, optionally followed by its static firing interval (default value is 30, infinite3); its input places (if any) separated by comas and optionally ascribed a weight (default 1); and finally its output places (if any). The exact format of transition declarations is the following, where optional occurrences are enclosed in brackets:

```
TrId (°a,b§) : (PlId(xN) (, PlId(xN))) -> (PlId(xN) (, PlId(xN)))
```

Spaces are optional; transition and place identifiers must begin with a letter and may only contain letters or digits; N is the weight ascribed to the edge connecting the corresponding place and the transition. An arrow delimits input and output places.

A net definition must also assign initial markings to the places for which this marking is not null. The format of marking assignement should be clear from the example given in table 1 of appendix. Finally, comments are allowed with the same format as comments in APL functions.

The APL function editor allows to list, augment, comment and modify the description of the net. TINA supports also some net library management functions; utilities are provided for listing the names of defined nets, renaming or deleting some nets, and merging several net definitions.

Once defined (as an APL function), a net can be loaded into the analyzer. loading a net consists of parsing the definition of the net, skipping comments, and compiling it into some internal representation suitable for fast enumeration of the state classes. The tables produced correspond, as a matter of fact, to the data structure of the net (input and output matrices, time intervals, initial marking, place and transition names). Care has been taken so that, in normal practice, the use need never to see or manipulate the internal representation of nets. In case the definition of a net is not syntactically correct, loading is aborted and the analyzer reverts to the state in which it was before.

The loader allows also to merge, at loading time, several net definitions, and to save their composition, in textual form, under a given name. But, once a net loaded, its components cannot be modified; if the analysis does not show the expected behavior for the net, a new textual description must be edited for it (possibly starting from the old description) and it must be loaded again.

The enumeration package:

Behavior analysis constitutes the core of the system. The analyzer applies the firing rule for state classes, starting depth first from the initial class, and builds the graph of state classes of the last TPN loaded.

A number of parameters may be set by the user for managing enumeration. These allows to interrupt the enumeration by setting increments for the CPU time and the number of classes. The user may also choose its boundedness sufficient condition among several, of increasing strength; or specify some conditions to be incrementally checked when enumerating the classes. These conditions must be entered as APL boolean expressions, through the 'option' command of the package. Predefined conditions include safeness and non-multiple enabling for transitions. User defined specific conditions, such as marking invariants, may be entered as well, but defining them currently requires knowledge of APL and of the internal representation of data in the analyzer; it is scheduled to introduce some language for expressing user defined conditions but this remains to be done.

For each transition firable from a class, a new class in computed, constituted of a marking and a firing domain; the class is compared for equality with those previously enumerated and the graph of state classes is updated accordingly. If the class is new, the boundedness sufficient condition and the optional user defined conditions are checked for this class; in any case, overflow conditions for CPU time and number of classes increment are checked before proceeding from the next non developed class.

When a class fails to satisfy a predefined condition, or when overflow occurrs on one of the increments, the anomaly is displayed and enumeration is interrupted. The user may then either abort enumeration, suspend it or resume. If suspended, enumeration may be resumed later from the point where it has been interrupted, eventually with different enumeration options. Partial results (the current graph of state classes and its content or part of it) may be examined at any time when at top level. Results of analysis may be stored under the name of the net for later use. Partial or total results are displayed in clear, in the language in which the net has been defined. An example of analysis session is given in table 2 of appendix.

Extracting information from the graph of classes

When the Time Petri net has been found bounded, its graph of state classes has been produced, together with information about its connexity. A set of standard utilities is provided for checking the liveness and cyclic properties of the net. For tricky cases, one may also have complete information about the strongly connected components of the graph of state classes (displayed as a graph of equivalence classes over classes). An example of liveness analysis is shown in table 3 of appendix.

The cyclic property for Time Petri nets deserves a comment. Let us recall that a TPN is cyclic if its initial state class can be recovered from any other. A live TPN, as for usual Petri nets, may be not cyclic. This is actually the case for most TPNs, due to temporal reasons. Most TPN models of actual systems exhibit a transient behavior before reaching a steady state behavior; this because the configuration of time intervals initially assigned to the enabled transitions is never recovered in normal behavior.

Other utilities report deadlocks and dead transitions, giving the classes from which comes the diagnosis. Combined use with firing sequence finding facilities allows detecting design errors faster. Saving the whole analysis results is possible and may be advantageous for further investigation on the same net.

Finally TINA may be connected with other existing packages not described here; among them are a package for structural analysis of Petri nets, and general purpose packages for graph and automata manipulation.

Performances:

Implementation and performance considerations are now addressed. TINA runs in APL on an IBM 3081 computer; experiments have been made with TPNs with up to 100 places and 100 transitions, producing more than 3000 classes. It becomes relatively slow as the number of classes grows, and depending on the parallelism exhibited by the net. But performances are in the line of what one can expect from an software written in an interpreted language.

TINA is used in the following section for processing a meaningfull example; efficiency is preserved by using an adequate proof methodology to control the state explosion.

3. PROWAY EXAMPLE: BUS ACCESS PROTOCOL ANALYSIS

PROWAY is an industrial Local Area Network /Auger 81, Kryskow 81/ linking functional units by means of a shared hardware bus, through which they communicate. Cooperation between the stations is ruled by the Highway Unit Protocol /CEI Part.3 84/, which specifies the frame structure, the message processing and the line access mechanism.

3.1. Bus access protocol description

Each station is organized in a three layer architecture, in a way similar to the OSI model. The bus allocation procedure is performed by the data link layer, and is based on the token bus access technique /IEEE 802.4 84/. Stations are distributed on a logical ring, independently of the physical organization. A baton gives to stations the control of the bus. Normal behavior is as follows: when a station has the baton, it can send an application message, whereas the other stations can only answer to it upon request. When the transaction is completed, the baton is passed to the next station on the virtual ring.

Baton-passing is based on a local live stations list, which represents the logical ring made of the 100 station adresses encompassed by a complete cycle of baton transfers. When a baton frame is detected on the bus, the entry in the live list corresponding to the source address is updated; i.e.

a "live" indication is assigned to that station.

Normal behavior may be altered in three main ways:

1) Messages losses may occur at emission or at reception. Assume that a unit, after using the bus, wants to transmit the baton; it checks the next entries in its live list.

- If the next live station, known as the Next Live Address, is located immediately after the sender, it receives the baton. If the sender does not listen any activity on the line a certain time later, a frame interval timer T1 times out. The sender supposes that the baton has been lost and does up to three retries. In case of repeated failures, the receiver is discarted from the local live stations list, and the baton is transmitted to the new next live unit on the ring, according to the list indications.

- If an Address Gap exists between the station's own address and its Next Live Address, the sender executes a Gap Searching Procedure. The search involves a sequential check of the non-live gap addresses. Only one gap address, said the Next Gap Address, is tested each time the station holds the baton. If the attempt in passing the baton fails, the Next Gap Address is incremented by one and the Next Live Address receives the baton.

2) A station may become permanently "dumb", and may induce a durable lost of the baton, if it has the baton. The other stations in the ring must detect that the baton-passing cycle is broken. Consequently each one has a lost baton timer T2, whose purpose is to initiate a new baton if the current baton appears to be lost. The value of T2 is indexed with the station's own address, in such a way that the live station with the smallest address monitors the recovery.

3) A station may become permanently "deaf" so that it does not detect any activity on the bus. If the faulty unit has the baton, it will be unable to pass it correctly. Because of no detecting transitions on the line after sending the baton, it will perform three retries, that may lead to a duplication of the baton on the ring. Recovery from this error is based on messages analysis. When a allowed sender receives an unexpected frame denoting another active sender, it gives up its bus controller status to come back to the bus listening state.

3.2. Bus access protocol modeling

The distributed and complex structure of the protocol needs the use of formal modeling and analysis techniques to verify its properties. Time Petri nets constitute a suitable tool, because they keep the preciseness and formal background of Petri nets, and allow direct expressing of the temporal constraints through the firing intervals associated with the transitions.

The way a station accesses the transmission medium and their behaviors are essentially identical for all stations. So we could use the same Time Petri net for modeling each unit. The global model for the system is made up of several instances of this net connected together /Ayache 82, Voss 84/. Additional hypotheses have been made to simplify the model. Only one application message is involved in each transaction, and the sender cannot hear its own transmission in normal behavior.

Figure 1 gives a Time Petri net model for station 1 of the PROWAY system, in a four unit configuration. Recovery mechanisms are represented but not the fault hypothesis. Time values associated with the transitions correspond to the standard specifications; the allowed working time for the stations has been computed such that it is compatible /Roux 85/. The station live list for unit i is restricted to the entries i+1 and i+2, because of the single fault hypothesis; we assume that any error is recovered before another one occurs. Hence i passes the baton to the Next Live Address i+1, and if a loss is detected, to i+2. In the next cycle, i tests the Next Gap Address i+1, and if a new loss is detected the baton is passed to the new Next Live Address i+2. From this time on, faulty unit i+1 is excluded from the ring. The local station live list is automaticaly updated upon detection of a baton frame from the successor station.

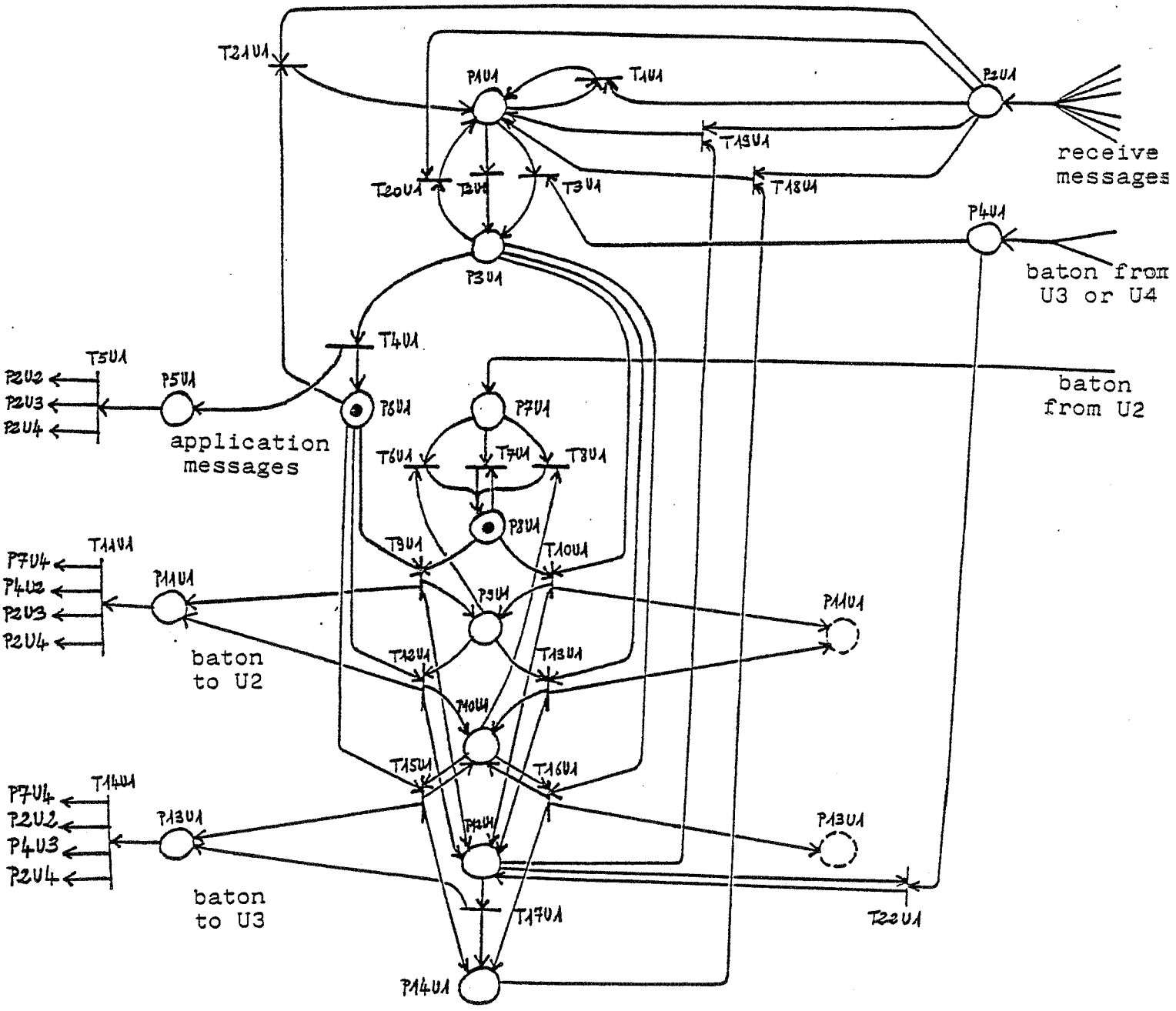


Figure 1

A TPN model for the PROWAY Bus Access Protocol

(Suffix **Ui** on places and transitions names refers to the units)

List of places:

P1U1: **U1** listens to the bus, for detecting message frames.
P2U1, P4U1: Receive places for application messages and batons.
P3U1:**U1** has accepted the baton, but must wait a minimun time before transmitting.
P5U1: Application message sent, to be broadcast.
P6U1: **U1** controls the bus, supervising the transaction.
P7U1, P8U1, P9U1, P10U1: Local station live list.
P11U1, P13U1: Baton sent to **U2** and **U3** respectively, to be broadcast.
P12U1, P14U1: Wait places for detection of transitions-on-line.

List of transitions:

T1U1(0,0): Processing of an application message, or a baton not addressed to **U1**.
T2U1(260,300): Expiration of the local lost baton timer **T2**; its value is defined as (200 + 80 x <i>) $\pm$ 20 us, <i> = address of **Ui**.
T3U1(0,0): Acceptance of the baton from **U3** or **U4**; detection times are neglected.
T4U1(16,24), T5U1(0,10): Sending and broadcast of an application message.
T6U1(0,0), T7U1(0,0), T8U1(0,0): Update of the station live list.
T9U1(50,100), T10U1(16,24): Sending of baton to **U2**; (50,100) is the allowed working time interval for the stations.
T11U1(0,10) (resp. T14U1(0,10)): Broadcast of baton addressed to **U2** (resp. **U3**).
T12U1(50,100), T13U1(16,24): Sending of baton to **U2**, after one failure in the last cycle.
T15U1(50,100), T16U1(16,24): Sending of baton to **U3**; **U2** is excluded from the ring after two successive failures in the preceeding cycles.
T17U1(50,53): Expiration of the local frame interval timer **T1**.
T18U1(0,0), T19U1(0,0): Detection of transitions-on-line certifying correct baton-passing.
T20U1(0,0), T21U1(0,0): Detection of an unexpected message on the line, while **U1** has the baton; it drops it and returns to the listening state.
T22U1(0,0): Rejection of the baton because **U1** has just used it.

Figure 1
(continued)

The global net model obtained by interconnecting four instances of the net of figure 1, represents the normal error-free behavior of the protocol. Table 1 of the appendix presents the way how the net has been edited using TINA.

3.3. Bus access protocol verification

Under the transient losses, "deaf" and "dumb" fault assumptions, and the single fault hypothesis, the bus access protocol must satisfy the following requirements /Menasche 83/:

(i) At most one station has the control of the bus at any time; i.e. no marking of the graph of state classes is reachable such that several stations have a baton.
(ii) The loss of the baton may not be permanent; i.e. there exists no circuit in the graph of classes that goes only through state classes in which no station has the control of the bus.
(iii) The cycle of baton transfers must respect the logical location of the stations on the logical ring; i.e. in all paths of the graph, classes are ordered in such a way that stations 1, 2, 3, and 4 (except for misbehaving units) are successively given the control of the line.

Proof of these properties has been conducted on the graph of state classes of the global Time Petri Net model. Tables 2 and 3 of the appendix describe the computer analysis session. Enumerative analysis of the error-free behavior produces 180 state classes. Exhaustive check of the markings in the classes shows that property (i) is satisfied. In the same way, no marking exists such that all stations are in the listening state; this fact is sufficient to prove property (ii) in the error-free case. Property (iii) is checked by applying automata reduction methods (not discussed in section 2 but available as a specialized package); the graph of state classes for the permanent behavior can be reduced to four nodes, each one corresponding to the baton-holding time by a given station. The cycle thus obtained shows that stations 1, 2, 3, and 4 control successively the bus.

In presence of errors, the single-fault assumption allows us to verify the behavior of the protocol by analyzing each fault seperately. If the protocol recovers a state of the error-free model from each of the three above fault hypotheses taken individually, it will recover its correct functioning from any sequence of these faults. So a distinct net is used to model the different fault hypotheses, including only one faulty station among the four making the system.

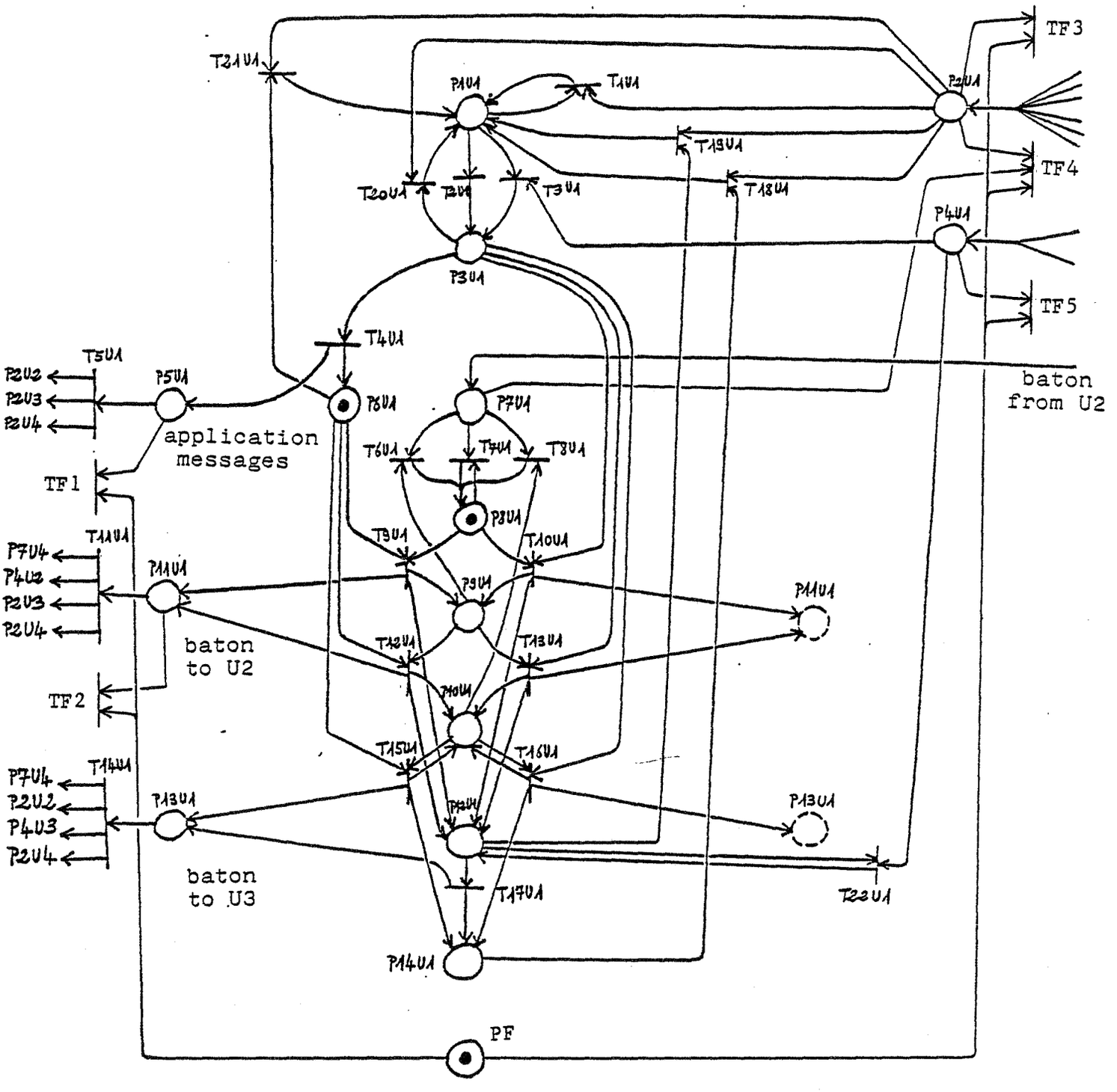
Figure 2 represents station 1 with the transient losses of messages. The analysis of the corresponding global net led to modify one protocol parameter to ensure a correct behavior. The station response time, when it receives a valid frame addressed to it, must be greater or equal than 16 us. This change has been reported previously on the figures 1 and 2. With this modification, the three required properties have been verified. When a loss occurs, the station which is responsible for baton-passing keeps the control of the line and initiates a recovery upon expiration of its time-out T1. The faulty station may be excluded from the current cycle, but reintegrates into the ring at the next one.

Figure 3 models the "deaf" and "dumb" fault hypotheses. Deletion of the outgoing (resp. incoming) arcs expresses the dumb (resp. deaf) behavior of a unit. Analysis of the global corresponding nets produces their graphs of state classes from which it comes that:

- With a dumb unit, a correct behavior for the system implies modifying the values of time-outs T2. Otherwise two recoveries from two different stations may interfere. The changes have been notified previously on the figures. The "dumb" station is excluded from the cycle of baton transfers within two turns; but it still receives application messages.

- In the other case, a misbehavior may appear if the station is in the listening state (without the baton) when it becomes deaf. Its time-out T2 will expire, initiating a recovery when the ring is functioning correctly. To deal with that case, the time-out mechanism T2 can be modified as follows: on figure 3 a local loop must be associated with the lost baton timer transition. Thus when T2 expires, a local check of the coupler internal circuitry is performed. Detection of a failure at the transmitter (dumb unit) or at the receiver (deaf unit), prevents the station from starting a recovery, by locking it in the local state, disconnected from the bus. The analysis conducted with that method, shows that the deaf station is removed from the ring within three turns at most. The remaining stations behave normally, accessing the line in the correct ordering.

This last protocol model has been analyzed with the three fault hypotheses. The graphs of state classes of all global nets produced show that no inconsistencies appear in the protocol behavior when operating in a four unit configuration.



TF1(0,10), TF2(0,10): Loss, at emission, of a broadcast message, application message or baton respectively.

TF3(0,0), TF4(0,0), TF5(0,0): Loss, at reception, of an application message or a baton.

Single-fault hypothesis is modeled by the extra place PF marked with one token. No fault may occur when station 1 is running a recovery (e.g. the baton addressed to station 3 may not be lost).

Figure 2

Transient fault hypothesis for station 1 of PROWAY

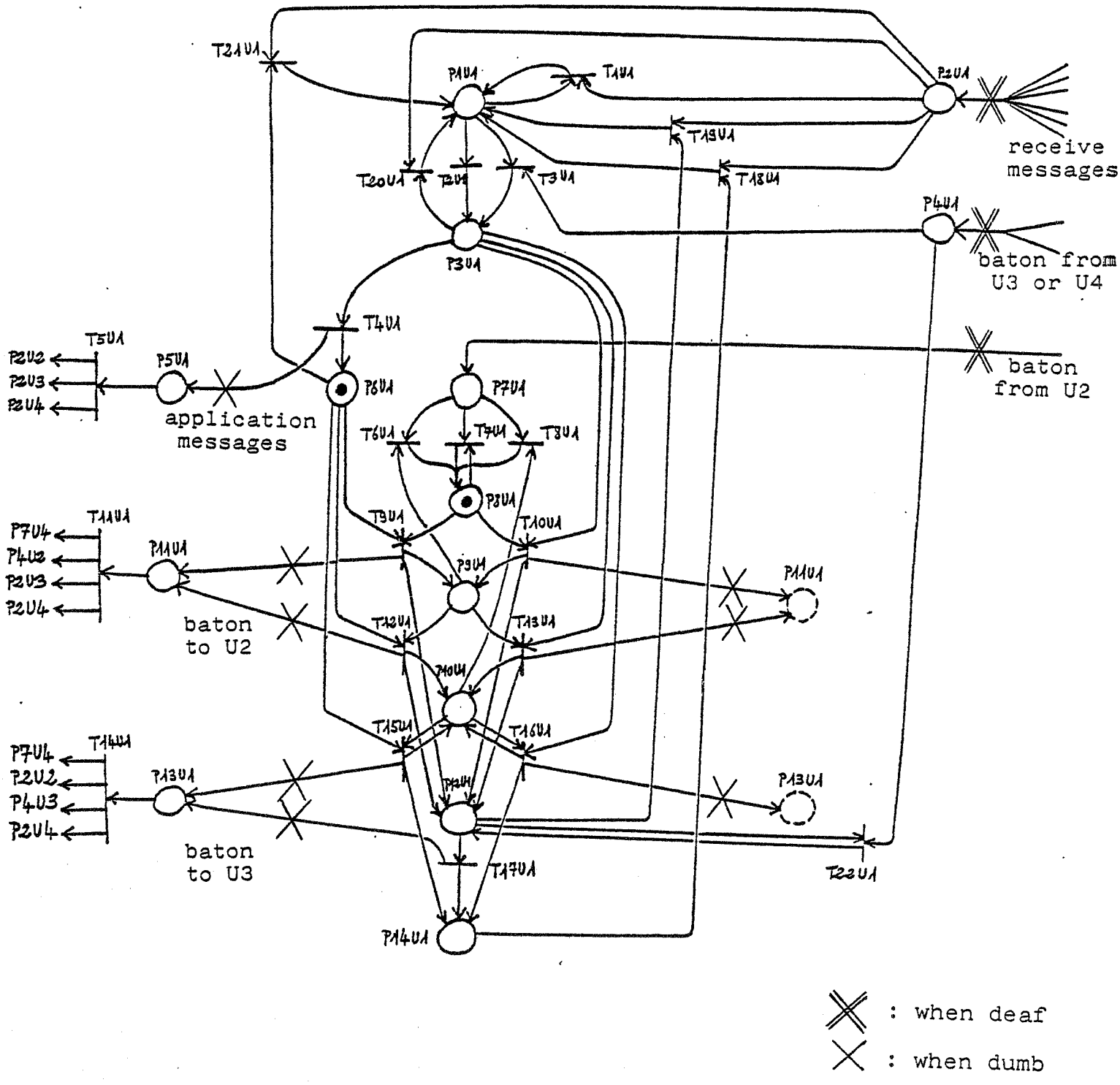


Figure 3

"Deaf" and "Dumb" fault hypotheses for station 1 of PROWAY

One must be careful in extending the proof of correctness to an arbitrary number of stations. Arguments such as: the diffusion like medium allows the number of stations varying without altering its features; the unprecision of time-outs can be expressed exactly with TPNs; or the time intervals associated with the transitions represent all actual behaviors defined by the specifications, are not sufficient to conclude for the correct behavior whatever the number of units is.

Another point of interest with the example studied was the protocol performance evaluation. Experiments have been carried out with the normal behavior and with the transient fault hypothesis. By examining the contents of the state classes and the firing schedules of the graph, we could devise on one hand the minimun and maximun times a station can keep the baton, on the other hand the maximun recovery times induced by transient losses /Roux 85/. This performance analysis is limited by the complexity of the graph of state classes. So investigations on more efficient schedule finding algorithms are currently in progress.

CONCLUSIONS AND FURTHER WORK

The PROWAY experiment induces some comments about the analysis method and the software package.

The number of state classes may grow very large (e.g. around 1000 for instance for the PROWAY ring with the transient fault hypothesis). This increases the cost of analysis, if state explosion is not controlled by an adequate proof methodology. With the PROWAY example, the same superposition like analysis method of considering separately each kind of faults, elsewhere successfully applied /Menasche 83/, has been proven adequate.

The analysis method requires that temporal parameters are preset; time bounds of intervals of transitions must be given values and cannot be considered as parameters. Some investigations on simple examples showed that it might be possible to handle temporal parameters, but no sufficiently general method has been devised yet.

The efficiency of the implemented TPN-analyzer allows handling quite large nets but, of course, a version of the package in compiled language would run much faster. This is considered; by the same opportunity, we will remove some limitations induced by the use of the APL for the prototype system. Editing a net will not require the (even light) knowledge of APL the current version requires.

The experiment also enlighted some known deficiencies of TINA; for instance the fact that entering user defined incrementally checked conditions require knowledge of some internal details of the analyzer, an improved version should include a full logical language for expressing such properties.

Despite this, TINA has been very helpful in analyzing the PROWAY protocol. Our future research will concentrate on performance evaluation; a promising approach is presented in /Roux 85/. It consists in extending the TPN model by using probabilities associated with the firing times of the transitions. A performance evaluation methodology is then proposed, based on the graph of state classes generated via the enumerative method; it contributes to fill the gap between functional and performance analysis.

TINA should be soon extended, in one hand, by a set of graph search functions and commands for finding constrained firing schedules in the graph of classes of a TPN and, in another hand, by an implementation of the above discussed performance analysis method for augmented TPNs.

REFERENCES

/Auger 81/ M. Auger, "Présentation de PROWAY", Doc. HN 231, EDF-GDF, Service Normalisation et Brevets, Nov. 1981.

/Ayache 82/ J. M. Ayache, J. P. Courtiat, M. Diaz, "REBUS, A Fault-Tolerant Distributed System for Industrial Real-Time Control", IEEE Tr. on Computers, Vol. C-31, N° 7, July 1982.

/Berthomieu 82/ B. Berthomieu, M. Menasche, "A State Enumeration Approach for Analyzing Time Petri Nets", 3rd European Workshop on Applications and Theory of Petri Nets, Varenna, Italy, Sept. 1982.

/Berthomieu 83/ B. Berthomieu, M. Menasche, "An Enumerative Approach for Analyzing Time Petri Nets", IFIP Congress 1983, Paris, Ed. North Holland, Sept. 1983.

/CEI Part.3 84/ CEI, Sous-Comité d'études 65C, "Process Data Highway (PROWAY) for distributed process control systems. Part 3: Specification for Highway Unit Protocol", Doc. 65(Central Office, March 1984.

/IEEE 802.4 84/ IEEE Project 802, Local Area Network Standards, "Token-Passing Bus Access Method", Document IEEE/802.4/84, Draft E, Approved Standard, 1984.

/Jones 77/ N. D. Jones, L. H. Landweber, Y. E. Lien, "Complexity of some problems in Petri Nets", Theoretical Computer Science 4, 1977.

/Karp 69/ R. M. Karp, R. E. Miller, "Parallel Program Schemata", Journal of Computer and System Sciences 3, 1969.

/Kryskow 81/ J. M. Kryskow, C. K. Miller, "Local Area Networks Overview-Part 1: Definitions and Attributes, Part 2: Standards Activities", Computer Design, Feb. and March 1981.

/Menasche 83/ M. Menasche, B. Berthomieu, "Time Petri Nets for Analyzing and Verifying Time Dependent Communication Protocols", 3rd IFIP/WG 6.1 International Workshop on Protocol Specification, Testing and Verification, Zurich, Switzerland, May 1983.

/Menasche 85/ M. Menasche, "PAREDE, An Automated Tool for the Analysis of Time(d) Petri Nets", 1st International Workshop on Timed Petri Nets, Torino, Italy, JUly 1985.

/Merlin 74/ M. Merlin, "A Study of the Recoverability of Computer Systems", Ph.D. Thesis, University of California, Irvine, 1974.

/Roux 85/ J. L. Roux, "Modelisation et Analyse des Systemes Distibues par les Reseaux de Petri Temporels", These de Docteur-Ingenieur, INSA, Toulouse, Decembre 1985.

/Voss84/ K. Voss, "A predicate/transition-net model of a local area network protocol", 5th European Workshop on Applications and Theory of Petri Nets, Aarhus Univ., Denmark, June 1984.

APPENDIX

```
      ∇ PROWAY1
[1]   ⍝ STATION 1
[2]    T1U1[0,0]: P1U1,P2U1 → P1U1
[3]    T2U1[260,300]: P1U1 → P3U1
[4]    T3U1[0,0]: P1U1,P4U1 → P3U1
[5]    T4U1[16,24]: P3U1 → P5U1,P6U1
[6]    T5U1[0,10]: P5U1 → P2U2,P2U3,P2U4
[7]    T6U1[0,0]: P7U1,P9U1 → P8U1
[8]    T7U1[0,0]: P7U1,P8U1 → P8U1
[9]    T3U1[0,0]: P7U1,P10U1 → P8U1
[10]   T9U1[50,100]: P6U1,P3U1 → P9U1,P11U1,P12U1
[11]   T10U1[16,24]: P3U1,P8U1 → P9U1,P11U1,P12U1
[12]   T11U1[0,10]: P11U1 → P4U2,P2U3,P2U4,P7U4
[13]   T12U1[50,100]: P6U1,P9U1 → P10U1,P11U1,P12U1
[14]   T13U1[16,24]: P3U1,P9U1 → P10U1,P11U1,P12U1
[15]   T14U1[0,10]: P13U1 → P2U2,P4U3,P2U4,P7U4
[16]   T15U1[50,100]: P6U1,P10U1 → P10U1,P13U1,P14U1
[17]   T16U1[16,24]: P3U1,P10U1 → P10U1,P13U1,P14U1
[18]   T17U1[50,53]: P12U1 → P13U1,P14U1
[19]   T18U1[0,0]: P2U1,P14U1 → P1U1
[20]   T19U1[0,0]: P2U1,P12U1 → P1U1
[21]   T20U1[0,0]: P2U1,P3U1 → P1U1
[22]   T21U1[0,0]: P2U1,P6U1 → P1U1
[23]   T22U1[0,0]: P4U1,P12U1 → P12U1
[24]  ⍝ INITIAL MARKING
[25]   M0=P6U1(1),P8U1(1)
      ∇
```

Description of the TPN model for station 1 of PROWAY.

```
      READNET 'PROWAY1'

PROWAY1 LOADED,
14 PLACES, 22 TRANSITIONS.

      READMERGENET 'PROWAY2'

PROWAY2 MERGED
28 PLACES, 44 TRANSITIONS.

      READMERGENET 'PROWAY3'

PROWAY3 MERGED
42 PLACES, 66 TRANSITIONS.

      READMERGENET 'PROWAY4'

PROWAY4 MERGED
56 PLACES, 88 TRANSITIONS.
```

Loading sequence of the global TPN model using functions "READNET" and "READMERGENET".

Local models for the other stations, PROWAY2, PROWAY3 and PROWAY4 respectively, are obtained from PROWAY1 by circular permutation of suffixes Uj (i.e. j = j+1 mod 4).

Table 1

Production of the global TPN model

```
OPTIONS

CPU TIME INCREMENT IS 10; ENTER NEW VALUE OR RETURN: 60

NUMBER OF CLASSES INCREMENT IS: 50; ENTER NEW VALUE OR RETURN: 100

BOUNDEDNESS TEST IS 2; ENTER NEW VALUE (1≤V≤4) OR RETURN:

SPECIFIC CONDITION IS PSAFE
TSAFE, PSAFE AND EMPTY ARE KNOWN CONDITIONS
ENTER NEW CONDITION OR RETURN:

BOUNDED

OVERFLOW NUMBER OF CLASSES
100 CLASSES ENUMERATED
CONTINUE (C) OR PAUSE (P) ?: P

ENUMERATION SUSPENDED, ENTER CONTINUE FOR MORE
100 CLASSES ENUMERATED SO FAR.
CPUT = 18.922S
```

Starting enumeration using function "BOUNDED" and option "PSAFE".

```
CLASSES '38*40'

CLASS C38:
M = P3U1, P2U3,
    P8U1, P1U2,
    P8U2, P1U3,
    P7U3, P9U3,
    P9U4, P12U4
D : 16 ≤ T4U1 ≤ 24
    16 ≤ T10U1 ≤ 24
    340 ≤ T2U2 ≤ 380
    0 ≤ T1U3 ≤ 0
    310 ≤ T2U3 ≤ 420
    0 ≤ T6U3 ≤ 0
    40 ≤ T17U4 ≤ 53
    T2U3 - T17U4 ≤
    370
    T17U4 - T2U3 ≤
    267

CLASS C39:
M = P3U1, P8U1,
    P1U2, P8U2,
    P1U3, P7U3,
    P9U3, P9U4,
    P12U4
D : 16 ≤ T4U1 ≤ 24
    16 ≤ T10U1 ≤ 24
    340 ≤ T2U2 ≤ 380
    420 ≤ T2U3 ≤ 460
    0 ≤ T6U3 ≤ 0
    40 ≤ T17U4 ≤ 53

CLASS C40:
M = P3U1, P8U1,
    P1U2, P8U2,
    P1U3, P8U3,
    P9U4, P12U4
D : 16 ≤ T4U1 ≤ 24
    16 ≤ T10U1 ≤ 24
    340 ≤ T2U2 ≤ 380
    420 ≤ T2U3 ≤ 460
    40 ≤ T17U4 ≤ 53

NODES '35*40'

C35 → (T11U4∈[0,10])/C36
C36 → (T3U1∈[0,0])/C37, (T1U2∈[0,0])/C84, (T1U3∈[0,0])/C88,
(T6U3∈[0,0])/C90
C37 → (T1U2∈[0,0])/C38, (T1U3∈[0,0])/C81, (T6U3∈[0,0])/C83
C38 → (T1U3∈[0,0])/C39, (T6U3∈[0,0])/C80
C39 → (T6U3∈[0,0])/C40
C40 → (T4U1∈[16,24])/C41, (T10U1∈[16,24])/C65

CONTINUE

BOUNDED
180 STATE CLASSES
CPUT = 44.134S
```

Continuation and termination of the analysis.

Parameters for enumeration are set via the function "OPTIONS"; Boundednes tests implement the sufficient conditions. Enumeration is suspended when the increment for the number of classes is reached. Then the partial graph can be examined using functions "CLASSES 'x*y'" and "NODES 'x*y'".

Table 2

Analysis of the gobal TPN model

```
      LIVE

NOT LIVE (USE LIVEDIAGNOSIS FOR MORE INFORMATION)
NOT CYCLIC

      LIVEDIAGNOSIS


LIVENESS DIAGNOSIS:

TRANSITIONS T1U1 T3U1 T4U1 T5U1 T6U1 T9U1 T10U1 T11U1 T19U1 T1U2 T3U2 T4
      U2 T5U2 T6U2 T9U2 T10U2 T11U2 T19U2 T1U3 T3U3 T4U3 T5U3 T6U3 T9U3
      T10U3 T11U3 T19U3 T1U4 T3U4 T4U4 T5U4 T6U4 T9U4 T10U4 T11U4 T19U4
      ARE LIVE

TRANSITIONS T2U1 T7U1 T8U1 T12U1 T13U1 T14U1 T15U1 T16U1 T17U1 T18U1 T20
      U1 T21U1 T22U1 T2U2 T7U2 T8U2 T12U2 T13U2 T14U2 T15U2 T16U2 T17U2
      T18U2 T20U2 T21U2 T22U2 T2U3 T7U3 T8U3 T12U3 T13U3 T14U3 T15U3 T16
      U3 T17U3 T18U3 T20U3 T21U3 T22U3 T2U4 T7U4 T8U4 T12U4 T13U4 T14U4
      T15U4 T16U4 T17U4 T18U4 T20U4 T21U4 T22U4.ARE DEAD FROM THE INITIA
      L CLASS
```

Function "LIVE" tests the liveness property. Diagnosis provided in our case shows that several transitions never fire. It comes from the fact that we are analyzing the error-free behavior of the protocol. As the TPN of figure 1 includes the recovery mechanisms but not the fault hypothesis, the corresponding transitions cannot fire.

The global net is not cyclic, because a transient initialization stage appears before the system enters its repetitive behavior. This is due to the fact that the firing intervals bounds may be interdependent in the permanent behavior, as opposed to the static values all transitions have received in the initial class.

Table 3
Characterization of the error-free behavior