Computational Analysis of a Power Distribution System with Petri Nets

Andrés Iglesias and Akemi Gálvez

Department of Applied Mathematics and Computational Sciences,
University of Cantabria, Avda. de los Castros,
s/n, E-39005, Santander, Spain
{iglesias,galveza}@unican.es
http://personales.unican.es/iglesias

Abstract. Petri nets provide a simple yet very intuitive graphical model for processes such as concurrency, parallelism and synchronization. Furthermore, they have a solid mathematical foundation and a number of analysis methods available. Due to these reasons, Petri nets are especially suited for the analysis of large and complex systems. In this paper we apply the Petri nets formalism to represent and analyze a power distribution system. To this aim, we report briefly a Mathematica package developed by the authors to deal with Petri nets. The package is subsequently applied to determine the possible scenarios of a failure of the system and some associated behaviors.

1 Introduction

The analysis of large and complex systems is still a challenge from the computational point of view. Many different approaches have been described during the last few years to tackle this issue. Among them, those based on Petri nets (PN) are gaining more and more popularity. Most of the PN interest lies on their ability to represent a number of events and states in a distributed, parallel, nondeterministic or stochastic system and to simulate accurately processes such as concurrency, sequentiality or asynchronous control [2, 3, 12]. In addition, they have a solid mathematical foundation and a number of analysis methods available: linear algebraic techniques to verify properties such as place invariants, transition invariants and reachability; graph analysis and state equations to analyze their dynamic behavior; simulation and Markov-chain analysis for performance evaluation, etc. As a consequence, Petri nets have been successfully applied to many interesting problems including finite-state machines, concurrent systems, multiprocessors, parallel and distributed computation, formal languages, communication protocols, software verification and validation and many others [4–6, 8, 11].

In this paper we apply the Petri nets formalism to analyze the behavior and all possible scenarios of a failure of a power distribution system. As it will be shown later on, the PN scheme allows us not only to characterize all situations leading to system failures but also to answer many related questions such as

T.-h. Kim et al. (Eds.): FGIT 2010, LNCS 6485, pp. 85–98, 2010.

Table 1. Mathematical definition of a Petri net

A Petri net (PN) is a 5-tuple $\mathcal{PN} = (\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{W}, \mathcal{M}_0)$ comprised of:

- a finite set of places, $\mathbf{P} = \{P_1, \ldots, P_m\}$,
- a finite set of transitions, $\mathbf{T} = \{t_1, \ldots, t_n\}$,
- a set of arcs, $\mathbf{A} \subseteq (\mathbf{P} \times \mathbf{T}) \bigcup (\mathbf{T} \times \mathbf{P})$
- a weight function: $\mathbf{W} : \mathbf{A} \to \mathbb{N}^q \quad (q = \#(\mathbf{A}))$
- an initial marking: $\mathcal{M}_0 : \mathbf{P} \to \mathbb{N}^m$

If $\mathcal{PN}$ is a finite capacity net, it also contains:

- a set of capacities, $\mathbf{C} : \mathbf{P} \to \mathbb{N}^m$
- a finite collection of markings $\mathcal{M}_i : \mathbf{P} \to \mathbb{N}^m$

the number of steps required to achieve a specific system state or the actions involved in such a process. All these questions are properly addressed by using a *Mathematica* package briefly reported in this paper.

The structure of this paper is as follows: firstly, some basic concepts and definitions about Petri nets (mainly intended for those unfamiliar with this kind of methodology) are given in Section 2. Section 3 describes briefly a *Mathematica* package to deal with some specific classes of Petri nets. The package is subsequently applied in Section 4 to the representation and failure analysis of the power distribution system. Some conclusions and further remarks close the paper.

2 Basic Concepts and Definitions

A *Petri net* (PN) is a special kind of directed graph, together with an initial state called the initial marking (see Table 1 for the mathematical details). The graph of a PN is a bipartite graph containing *places* $\{P_1, \ldots, P_m\}$ and *transitions* $\{t_1, \ldots, t_n\}$. Figure 1 shows an example of a Petri net comprised of three places and six transitions. In graphical representation, places are usually displayed as circles while transitions appear as rectangular boxes. The graph also contains arcs either from a place P_i to a transition t_j (*input arcs* for t_j) or from a transition to a place (*output arcs* for t_j). These arcs are labeled with their weights (positive integers), with the meaning that an arc of weight w can be understood as a set of w parallel arcs of unity weight (whose labels are usually omitted). In Fig. 1 the input arcs from P_1 to t_3 and P_2 to t_4 and the output arc from t_1 to P_1 have weight 2, the rest having unity weight.

A *marking* (state) assigns to each place P_i a nonnegative integer, k_i. In this case, we say that P_i *is marked with* k_i *tokens*. Graphically, this idea is represented by k_i small black circles (tokens) in place P_i. In other words, places hold tokens to represent predicates about the world state or internal state. The presence or absence of a token in a place can indicate whether a condition associated with

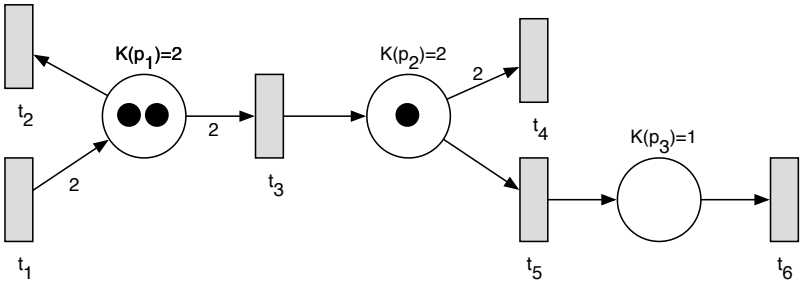

Fig. 1. Example of a Petri net

this place is true or false, for instance. For a place representing the availability of resources, the number of tokens in this place indicates the number of available resources. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. All markings are denoted by vectors $\mathcal{M}$ of length m (the total number of places in the net) such that the i-th component of $\mathcal{M}$ indicates the number of tokens in place P_i. From now on the initial marking will be denoted as $\mathcal{M}_0$. For instance, the initial marking (state) for the net in Figure 1 is $\{2, 1, 0\}$.

The pre- and post-sets of nodes are specified in this paper by a dot notation, where $\bullet u = \{v \in \mathbf{P} \bigcup \mathbf{T} / (v, u) \in \mathbf{A}\}$ is called the pre-set of u, and $u\bullet = \{v \in \mathbf{P} \bigcup \mathbf{T} / (u, v) \in \mathbf{A}\}$ is called the post-set of u. The pre-set of a place (transition) is the set of input transitions (places). The post-set of a place (transition) is the set of output transitions (places). The dynamical behavior of many systems can be expressed in terms of the system states of their Petri net. Such states are adequately described by the changes of markings of a PN according to a *firing rule* for the transitions: a transition t_j is said to be *enabled* in a marking $\mathcal{M}$ when all places in $\bullet t_j$ are marked. For instance, transitions t_2, t_3 and t_5 in Figure 1 are enabled, while transitions t_4 and t_6 are not. Note, for example, that transition t_4 has weight 2 while place P_2 has only 1 token, so arc from P_2 to t_4 is disabled. If transition t_j is enabled, it may or may not be fired (depending on whether or not the event represented by such a transition occurs). A firing of transition t_j removes $w_{i,j}$ tokens from each input place P_i of t_j and adds $w_{j,k}$ tokens to each output place P_k of t_j, $w_{j,k}$ being the weight of the arc from t_j to P_k. The fireability property of a transition t_j is denoted by $\mathcal{M}[t_j >$ while the creation of a new marking $\mathcal{M}'$ from $\mathcal{M}$ by firing t_j is denoted by $\mathcal{M}[t_j > \mathcal{M}'$.

A marking $\bar{\mathcal{M}}$ is *reachable* from any arbitrary marking $\mathcal{M}$ *iff* there exists a sequence of transitions $\sigma = t_1 t_2 t_3 \dots t_n$ such that $\mathcal{M}[t_1 > \mathcal{M}_1[t_2 > \mathcal{M}_2 \dots \mathcal{M}_{n-1}[t_n > \bar{\mathcal{M}}$. For short, we denote that the marking $\bar{\mathcal{M}}$ is reachable from $\mathcal{M}$ by $\mathcal{M}[\sigma > \bar{\mathcal{M}}$, where σ is called the *firing sequence*. The set of all markings reachable from $\mathcal{M}$ for a Petri net $\mathcal{PN}$ is denoted by $\biguplus[(\mathcal{PN}, \mathcal{M}) >$. Given a Petri net $\mathcal{PN}$, an initial marking $\mathcal{M}_0$ and any other marking $\mathcal{M}$, the problem of determining whether $\mathcal{M} \in \biguplus[(\mathcal{PN}, \mathcal{M}_0) >$ is known as the *reachability problem* for Petri nets. It has been shown that this problem is decidable [9] but it is

also *EXP-time* and *EXP-space hard* in the general case [10]. In many practical applications it is interesting to know not only if a marking is reachable, but also what are the corresponding firing sequences leading to this marking. This can be done by using the so-called *reachability graph*, a graph consisting of the set of nodes of the original Petri net and a set of arcs connecting markings $\mathcal{M}_i$ and $\mathcal{M}_j$ *iff* $\exists t \in \mathbf{T}/\mathcal{M}_i[t > \mathcal{M}_j$.

A transition without any input place is called a *source transition*. Note that source transitions are always enabled. In Figure 1 there is only one source transition, namely t_1. A transition without any output place is called a *sink transition*. The reader will notice that the firing of a sink transition removes tokens but does not generate new tokens in the net. Sink transitions in Figure 1 are t_2, t_4 and t_6. A couple (P_i, t_j) is said to be a self-loop if $\mathbf{P}_i \in (\bullet t_j \bigcap t_j \bullet)$ (i.e., if P_i is both an input and an output place for transition t_j). A Petri net free of self-loops is called a *pure* net. In this paper, we will restrict exclusively to pure nets.

Some PN do not put any restriction on the number of tokens each place can hold. Such nets are usually referred to as *unfinite capacity net*. However, in most practical cases it is more reasonable to consider an upper limit to the number of tokens for a given place. That number is called the *capacity* of the place. If all places of a net have finite capacity, the net itself is referred to as a *finite capacity net*. All nets in this paper will belong to this later category. In such case, there is another condition to be fulfilled for any transition t_j to be enabled: the number of tokens at each output place of t_j must not exceed its capacity after firing t_j. For instance, transition t_1 in Figure 1 is initially disabled because place P_1 has already two tokens. If transitions t_2 and/or t_3 are applied more than once, the two tokens of place P_1 will be removed, so t_1 becomes enabled. Note also that transition t_3 cannot be fired initially more than once, as capacity of P_2 is 2.

3 A *Mathematica* Package for Petri Nets

In this section a *Mathematica* package to deal with Petri nets is briefly reported (the reader is referred to [7] for a more detailed information about this software). We start our discussion by loading the package:

```
In[1]:= <<PetriNets‘
```

According to Table 1, a Petri net (like that in Figure 1 and denoted onwards as `net1`) is described as a collection of lists. In our representation, `net1` consists of three elements: a list of couples $\{place, capacity\}$, a list of transitions and a list of arcs from places to transitions along with its weights:

```
In[2]:= net1={{{p1,2},{p2,2},{p3,1}},{t1,t2,t3,t4,t5,t6},{{p1,t1,2},
    {p1,t2,-1},{p1,t3,-2},{p2,t3,1},{p2,t4,-2},{p2,t5,-1},{p3,t5,1},
    {p3,t6,-1}}};
```

Note that the arcs are represented by triplets $\{place, transition, weight\}$, where positive value for the weights mean output arcs and negative values denote input arcs. This notation is consistent with the fact that output arcs add tokens to the places while input arcs remove them. Now, given the initial marking $\{2, 1, 0\}$ and any transition, the `FireTransition` command returns the new marking obtained by firing such a transition:

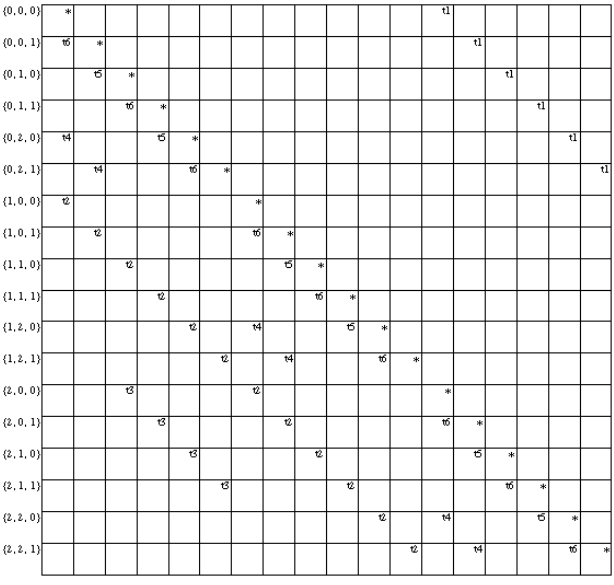

Fig. 2. Reachability graph for the Petri net in Figure 1

```
In[3]:= FireTransition[net1,{2,1,0},t2];
```
Out[3]:= {1,1,0}

Given a net and its initial marking, an interesting question is to determine whether or not a transition can be fired. The `EnabledTransitions` command returns the list of all enabled transitions for the given input:

```
In[4]:= EnabledTransitions[net1,{2,1,0}];
```
Out[4]:= {t2,t3,t5}

The `FireTransition` command allows us to compute the resulting markings obtained by applying these transitions onto the initial marking:

```
In[5]:=FireTransition[net1,{2,1,0},#]& /@ %;
```
Out[5]:= {{1,1,0},{0,2,0},{2,0,1}}

Note that, since transition $t1$ cannot be fired, an error message is returned:

```
In[6]:= FireTransition[net1,{2,1,0},t1];
```
Out[6]:= FireTransition: Disabled transition: t1 cannot be fired for the given net and the {2,1,0} marking.

From *Out[4]* and *Out[5]*, the reader can easily realize that successive applications of the `EnabledTransitions` and `FireTransition` commands allows us to obtain all possible markings and all possible firings at each marking. However, this is a tedious and time-consuming task to be done by hand. Usually, such markings and firings are graphically displayed in the reachability graph (see description above). The next input returns the reachability graph for our Petri net and its initial marking[1]:

[1] For an arbitrary PN the reachability graph may be of infinite size. This is not the case in this paper, as we restrict ourselves to finite PN.

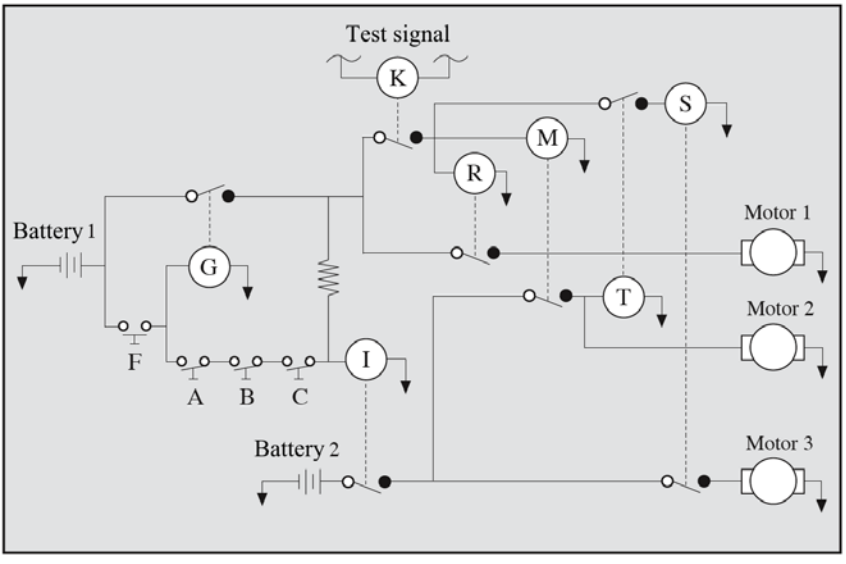

Fig. 3. Power distribution system diagram

```
In[7]:= ReachabilityGraph[net1,{2,1,0}];
```
Out[7]:= See Figure 2

Figure 2 can be interpreted as follows: the outer column on the left provides the list of all possible markings for the net. Their components are sorted in increasing order from the top to the bottom, according to the standard lexicographic order. For any marking, the row in front gives the collection of its enabled transitions. For instance, the enabled transitions for the initial marking $\{2, 1, 0\}$ are $\{t2, t3, t5\}$ (as expected from *Out[4]*), while they are $\{t1, t4, t6\}$ for $\{0, 2, 1\}$. Given a marking and one of its enabled transitions, we can determine the output marking of firing such transition by simply moving up/down in the transition column until reaching the star symbol: the marking in that row is the desired output. By this simple procedure, results such as *Out[5]* can readily be obtained.

4 Applying Petri Nets to a Power Distribution System

Petri nets have been successfully used as a formal method for the representation and analysis of large and complex dynamical systems during the last two decades. The reason is the large amount of mathematical tools available to analyse normal Petri nets [1, 12]. This allows us to perform a formal check of the properties related to the behavior of the underlying system, e.g., precedence relations amongst events, concurrent operations, appropriate synchronization, freedom from deadlock, repetitive activities, and mutual exclusion of shared

Table 2. Definition of variables

Q	EMF applied to Motor 2 for $t > 60$ seconds
N	M relay contacts remain closed for $t > 60$ seconds
J	I relay contacts fail to open when M contacts have been closed to $t > 60$ seconds
L	EMF remains on M coil for $t > 60$ seconds
H	G relay contacts fail to open when M contacts have been closed for $t > 60$ seconds
E	G relay contacts fail to open when K contacts have been closed for $t > 60$ seconds
D	EMF not removed from G relay coil when K contacts have been closed for $t > 60$ seconds
A, B, C	Primary failure of timer A, B, C (resp.)
F	Primary failure of pushbutton F
G, I, M	Primary failure of relays G, I, M (resp.)
K	Primary failure of test signal K

resources, to mention just a few. The ability of PN to verify the model formally is especially important for realtime safety-critical systems, such as air-traffic control systems, rail-traffic control systems, nuclear reactor control systems and so on. In this section, we apply Petri nets' formalism to failure detection of the system described in the next paragraphs.

4.1 Description of the System

Figure 3 shows a power distribution system with three motors, 1, 2 and 3, and three timers, A, B and C, which are normally closed. A momentary depression of pushbutton F applies power from a battery to the coils of cutthroat relays G and I. Thereupon G and I close and remain electrically latched. To check whether the three motors are operating properly, a 60-second test signal is impressed through K. Once K has been closed, power from battery 1 is applied to the coils of relays R and M. The closure of R starts motor 1. The closure of T applies power from battery 1 to coil S. The closure of S starts motor 3.

After an interval of 60 seconds, K is supposed to open, shutting down the operation of all three motors. Should K fail to be closed after the expiration of 60 seconds, all three timers A, B and C open, de-energizing the coil of G, thus shutting down the system. Suppose K open to de-energize G, and motor 1 stops. B and C act similarly to stop motor 2 or motor 3 should either M or S fail to be closed. For the sake of simplicity and brevity, in the following only the effect of motor 2 is analyzed.

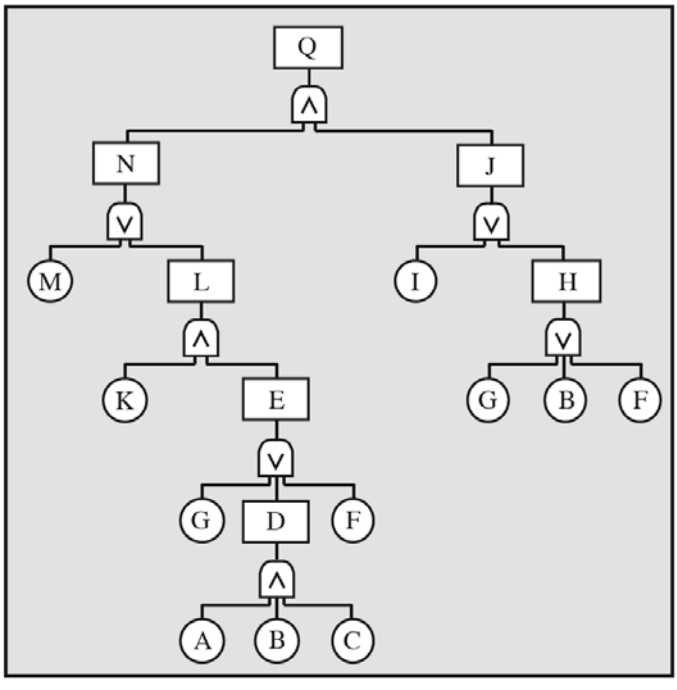

Fig. 4. Failure tree of the system

4.2 Failure Tree

To do so, let us consider Q to represent the event of failure of motor 2. We use the notation $Q = q$ to denote the failure of motor 2. Similarly, $a, b, c, \ldots$ represent the failures of the respective components $A, B, C, \ldots$. Based on the previous description of the system, we can derive the following logical expression for the failure of motor 2:

$$q = [m \vee (k \wedge g) \vee (k \wedge (a \wedge b \wedge c)) \vee (k \wedge f)] \wedge (i \vee g \vee b \vee f)$$

where the symbols $\vee$ and $\wedge$ denote the logical operators *or* and *and*, respectively. This expression is obtained by combining all possible partial failures of the components of the system that eventually produce a failure of motor 2. This equation can be expressed in a more intuitive way using the so-called failure tree. In this representation, the failures of timers A, B and C are combined into an intermediate cause of failure, D; then D is combined with G and F to define another intermediate cause of failure, E and so on. The resulting tree, depicted in Figure 4, includes the initial variables $\{A, B, C, F, G, I, K, M\}$ along with the intermediate failures $\{D, E, H, J, L, N\}$ that imply the failure of the motor. The final set of variables used in this example is $\{A, B, C, D, E, F, G, H, I, J, K, L, M, N, Q\}$. Their meaning is given in Table 2.

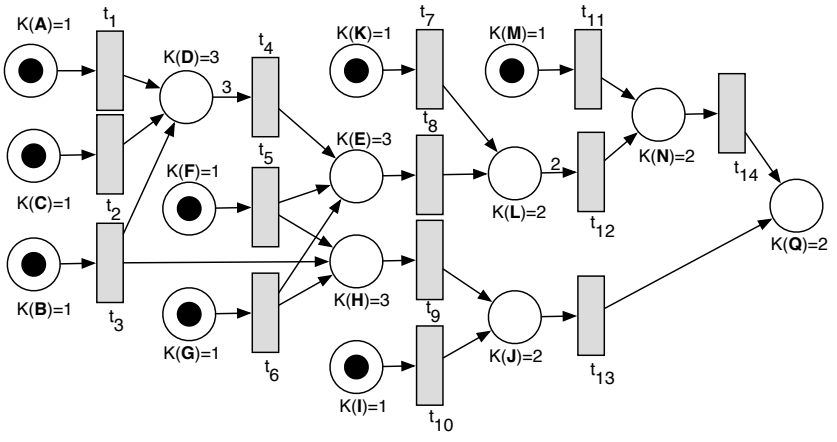

Fig. 5. Petri net of the power distribution system

4.3 Petri Net Representation

This system can be represented in *Mathematica* as:

```
In[8]:=motor={{{A,1},{C,1},{B,1},{D,3},{F,1},{G,1},{K,1},{E,3},{H,3},{I,1},
{M,1},{L,2},{J,2},{N,2},{Q,2}},{t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,t13,
t14},{{t1,A,-1},{t1,D,1},{t2,C,-1},{t2,D,1},{t3,B,-1},{t3,D,1},{t3,H,1},
{t4,D,-3},{t4,E,1},{t5,F,-1},{t5,E,1},{t5,H,1},{t6,G,-1},{t6,E,1},{t6,H,1},
{t7,K,-1},{t7,L,1},{t8,E,-1},{t8,L,1},{t9,H,-1},{t9,J,1},{t10,I,-1},
{t10,J,1},{t11,M,-1},{t11,N,1},{t12,L,-2},{t12,N,1},{t13,J,-1},{t13,Q,1},
{t14,N,-1},{t14,Q,1}}};
```

leading to the Petri net displayed in Figure 5.

For illustrative purposes, the net is represented along with an initial marking represented by tokens in some places. In particular, the net in Figure 5 does represent the situation where all system components fail simultaneously (thus leading to a general failure of the system), given by the marking:

```
In[9]:=imk={1,1,1,0,1,1,1,0,0,1,1,0,0,0,0};
```

4.4 System Analysis

Identification of primary variables. From Fig. 4, it seems that the failure of the motor, Q, is a consequence of the failures of $\{A, B, C, D, E, F, G, H, I, J, K, L, M, N, Q\}$. However, some of these variables are intermediate variables, meaning that their failures are consequence of other primary variables (associated with the real components of the system). According to the description of the problem in terms of Petri nets, this means that Q is reached iff:

$(t_{14} \vee t_{11} \vee t_{12} \vee (t_7 \wedge (t_8 \vee t_6 \vee t_5 \vee t_4 \vee (t_1 \wedge t_2 \wedge t_3)))) \wedge (t_{13} \vee t_{10} \vee t_9 \vee (t_6 \vee t_3 \vee t_5))$

Note however that t_{14} is initially disabled because it requires either t_{11} or t_{12} to be fired. Note also that similar arguments can be applied to t_{13}, t_9 and t_8. At its turn, t_{12} is initially disabled because it requires both t_8 and t_7 to

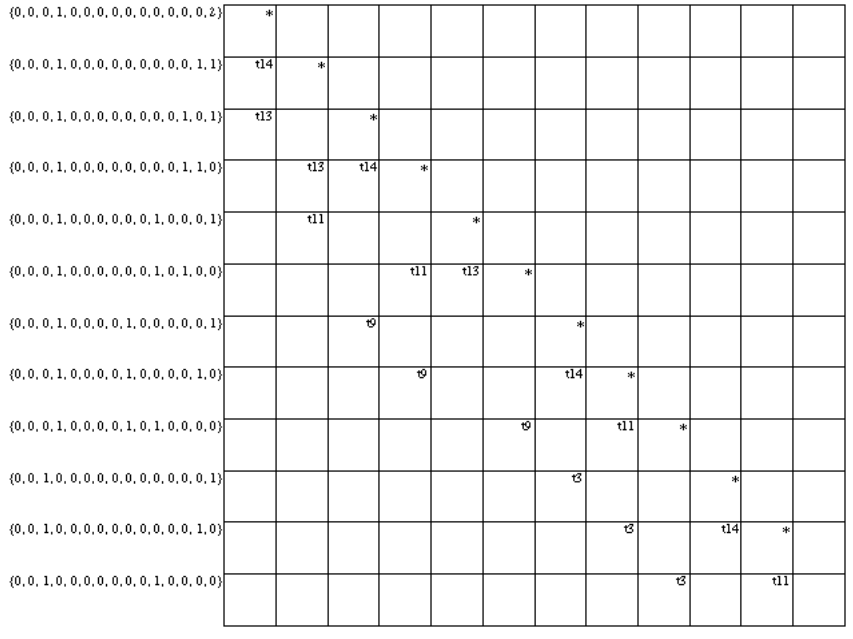

Fig. 6. Reachability graph of the power distribution system for failures of components B and M

be previously fired and the same applies to t_4. So, previous expression can be simplified to:
$(t_{11} \vee (t_7 \wedge (t_6 \vee t_5 \vee (t_1 \wedge t_2 \wedge t_3)))) \wedge (t_{10} \vee (t_6 \vee t_3 \vee t_5))$
that identifies the primary transitions of the system. This process can more easily be done by using the `EnabledTransitions` command:

```
In[10]:=EnabledTransitions[motor,imk]
```

Out[10]:= {*t1,t2,t3,t5,t6,t7,t10,t11*}

Reachable states of the system. This input returns the list of reachable markings (states) for the initial marking in Fig. 5:

```
In[11]:=lm=ListReachableMarkings[motor,%];
```

The output (stored in variable `lm`) is too large to be displayed here, as it includes 7074 possible states[2]:

```
In[12]:=Length[lm]
```

Out[12]:= 7074

However, the situation in which all system components fail at the same time is very rare. A more usual scenario is the failure of a specific component and then the discussion of the resulting cases simplifies greatly, as it reduces to a change

[2] Note that temporary states are also included in our analysis. Although they are associated with transient states of the system, they are often helpful in order to determine the sequences leading to system steady states.

in the initial marking, leading to different token locations (and hence, different states for the system) while the net topology no longer changes. In the following we will use several initial markings to account for failures of different system components. For instance, the failure of G and M yield 21 different states in the system:

```
In[13]:=fgm=ListReachableMarkings[motor,{0,0,0,0,0,1,0,0,0,0,1,0,0,0,0}];
Length[%]
```

Out[13]:= 21

of which only two lead to a failure of motor 2, represented by the condition $Q = 2$ (meaning that Q has two tokens):

```
In[14]:=Select[fgm,MatchQ[#,{___,2}]&]
```

Out[14]:= {{0,0,0,0,0,0,0,0,0,0,0,1,0,0,2},{0,0,0,0,0,0,0,1,0,0,0,0,0,0,2}}

while that the additional failure of B and M leads to the following reachability graph:

```
In[15]:=ReachabilityGraph[motor,{0,0,1,0,0,0,0,0,0,0,1,0,0,0,0}];
```

Out[15]:= See Figure 6

Analyzing transition sequences. One of the most interesting features of our approach is that it allows us to compute sequences of transitions leading to different states. For instance, we can determine the minimum number of transitions required for a given state to be reached. Because the Petri net is a planar graph, we implemented a modification of the minimal spanning tree algorithm. The command `MinimalTransitionSequence` admits three arguments: the Petri net, the initial marking and the target state and returns the list of transitions of minimal length yielding such target:

```
In[16]:=MinTransitionSequence[motor,imk,Q->2]
```

Out[16]:={t10,t11}

The obtained output means that there is only one possibility for the failure of motor 2 involving the minimum number of transitions. Note that t_{12} is not included in this solution, as it requires the prior firing of both t_7 and t_8, thus leading to larger sequences of transitions. Similarly, t_9 is not part of the solution to this problem because it requires the firing of either t_3, t_5 or t_6. This output means that the scenario of a system failure involving the minimum number of processes is that of a primary failure of relay M and the simultaneous occurrence of event J.

The target state in the previous command can be either a single condition (such as $Q = 2$) or a collection of multiple conditions involving Boolean operators. They are given in the form $lhs \to rhs$, according to the standard programming syntax in *Mathematica* for replacement rules. Thus:

```
In[17]:=MinimalTransitionSequence[motor,{0,0,1,0,0,0,0,0,0,0,1,0,0,0,0},
{I->0,Q->2}]
```

Out[17]:={{t3,t9,t11,t13,t14},{t3,t9,t11,t14,t13},{t3,t9,t13,t11,t14},
{t3,t11,t9,t13,t14},{t3,t11,t9,t14,t13},{t3,t11,t14,t9,t13},{t11,t3,t9,t13,t14},
{t11,t3,t9,t14,t13},{t11,t3,t14,t9,t13},{t11,t14,t3,t9,t13}}

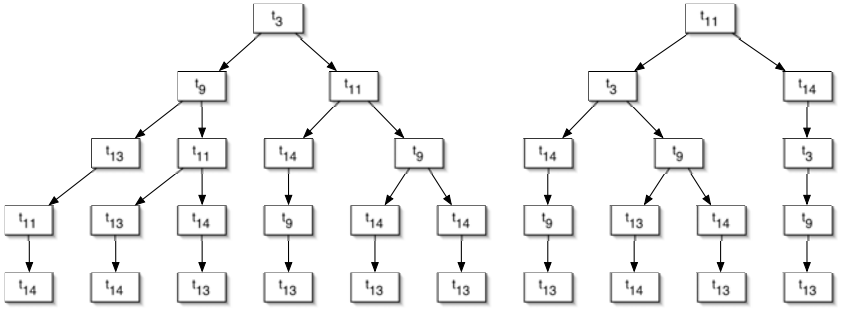

Fig. 7. Hierarchical tree of the sequences in *Out[17]*

returns the shortest sequences from the given marking that produces the failure of Q while the relay I is working properly. Such states are related to the failures of the following components:

```
In[18]:=PNComponents[motor,{0,0,1,0,0,0,0,0,0,0,1,0,0,0,0}#]& /@ %
```

Out[18]:={{*H,I*},{*E,I*},{*G,I*},{*F,I*}}

Note that the initial marking already states that B and M fail simultaneously. The following input returns the same result graphically:

```
In[19]:=HierarchicalTree[%,Sort->Descending,ChildOrdering->LeftRight,
LayerOffset->Automatic,BoundingBox->Rectangle,TextAlign->{Middle,
Center}]
```

Out[19]:=See Figure 7

The information stored in the hierarchical tree of Figure 7 can also be expressed in terms of the processes executed during the runtime. The input:

```
In[20]:=DirectedGraph[%%,Field->Linear,BoundingBox->Oval,
Configuration->Polygonal,TextAlign->{Middle,Center},LocalStates->True]
```

Out[20]:=See Figure 8

returns the directed graph of the processes being executed according to the sequences in Figure 7. In this graph, failures of intermediate variables are denoted as **LF** (for local failures) and that the value $Q = 2$ yields the failure of the system (motor 2 in this example).

The interpretation of this directed graph is as follows: each node of the graph corresponds to a specific process denoted by a variable according to Table 2. The arrows indicate the flow of execution of such processes, as depicted in Figure 7. Note that some arrows are bidirectional, as some processes can be executed in reverse order as shown in Figure 7. By this simple procedure, all sequences described in the two hierarchical trees of Figure 7 can be embedded into the single graph of Figure 8. Note also that some other processes cannot be executed in reverse order, as they rely on previous steps. For instance, execution of transition t_9 does require the prior execution of transition t_3; this means that the processes associated with these transitions, namely, $H = 1$ and $J = 1$ respectively, must always be executed in that order and hence, reverse order is never allowed. This fact is represented by the directed arrow from $H = 1$ to $J = 1$. Similarly, the process $Q = 2$ (general failure of the system) always require the prior process

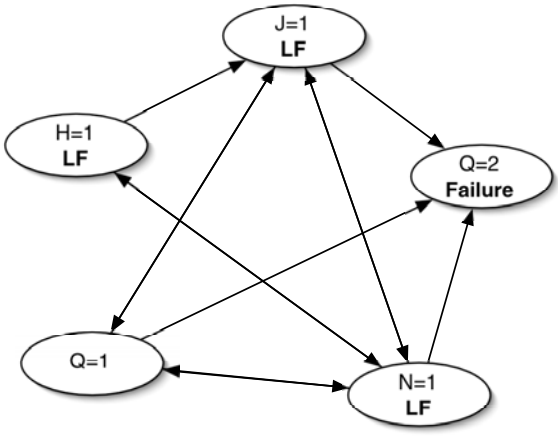

Fig. 8. Directed graph of the hierarchical tree of Figure 7

$Q = 1$ (failure of either N or J, the ultimate system components of Q) and thus, the arrow connecting both processes is undirectional. Therefore, this directed graph allows us to detect this kind of directional behavior at a glance. This feature is especially valuable for large systems, whose behavior is hard to be explained without the assistance of specialized tools.

5 Conclusions and Further Remarks

In this paper we have applied the Petri nets formalism to analyze the behavior of a power distribution system and obtain all possible situations of a failure of the system. To this aim, a *Mathematica* package developed by the authors has been successfully used. Our approach is very general in the sense that it can also be applied to the analysis of a bulk of larger and more complex examples not mentioned here because of limitations of space.

Regarding the implementation, it has been done in the computer algebra system *Mathematica* version 5.2. This software is available for all major operating systems (Windows, Macintosh, UNIX, Linux, etc.) on a variety of platforms and hardware configurations. All computations in this paper have been done on a Pentium IV, 3 GHz. with 1 GB. of RAM. Minimal requirements in terms of memory and storage capacity for our package do not exceed those commonly specified to install *Mathematica* thus facilitating the use of our package. Furthermore, all executions in our trials have been obtained in fractions of a second, making the package especially well suited for real-time applications and web services.

Future work includes the extension of our scheme to probabilistic systems and the inclusion of the system state equations in our analysis. In particular, the symbolic manipulation of the system state equations might lead to potential advantages for some problems such as the reachability, liveness and others.

This research has been supported by the Spanish Ministry of Education and Science, National Program in Computer Science (Project #TIN2006-13615) and the University of Cantabria.

References

1. Bourdeaud'huy, T., Hanafi, S., Yim, P.: Mathematical programming approach to the Petri nets reachability problem. European Journal of Operational Research 177, 176–197 (2007)
2. Gálvez, A., Iglesias, A., Corcuera, P.: Representation and Analysis of a Dynamical System with Petri Nets. In: Proc. International Conference on Convergence Information Technology-ICCIT 2007, Gyeongju, Korea, pp. 2009–2015. IEEE Computer Society Press, Los Alamitos (2007)
3. German, R.: Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets. John Wiley and Sons, Inc., New York (2000)
4. Iglesias, A.: A New Framework for Intelligent Semantic Web Services Based on GAIVAs. International Journal of Information Technology and Web Engineering, IJITWE 3(4), 30–58 (2008)
5. Iglesias, A.: Software Verification and Validation of Graphical Web Services in Digital 3D Worlds. In: Communications in Computer and Information Science. CCIS, vol. 56, pp. 293–300 (2009)
6. Iglesias, A.: Pure Petri Nets for Software Verification and Validation of Semantic Web Services in Graphical Worlds. International Journal of Future Generation Communication and Networking 3(1), 33–46 (2010)
7. Iglesias, A., Kapcak, S.: Symbolic computation of Petri nets. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4488, pp. 235–242. Springer, Heidelberg (2007)
8. Iglesias, A., Luengo, F.: New Goal Selection Scheme for Behavioral Animation of Intelligent Virtual Agents. IEICE Transactions on Information and Systems, Special Issue on CyberWorlds E88-D(5), 865–871 (2005)
9. Kosaraju, S.R.: Decidability of reachability in vector addition systems. In: Proc. 14th Annual ACM Symp. Theory Computing, pp. 267–281 (1982)
10. Lipton, R.: The reachability problem requires exponential space.Technical report, Computer Science Department, Yale University (1976)
11. Luengo, F., Iglesias, A.: Designing an Action Selection Engine for Behavioral Animation of Intelligent Virtual Agents. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3482, pp. 1157–1166. Springer, Heidelberg (2005)
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)