2011 American Control Conference
on O'Farrell Street, San Francisco, CA, USA
June 29 - July 01, 2011

Necessary Condition for a Petri Net Model That Incorporates Resources to Produce an Event Stream From an Unknown Initial State

L. V. Allen & D. M. Tilbury

Abstract— **This paper presents conditions under which a type of Petri net model with resources, called System of Transition Processes with Resources ($STPR$), can produce a given event stream. In particular, a necessary condition is given for the event stream to be produced based on checking upper and lower bounds of the marking on the Petri net. Algorithms for checking the condition are presented; the computational complexity of checking the condition is polynomial in the number of places and transitions the first time it is checked for a particular model, and linear in the number of transitions when the model is checked for subsequent event streams. An example is presented showing how this condition can be used, together with a model generation approach, for anomaly detection in manufacturing systems.**

I. Motivation and Problem Description

This work was motivated by a problem in anomaly detection for manufacturing systems [1]. Given a discrete-event (Petri net) model of a manufacturing system, and an event stream observed from that system, determine whether the model could have generated the stream. If the stream is inconsistent with the model, then it is called *anomalous* and may indicate the occurrence of a fault in the system. If the observed event stream and the model both start in the same (initial) state, the problem is straightforward to solve. However, many manufacturing systems run continuously, and the state from which the observed event stream begins may not be easily determined. Thus, a method to determine whether a Petri net model *could have* generated an event stream, starting from an unknown initial state, was needed. This paper presents such a method for a specific type of Petri net model that incorporates resources, called System of Transition Processes with Resources ($STPR$).

Section II describes relevant existing work, including the $STPR$ formalism, how the issue of determining whether a model could produce an event stream has been handled before, and estimating the marking of a Petri net. The issue of checking whether a model could produce an event stream whose initial state is unknown is addressed in Section III through developing theory and corresponding algorithms for a necessary condition. The algorithms are illustrated using a simple example manufacturing system in Section IV. Section V discusses how the necessary condition enables the application of an anomaly detection solution to a real manufacturing system. Finally, conclusions and future work from this project are summarized in Section VI.

This work was supported in part by the National Science Foundation grants EEC 95-92125, and CMS 05-28287.

L. V. Allen is currently with Creare Inc., Hanover NH. At the time of the research, L.V. Allen was with, and D. M. Tilbury is currently with, the University of Michigan, Departments of Electrical Engineering: Systems and Mechanical Engineering, Ann Arbor, MI 48109-2125. Corresponding author: lzallen@umich.edu

II. Existing Work

A. Petri Net Structure

Definition 1 (Petri net, Marked Petri net [12]): A *Petri net* N is a graph (P,T,F), where P is a finite set of places, T is a finite set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs from places to transitions and from transitions to places. A *marked Petri net* $N = (P,T,F,M)$ is a Petri net (P,T,F) with M as a marking of the set of places P that represents the Petri net's state. The *initial marking* M_0 of a Petri net corresponds to the initial state.

The marking M is the number of tokens in each place. $M(p_i)$ refers to the number of tokens in place p_i. The notation $\bullet p$ refers to all transitions t that put a token in p when they fire ($(t,p) \in F$), whereas $p\bullet$ are the transitions that remove a token from p to fire ($(p,t) \in F$), and similarly for $\bullet t$ and $t\bullet$. A transition t is enabled when $\forall p \in P$ such that $(p,t) \in F$, $M(p) \geq 1$. When transition t fires, the Petri net has a new marking M' based on removing tokens from the places that feed t ($\bullet t$) and adding tokens to the places that receive from t ($t\bullet$). The incidence matrix, A, of a Petri net is a representation of the set of arcs connecting places and transitions and vice versa. The ith column of A is $t_i\bullet - \bullet t_i$. Only ordinary Petri nets are considered here – those whose arcs all have weight one. More information about Petri nets can be found in Chapter 4 of [3].

System of Transition Processes with Resources (or $STPR$) [2] is a type of Petri net that incorporates resources and is modularly constructed. One of the motivations behind $STPR$s was the ability to modularly model highly parallel systems. In one example industry system studied, there were two gantries and six CNCs that could collectively handle up to eight parts at a time. Even assuming that each of these eight machines only has three states (a conservative estimate), that is still more than $6,000$ (3^8) possible states, but could be modeled by a relatively small $STPR$.

Significant detail about $STPR$s, including formal definitions, is provided in [2] and only the aspects of $STPR$s relevant to this research will be highlighted here. An $STPR$ consists of a set of processes that interact through shared resources, where resources include machines that change a part (i.e. CNC), machines that transport a part (i.e. robot), and components that transport a part (i.e. pallet). Each resource is represented by a place and the place is marked if and only if the resource is available. There are different types of resources – type A resources are always available

978-1-4577-0079-8/11/$26.00 ©2011 AACC

when not in use (robot, CNC, etc.), type C resources are created and consumed (empty pallets), and type NA and NC resources are negative versions of the type A and type C resources that represent when the resource is unavailable. The places for type A and type NC resources are initially marked, whereas those for type C and type NA resources are initially unmarked. Events in an $STPR$ can acquire and release resources, where acquiring a resource is taking a token from the place associated with that resource and releasing a resource is putting a token in said place. Resources are the only places that may initially be marked. Events and transitions have a one-to-one mapping in $STPR$s.

An $STPR$ can have mutual exclusion decisions, i.e. one place leads to two transitions, and hence when there is a token in that place, a decision must be made about which transition to fire. Additionally, an $STPR$ can exhibit concurrency, both with multiple processes running at the same time and multiple instances of a single process running at the same time. Each type A resource in an $STPR$ is conserved, a property that is guaranteed by requiring that each type A resource be associated with a p-semiflow [12], which is a vector $y \geq 0$ such that $A^T y = 0$ and $y(i) > 0$ for at least one element of y.

An example $STPR$ that models a small manufacturing cell is illustrated in Figure 1. This $STPR$ has three resource places: R1, R2, and NotR2, corresponding to the availability of resources R1 and R2 and unavailability of resource R2, respectively. It has two processes, one for the events ending in '1' and one for the events ending in '2', and these two processes interact through the shared resource R1. As an example of how the events interact with resources, R1 is acquired by $g1$ and $g2$ and released by $q1$, $m1$, and $q2$.

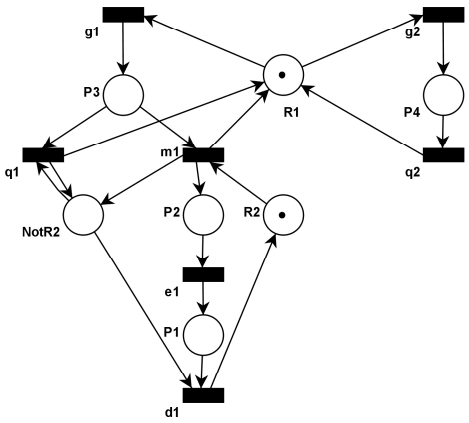

Fig. 1. $STPR$ Model of Small Manufacturing Cell

B. Event Stream From Unknown Initial State

It is common in academic work to assume that all observed event streams start from the system's initial state. For example, in the area of discrete event systems (DES), system identification often assumes that the system's behavior is cyclic and that the observed event streams start and end in the initial state [8], [13], [11]. In workflow mining, an area of computer science that also involves system identification of some forms of DES, a similar assumption is made [15] [5]. As discussed in Section I, however, many industry manufacturing systems run continuously and thus observed event streams may not start from the system's initial state.

Several existing approaches were considered to determine whether an $STPR$ could have generated an event stream, starting from an unknown initial state. The set of reachable states could be calculated [10] and then searched for sequences of valid markings that would allow the event stream to occur. If at least one such sequence of markings existed, then the model would accept that event stream, but if no such sequence existed, then the model would not accept the event stream. A major concern about this approach is the computational complexity of calculating the set of reachable states, which in general takes exponential time [12], searching through it, and keeping track of the different possible marking sequences thus far.

Another approach considered was searching the reachable markings for a sequence that allows the event stream, but calculating the set of reachable markings in a simpler way. For some classes of Petri nets, the set of reachable markings can be described concisely without reference to such a graph [12]. However, $STPR$s do not, in general, belong to these classes and thus their results do not apply.

Because the state from which an observed event stream starts is unknown, and the set of reachable states cannot be easily described for $STPR$s, state (marking) estimation is necessary. Some work in marking estimation, such as [4], assumes the initial marking is known and estimates the current marking, where the uncertainty is due to silent transitions. Other research estimates the initial marking through observation of an event stream. In [6], a lower bound on the initial marking of a Petri net is determined through observing a stream of events, where the lower bound is updated after each observed event. In [9], the minimum initial marking is calculated for the case in which transition labels are observed and the uncertainty comes from the labels and transitions not having a one-to-one marking. In both of these approaches, only a lower bound on the initial marking is determined, not a lower bound on each of the markings associated with the event stream occurring, and hence these approaches cannot be used directly. The idea of a lower bound on the initial marking is used in developing the theory in Section III, although extended to a lower bound on all the markings for producing an event stream.

III. Initial State Issue and Resolution

The problem, as described in Section I, is: given an event stream starting from an unknown initial state and a Petri net ($STPR$) model, determine whether there exists a sequence of states (markings) that could reproduce this event stream.

A. Theoretical Foundation for Necessary Condition

The theory determines a sequence of lower bounds on the markings necessary for the given model to have produced the given event stream and checks whether these lower bound

markings violate the upper bound marking restrictions caused by the conservation of type A resources. If the lower bound of the markings does not exceed the upper bound marking restrictions, then the model may be able to produce the event stream, whereas if the any of these lower bounds exceeds the upper bound, the model cannot produce the event stream. The first event for which the upper bound is violated is returned by the algorithm. This theory assumes that the model's initial marking is known (all type A and type NC resources are initially marked, and all other places are initially unmarked), but that the event stream's initial state does not necessarily correspond to to the model's initial marking.

Given an event stream and a model, a set of lower bound markings can be calculated for that model to have produced that event stream. The theory builds up from the simplest case – a single event.

Lemma 1: Given a Petri net and an event e that can occur in that Petri net, the lower bound on the model's marking prior to the event e occurring is $M_{LB,0} = \bullet e$ and the lower bound after the event e has occurred is $M_{LB,1} = e\bullet$.

A slight abuse of notation is made to describe $\bullet e$ as not only the places that feed tokens into event e, but also as the marking of those places. This lemma is evident based on the definition of a Petri net, Definition 1. For an event e to occur in a Petri net, it must be that $M_0 \geq \bullet e$ so that e is enabled. Likewise, when an event e occurs it will necessarily produce $e\bullet$, and thus $M_1 \geq e\bullet$.

Lemma 2: Given a Petri net, a stream of events $\sigma = e_1...e_m$ where $e_i \ \forall i = 1...m$ can occur in the Petri net, and a lower bound on the markings $M_{LB,0}...M_{LB,m-1}$ based on events $e_1...e_{m-1}$, the lower bound on the model's markings when e_m is included are $M'_{LB,0}...M'_{LB,m-1}M'_{LB,m}$, where

- $M'_{LB,i} = M_{LB,i} + NE_m$ for $i = 1...m-1$
- $NE_m = max((\bullet e_m - M_{LB,m-1}), 0)$
- $M'_{LB,m} = M'_{LB,m-1} - \bullet e_m + e_m\bullet$

and thus for $i = 1...m$ $M_{LB,i} \leq M'_{LB,i} \leq M_i$.

NE_m is the set of tokens required for e_m ($\bullet e_m$) that are not explained by the previous state ($M_{LB,m-1}$), and thus need to be added to the previous lower bounds ($M_{LB,0}...M_{LB,m-1}$) to update them ($M'_{LB,0}...M'_{LB,m-1}$) so that e_m could occur. The lower bound marking for $M'_{LB,m}$ is then simply the previous marking $M'_{LB,m-1}$ with the effect of e_m included. The lower bound markings associated with a stream of events can be calculated iteratively using Lemma 1 to initialize the lower bound markings based on the first event and Lemma 2 to update the lower bound markings for each subsequent event.

Next, the upper bound marking restrictions based on the type A resources are determined. The general idea with these restrictions is that an $STPR$ model has at least some type A resources and because these resources are limited to known quantities (e.g., number of robots in system), they impose restrictions on the markings. For example, a single robot is used for exactly one thing at a time and this must be reflected in the marking – a token associated with the robot is in exactly one place. Because only a lower bound on the markings is known, this conservation constraint which should be an equality is instead relaxed to an inequality.

Theorem 3: Given an $STPR$ (N, M_0), the set of reachable markings $R(N, M_0)$ is upper bounded by the resource constraint equations $\Sigma_{i=1}^{||P||} y_j M(p_i) = c_j$ for every p-semiflow y_j associated with a type A resource ($j = 1...$number type A resources), where $c_j = \Sigma_{i=1}^{||P||} y_j M_0(p_i)$.

Proof: One of the properties of an $STPR$ is that each type A resource is associated with a p-semiflow. An important property of p-semiflows is that for any p-semiflow y, $M^T y = M_0^T y$ for any given initial marking M_0 and any $M \in R(N, M_0)$ [12]. For a given initial marking M_0 and p-semiflow y_j, $M^T y_j = \Sigma_{i=1}^{||P||} y_j M(p_i)$ and $M_0^T y_j$ is equal to constant c_j, which means that $\Sigma_{i=1}^{||P||} y_j M(p_i) = c_j$. Thus from [3], the $STPR$ N is conservative with respect to each p-semiflow y_j, and each of these p-semiflows enforces a restriction on the marking expressed by the previous equation. ■

Each resource constraint inequality is of the form

$$y^1 M(p_1) + ... + y^{||P||} M(p_{||P||}) \leq max \tag{1}$$

where

$$max = y^1 M_0(p_1) + ... + y^{||P||} M_0(p_{||P||}) \tag{2}$$

$y^1...y^{||P||}$ are the elements of a p-semiflow y and also the coefficients of the inequality, and $y^1 M_0(p_1) + ... + y^{||P||} M_0(p_{||P||})$ is a constant and the maximum value of the inequality. These lower and upper bound constraints can be used together to create a necessary condition for a model to produce an event stream.

Theorem 4: If there exists a prefix $\hat{\sigma} = e_1...e_j$ of an event stream $\sigma = e_1...e_m$, $j \leq m$, for which any marking in its sequence of lower bound markings ($M_{LB,0}...M_{LB,j}$) exceeds any of the resource conservation constraints of a model, then that model could not have generated σ.

Proof: Using proof by contradiction, suppose the "if" conditions hold but the "then" conditions do not. In other words, there is a violation of at least one of the resource conservation constraints, but the model could generate the event stream and would create a valid sequence of markings associated with it. Let σ be an event stream and $\hat{\sigma}$ be a prefix of σ such that for at least one $M_{LB,i}$ $i = 0...j$, there is at least one p-semiflow y_k such that $M_{LB,i}^T y_k > M_0^T y_k$. Because y_k is a p-semiflow, and M_0 is a marking, $y_k \geq 0$ and $M_0 \geq 0$, hence $M_0^T y_k \geq 0$. Similarly, $M_{LB,i} \geq 0$. From Lemma 2, $M_i \geq M_{LB,i}$ for each element. Combining this information implies $M_i^T y_k > M_0^T y_k$, which means that this marking is not valid, and hence a contradiction. ■

B. Algorithms for Checking the Necessary Condition

This theory is coded into a set of algorithms. The lower bound calculations in Lemma 2 were coded into Algorithm 1. The resource conservation constraints developed in Theorem 3 are implemented in Algorithm 2. This algorithm has computational complexity that is polynomial in the number of places and transitions (events) in the model due to calculating the p-semiflows using singular value decomposition,

Algorithm 1 Calculate Lower Bound Marking

Inputs Previous lower bound (*prevLB*), current event in stream (e), $\bullet e$, $e\bullet$.

1: Calculate the change required for e : *chngReq* $= \bullet e -$ *prevLB*(*prevState*)
2: Calculate what is required for e to occur that has not already been explained: *NE* $= \max($*chngReq*$, 0)$
3: Update *prevLB* by adding *NE* to each state in it
4: Make newest entry to lower bound as *newState* $=$ *prevLB*(*prevState*) $- \bullet e + e\bullet$
5: Create *newLB* as concatenation of updated *prevLB* and *newState*

Outputs new lower bound (*newLB*)

although it only needs to be run once for each model. This algorithm checks a condition that is only necessary, and not sufficient – the lack of sufficiency can be seen as a trade-off for reduced computational complexity.

Algorithm 2 Calculate Resource Constraint Equations

Inputs Model, events

1: Calculate the p-invariants as solutions to $A^T y = 0$
2: Determine which p-invariants are p-semiflows (have all non-negative values)
3: **for** each p-semiflow **do** ▷ create a resource constraint (RC)
4: Max value for constraint is p-semiflow times initial marking of model
5: Coefficients for constraint are p-semiflow
6: **end for**

Outputs Resource constraints (RC) for model

The necessary condition is checked in Algorithm 3, which requires the results from Algorithm 2 and calls Algorithm 1. The computational complexity of Algorithm 3, which is the online portion of checking the condition, is linear in the number of events (transitions). This condition is only necessary – there may be some event streams the model cannot generate that do not violate these constraints, and thus will not be found by checking this condition. In contrast to the other approaches, however, this approach is valid for $STPR$ models and avoids the computational complexity of creating reachability graphs.

IV. Application to Small Manufacturing Cell

A. Description of Small Manufacturing Cell

This manufacturing cell, illustrated in Figure 2, interacts with two different components, called $C1$ and $C2$, and performs machining. The cell has two resources – a robot (R1) to transport components, and a milling machine (R2) to mill components. When a component $C1$ arrives at the cell, the robot will get it, and if the milling machine is available, place it in the milling machine for further processing, after which the milling machine pushes it out of the cell. If the milling machine is not available, the robot will place it in a queue for $C1$ components waiting to be processed. When a component $C2$ arrives at the cell, the robot will get it and place it in a queue for $C2$ components, where removal from this queue is outside the context of this cell. Thus, the process performed by this cell is naturally broken down into two processes, one for the handling of each component.

Algorithm 3 Check Necessary Condition

Inputs Event stream ($e_1 e_2 \dots e_m$), resource constraints (RC), $e_i\bullet$ and $\bullet e_i$ for each e_i

1: Initialize *prevLB* $= [e_1 \bullet \quad \bullet e_1]$
2: **for** each event $e_2 \dots e_m$ **do**
3: Call Algorithm 1 to update the lower bound *newLB*
4: **for** each resource constraint (RC) **do**
5: Calculate the left-hand-side (LHS) of eq. 1 using *newLB* and RC's coefficients
6: If LHS $>$ max from RC, then anomaly present.
7: **end for**
8: **if** none of the resource constraints are violated **then**
9: Event can be produced
10: Update *prevLB*= *newLB*
11: **end if**
12: **end for**

Outputs If model can produce stream, and if not, at what location in event stream model first cannot produce it.

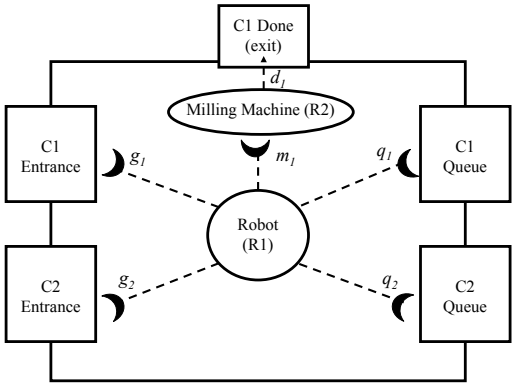

Fig. 2. Illustration of manufacturing cell, where dashed lines show possible movements of the robot and milling machine, and their associated events (in italics)

The cell's events are listed in Table I and illustrated in Figure 2. Resource information is summarized in Table II, including the interaction between resources and events. An $STPR$ model of this manufacturing cell is shown in Figure 1, and this model will be used to show how the algorithms work.

B. Application of Algorithms to Cell

To illustrate the algorithms, we apply them to the model of the example small manufacturing cell, shown in Figure 1, and an event stream: $g_2 q_2 e_1 q_1$. First the resource constraints must be generated for this model using Algorithm 2.

TABLE I

EVENTS IN MANUFACTURING CELL

Name	Process	Description
g_1	1	**G**et $C1$
m_1	1	Put $C1$ in **M**achine, begin processing it
d_1	1	$C1$ is **D**one
q_1	1	Put $C1$ in **Q**ueue
g_2	2	**G**et $C2$
q_2	2	Put $C2$ in **Q**ueue
e_1	1	**E**nd of processing $C1$

TABLE II

RESOURCE INFORMATION FOR MANUFACTURING CELL

Name	Type	Acquire Events	Release Events
R1 (robot)	A (Always)	g_1, g_2	m_1, q_1, q_2
R2 (machine)	A (Always)	m_1	d_1
NotR2 (neg. machine)	NA	d_1, q_1	m_1, q_1

- Step 1: The model's incidence matrix is

$$A = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ -1 & 1 & 0 & 1 & -1 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

  where the places (rows) are ordered [P1 P2 P3 P4 R1 R2 NotR2 $]^T$ and the events (columns) are ordered as in Table I, and based on that, its p-invariants are

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- Step 2: The p-semiflows are $y_1 = [0011100]^T$ and $y_2 = [1100010]^T$.
- Step 3 for first p-semiflow: Max value for constraint is $M_0^T y_1 = [0000110] * [0011100]^T = 1$ (where M_0 is the initial marking of Figure 1), and coefficients are p-semiflow itself $[0011100]^T$.
- Step 3 for second p-semiflow: Max value for constraint is $M_0^T y_2 = 1$, and coefficients are p-semiflow itself $[1100010]^T$.

Next these resource constraints and $g_2 q_2 e_1 q_1$ are used as input to Algorithm 3.

- Step 1:

$$prevLB = [\bullet g_2 \; g_2 \bullet] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

- Step 3 for second event (q_2): use Algorithm 1 to update lower bound
  - Step 1: *chngReq*= $\bullet q_2 - prevLB(:,2) = [0000000]^T$
  - Step 2: *NE* $= \max(chngReq, 0) = [0000000]^T$
  - Step 3: updated *prevLB*= *prevLB*+ *NE*
  - Step 4: *newState* = *prevLB*(*prevState*) $- \bullet q_2 + q_2 \bullet = [0001000]^T - [0001000]^T + [0000100]^T = [0000100]^T$
  - Step 5: *newLB* =

$$[updated prevLB \; newState] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Step 4 for second event (q_2): for each of the three states in *newLB*

$$[0000100]^T, [0001000]^T, [0000100]^T$$

  checked $M^T y_1 \leq 1$ and $M^T y_2 \leq 1$ for each state M.
- Step 8 for second event (q_2): both constraints held, so the model may be able produce the stream $g_2 q_2$. Note that the uncertainty is because the algorithms check a necessary, but not sufficient, condition for a model to produce an event stream from an unknown initial state.
- Repeat Step 2 for third and fourth events (e_1 and q_1), and find that for fourth event, the constraint for resource 1 (R1) is violated and thus the model may be able to produce the stream $g_2 q_2 e_1$, but cannot produce the stream $g_2 q_2 e_1 q_1$.

V. SUMMARY OF SOLUTION AND APPLICATION

To detect anomalies in systems without pre-existing formal models, an anomaly detection solution was developed [2]. This solution is a form of supervised learning for binary classification. Given a set of data that belongs to one of two classes, a classifier (a black box decision-maker or function) is learned. The classifier can then be used to classify future data produced from the same system [14] [7]. In this case, the input is a set of observed event streams that are labeled as to whether or not they contain an anomaly, some information about the resources in the system, and an unlabeled event stream in which we wish to detect if there is an anomaly. The output is a label (normal or anomalous) for the previously unlabeled event stream, and if the event stream is anomalous, then at which event in the stream the anomaly was first detected. The solution has three main steps:

1) Model Generation
2) Performance Assessment
3) Anomaly Detection

These steps, including their inputs and outputs and relationships to one another, are illustrated in Figure 3. The model generation step uses the labeled event streams and the resource information to learn a set of models of the system's

behavior that can act as classifiers. The second step is performance assessment, which consists of using additional labeled event streams to assess the models' performance on detecting anomalies. Finally, anomaly detection itself is done in the third step, where the unlabeled event stream is classified based on the models' agreement or disagreement with the streams and their previous performance.

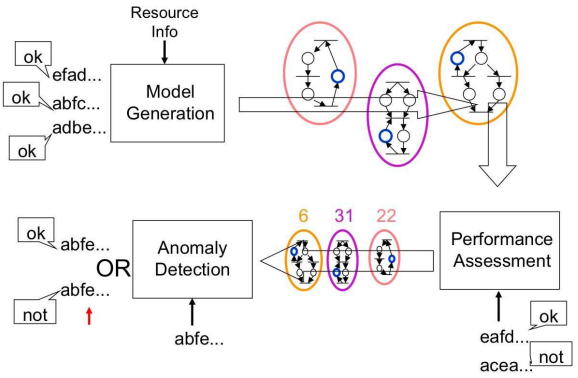

Fig. 3. Overview of Anomaly Detection Solution Steps

The performance assessment and anomaly detection steps of this solution require that event streams be checked as to whether a model could have produced them. Because application to an industry system is a main purpose of the anomaly detection solution, and the event streams may start in an unknown initial state, the anomaly detection solution requires the algorithms developed herein to check whether the generated models can produce the event streams for performance assessment and anomaly detection. If a model cannot produce an event stream, as evidenced by the necessary condition not holding, then the model indicates that the event stream is anomalous.

Application to the anomaly detection solution is illustrated through continuing the example from Section III. A set of 35 event streams from this manufacturing cell – 25 normal and 10 anomalous – were used for performance assessment and anomaly detection. The anomalous streams were generated by taking normal streams and altering them so that the example model could not have produced them. These event streams start from a variety of states and are randomly assigned for either performance assessment or anomaly detection, both of which use Algorithm 3 to test the necessary condition for a model to have produced the given event stream. Using this algorithm, the performance assessment and anomaly detection algorithms were run. The anomaly detection was run on 19 event streams with a total of 410 events, resulting in 100% of the event streams labeled correctly. Thus, the theory and algorithms developed to check whether a model can produce an event stream from an unknown initial state enables this anomaly detection solution.

VI. Conclusions and Future Work

This paper developed a necessary condition for a particular type of Petri net, $STPR$, to produce a given event stream using a lower bound on the Petri net's markings due to the events in the stream and an upper bound on those markings due to resource conservation. This condition can assist in determining whether an $STPR$ can produce an event stream that starts from an unknown initial state. By developing this condition and algorithms to check it, the anomaly detection solution from [2] is more applicable to industry systems.

The condition for a model to produce an event stream from an unknown initial state is only necessary and not sufficient. Thus, there may be some situations in which a model cannot produce a particular event stream, but the necessary condition is not violated. Only a necessary condition is possible because a concise representation of the reachable states is not possible for $STPR$s, as described in Section II.

Future work includes both developing a concise representation of the reachable states for the $STPR$ formalism, and exploring the conditions under which a model can produce an event stream from an unknown initial state when a concise representation of reachable states exists.

References

[1] L. V. Allen & D. M. Tilbury. Event-Based Fault Detection of Manufacturing Cell: Data Inconsistencies Between Academic Assumptions and Industry Practice. *Proceedings of the 6th IEEE Conference on Automation Science and Engineering*, August 2010.

[2] L.V. Allen. *Verification and Anomaly Detection for Event-Based Control of Manufacturing Systems*. PhD thesis, University of Michigan, December 2010.

[3] C.G. Cassandras & S. Lafortune. *Introduction to Discrete Event Systems*. Massachusetts: Kluwer Academic Publisher, 1999.

[4] D. Corona, A. Giua, & C. Seatzu. Marking Estimation of Petri Nets with Silent Transitions. *Proceedings of 43rd International Conference on Decision and Control*, The Bahamas, 2004.

[5] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, & A. J. M. M. Weijters. "Process Mining: Extending the α-algorithm to Mine Short Loops." prom.win.tue.nl

[6] A. Giua. Petri Net State Estimators Based on Event Observation. *Proceedings of the 36th Conference on Decision & Control*, 4086-4091, San Diego California, 1997.

[7] T. Hastie, R. Tibshirani, & J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer Science+Business Media Inc. 2001.

[8] S. Klein, L. Litz, & J-J. Lesage. "Fault Detection of Discrete Event Systems Using an Identification Approach." *Proceedings of the 16th IFAC World Congress*, 2005.

[9] L. Li & C. N. Hadjicostis. Minimum Initial Marking Estimation in Labeled Petri Nets. *2009 American Control Conference*, 5000-5005, St. Louis Missouri, 2009.

[10] E. W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *Proceedings of the 13th Annual ACM Symposium*, 1981.

[11] M. E. Meda-Campana & E. Lopez-Mellado. "Incremental synthesis of Petri net models for identification of discrete event systems." *Proceedings of the 41st IEEE Conference on Decision and Control*, 805-810, December 2002.

[12] T. Murata. "Petri Nets: Properties, Analysis and Applications." *Proceedings of the IEEE*, **77**(4): 541-580, 1989.

[13] M. Roth, J. J. Lesage, & L. Litz. "Distributed identification of concurrent discrete event systems for fault detection purposes." *European Control Conference*, August 2009.

[14] S. Russell & P. Norvig. *Artificial Intelligence: A Modern Approach*. New Jersey: Pearson Education Inc., 2003, Chapter 18.

[15] W. van der Aalst, T. Weijters, & L. Maruster. "Workflow Mining: Discovering Process Models from Event Logs." *IEEE Transactions on Knowledge and Data Engineering*, **16**: 1128-1142, 2004.