Formal Modeling Approach for Supply Chain Event Management

Rong Liu and Akhil Kumar
Smeal College of Business
Penn State University
University Park, PA 16802, USA
{rul110,akhilkumar}@psu.edu

Wil van der Aalst
Faculty of Technology Management
Eindhoven Technical University
Eindhoven, Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract: As supply chains become more dynamic, it is important to be able to model them formally as business processes. In particular, there is a need for a sense-and-respond capability to react to events in a real-time manner. In this paper, we propose Petri nets extended with time and color (case data) as a formalism for doing so. Hence, we describe seven basic patterns that are used to capture modeling concepts that arise commonly in supply chains. These basic patterns may be used by themselves and also combined to create new patterns. Next, we show how to combine the patterns to build a complete Petri net and analyze it using dependency graphs and simulation. Dependency graphs can be used to analyze the various events and their causes. Simulation was, in addition, used to analyze various performance indicators (e.g., fill rates, replenishment times, and lead times) under different strategies. We showed it is possible to performed sensitivity analysis to study the effect of changing parameter values on the performance indicators. This approach thus makes a very complex problem tractable.

Keywords: Supply Chain Event Management, Petri nets, time Petri nets, colored Petri nets, event causality, dependency graph, reachability.

1. Introduction

The pressures of global competition and the need for extensive inter-organizational collaboration are forcing companies to streamline their supply chains and make them agile, flexible and responsive. Consequently, a supply chain must be able to handle large numbers of events, both expected and unexpected. The unexpected events, also called *exceptions*, typically arise because there is usually a gap between supply chain planning and execution [3]. Supply chain planning sets a target that can be achieved based on a given set of constraints at a given time. In a dynamic supply chain environment, the constraints are always changing, so exceptions or deviations from plans occur almost regularly. Examples of exceptions are inaccurate forecast, product out-of-stock, shipment delay, etc., and they are costly. Moreover, events tend to propagate in

collaborative supply chains across partners, resulting in the well-known bullwhip effect [13]. Such risks have given rise to a new research area of Supply Chain Event Management (SCEM). The goal of SCEM is to introduce a control mechanism for managing events, in particular, exceptions, and responding to them dynamically.

A supply chain event is "any individual outcome (or non-outcome) of a supply chain cycle, (sub) process, activity, or task" [2]. Events are correlated with each other to form a "cloud" of events; some events have significant consequences and therefore they must be monitored closely, while others are of lesser importance. The critical problem lies in extracting the significant events and responding to them in real-time. Doing so requires an ability to monitor them proactively, simulate them to help decision-making, and use them to control and measure business processes [21, 26]. In this paper, we present a methodology that uses a Petri net approach to formulating supply chain event rules and analyzing the cause-effect relationships between events.

Petri-nets are a powerful modeling technique for problems involving coordination in a variety of domains. A variant of Petri-nets called *time Petri-nets* allows us to model time intervals also. Considering the dynamic characteristic of supply chain events, such Petri nets are useful for describing the time constraints associated with events. Examples of time constraints are: "*event* e_1 *follows event* e_2 *after time T*" and "*N occurrences of event* e_1 *within time T lead to event* e_2". These temporal constraints are important for proper correlation between events; otherwise, the management could be unable to anticipate events or track causes of events. To deal with variety in case data (e.g., order ids, order quantities, rush orders versus normal orders, etc.) we extend the model with "token colors", i.e., we use *time colored Petri-nets*.

Using time colored Petri-nets we can model event patterns common in Supply Chain Management (SCM). These patterns can be composed as will be demonstrated using a Vendor Managed Inventory (VMI) example. To demonstrate that the Petri-net basis allows for different types of analysis we used CPN Tools [23] to simulate different scenarios for the VMI example. The mapping onto CPN Tools allows us to investigate the performance of event resolution strategies. In addition, dependency graphs are used to analyze cause and effect relationships of events.

There are a variety of SCEM systems from companies, such as SAP, i2, and Manugistics [26]. Most systems mainly perform monitoring and provide "early warning" rather than analyzing

events and suggesting solutions [3, 5, 17, 19, 21, 26,]. Actually, the more powerful part of SCEM would be the capability of "aggregating data from key business systems at a high level and presenting the ramifications of exceptions and the possibilities of solutions" [17]. Therefore, this research can contribute to the research area of SCEM in three ways. First, this work introduces a formal and general approach to modeling events and event rules, and the approach provides flexibility in associating occurrence counts and temporal constraints with events, avoiding the customization problem which often poses as an obstacle to the implementation of the existing SCEM systems [5]. Second, this approach allows excellent event analysis, including event forecasting with time information and causality analysis, which provide real-time visibility about the implications of events and traceability to the root causes for events. Third, it offers a way to track supply chain performance metrics by events, and shows how through simulation decision makers can compare different strategy alternatives or fine-tune a solution in terms of key performance indicators.

The paper is structured as follows. Section 2 gives an overview of events, event rules, event aggregation, event causality, and our notion of a dynamic supply chain. Section 3 describes Petri nets briefly. Section 4 introduces event semantics and seven event patterns or building blocks of event rules. It shows that a complete event Petri net can be constructed easily by using these blocks. Section 5 presents a detailed example to illustrate how to use Petri nets to examine event causality and forecast subsequent events. Section 6 gives simulation results of the example to illustrate the practical value of our approach. Section 7 describes related work and compares our approach with others, while Section 8 concludes the paper with a brief description of future work.

2. Overview of Supply Chain Events

When supply chain partners are integrated, events at one partner may have impact on other partners, and their responses to these events may cause a storm of events. Therefore, causality analysis is the key to controlling such a storm. Our analysis begins with events and event rules.

In general, events in an organization occur in the following three types: (1) *Task status related events,* such as the end of a task or the beginning of a task. These events are usually regular; (2) *Events produced by a task*: for example, events "stock partially available" and "out of stock" are the result of the "check availability" task; and, (3) *External events* which may arrive from other

supply chain partners or from the external environment, e.g., *new order arrival, inbound shipment delay, import policy change* etc.

These types of events are captured directly during a process, and called *simple or primitive* (as opposed to *composite*) *events*. Composite events are *derived* from simple events by *event aggregation*. A composite event is deduced when a group of simple events occurs [16]. A group of simple events may together reveal potential problems. For example, if a product is out of stock once in a month, perhaps it is quite normal and an alarm should not be generated, but *if this stock out happens two times in a week, then it may reflect some underlying problems in the product supply chain and this should be recognized by generating an event.* As another example, a group of stock trading events, related by accounts, timing and other data, taken together, may constitute a violation of a policy or a regulation [16]. *Event aggregation* is a mechanism to filter simple events and extract meaningful information from them by setting up alarms in advance.

Thus, *event aggregation* extracts value from a management point of view out of trivial and unorganized simple events. In order to achieve this objective, it is important to recognize event patterns and set up *aggregation rules*. Besides aggregation rules, *business rules* must also be considered. Business rules capture the causal relationships between events. For example, if an order is delayed for more than time *T*, then it is automatically cancelled. Therefore, a rule is needed to express that the event "order delayed by *T*" is a cause of event "order cancelled".

Moreover, a supply chain is viewed as a series of synchronous and asynchronous interactions among trading partners. Usually, when an event, particularly an exception, happens, the trading partner responsible for it may react to this event within a reasonable *resolution time* to resolve it. For instance, suppose an order is delayed for delivery. If the delay is within an acceptable range specified by the customer, the customer is notified of the delay and the order is processed. However, if the delay exceeds the acceptable tolerance (also called *expiration time*), the order should be automatically cancelled, and hence, the event "order delay" is not relevant in this case. On the other hand, a series of new actions arise because of this new event, such as canceling the order, removing any reservations made, refunding any payments, etc. Therefore, to model events and event rules precisely, our modeling approach should be able to capture such temporal constraints correctly. In our analysis, each event is associated with two time values: *resolution time* and *expiration time*. In most cases, event resolution takes an unpredictable amount of time

because of complexities of various business situations and it is more realistic to set up a resolution time interval. We will show how to capture the dynamic aspect of events in the later sections.

3. Petri Net Preliminaries

A Petri net is a directed graph consisting of two kinds of nodes called *places* and *transitions*. In general, places are drawn as circles and transitions as boxes or bars. Directed arcs connect transitions and places either from a transition to a place or from a place to a transition. Arcs are labeled with positive integers as their weight (the default weight is 1). Places may contain tokens. In Figure 1, one token is represented by a black dot in place *p1*. A marking is denoted by a vector M, where its p^{th} element $M(p)$ is the number of tokens in place p. The firing rules of Petri nets are [22]:

(1) A transition t is *enabled* if each input place of t contains at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc from p to t. (By default, $w(p,t)$ is 1.)

(2) The *firing* of an enabled transition t removes $w(p,t)$ tokens from each input place p of t, and adds $w(t,p)$ tokens to each output place p of t, where $w(t,p)$ is the weight on the arc from t to p.

There is another special type of arc called the inhibitor arc with a small circle rather than arrow at the end. An inhibitor from a place to a transition prohibits the transition from being enabled, and thus firing, if there is a token in the place. An example of an inhibitor arc is given later.

In this paper, we use *Time Colored Petri Nets* (TCPN), i.e., Petri nets extended with *time intervals* and *token values*. First of all, the above classical Petri nets can be extended by associating a time interval $[I_1, I_2]$ with each transition, where I_1 (I_2) is the *minimum* (*maximum*) time the transition must wait for before firing *after* it is enabled. Such a Petri net is known as Time Petri net (TPN) [27]. If $I_1 = I_2$, we just associate one time value with each transition[1], while if the interval is not specified, then $I_1 = I_2 = 0$. Analysis techniques for TPNs are discussed in [4, 27]. Second, tokens can be tagged with data values (or a color) to create a colored Petri net (CPN)

[1] The *Time* Petri Nets discussed in this paper should not be confused with *Timed* Petri Nets. A Petri net is called *Timed* Petri net [27, 30] if each transition is associated with a fixed time instead of a time interval. The two types of Petri nets have very different semantics. As discussed in [4], *Time* Petri Nets are more general than *Timed* Petri Nets.

[11, 12]. For example, we use tokens of different colors (or values) for each order or product. For a given place, all tokens must be from one color set.

In Figure 1, *Q, R* and *S* represent different color sets. *q*, *r*, and *s* are variables, such that $q \in Q$, $r \in R$, and $s \in S$. In a TCPN the arcs are also labeled with colors. For example, in Figure 1, two tokens colored "*q*" are consumed if transition *t1* fires. The fired transition *t1* will put one token colored "*r*" in place *p2*. Moreover, if there are two tokens colored "*q*" continuously existing in place *p1*, transition *t1* will fire no later than time 4. If there is still a token colored "*q*" remaining in place *p1* after time 4 (relative to arrival of this token), transition *t2* will fire shortly after time 4 (denoted as 4+Δ, where Δ is a very short time period, close to 0) and before or at time 8.

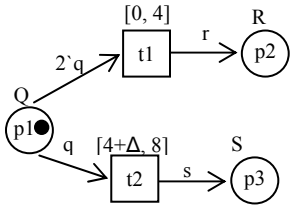


Figure 1: Colored time Petri net

4. Event Formulation and Event Patterns

4.1 Event semantics

Having given a preliminary introduction to Petri nets, now we turn to developing the techniques to formulate event related rules as Petri net structures. In most cases, events are not only the triggers but also consequences of supply chain tasks, i.e. one event causes another event. Therefore, it is quite natural to model events as places that represent pre-conditions or post-conditions of transitions. Thus, events and places will be used interchangeably while modeling events. Moreover, time Petri nets offer an attractive choice for modeling the dynamic aspect in supply chains. To make such models, we first formulate events and event rules as follows:

Event rule *R*: $e_1\ (n_1x_1,\ I_0) \xrightarrow{[I_1, I_2]} e_2(x_2)$, where

- e_1 : input event class
- n_1 : number of event instances (for simplicity, we just say events), i.e., number of tokens (by default, $n_1 = 1$).
- x_i : data value of event *i* for $i = 1,2$. In other words, the color of tokens, $x_i \in$ color set X_i.
- I_0: expiration time of e_1.

$\rightarrow$: "imply" or "lead to", which establishes a cause-effect relationship between the left side and right side of the rule.

$[I_1, I_2]$: an optional time interval which corresponds to the event resolution time. In order not to make the problem trivial, we require $I_2 < I_0$. If this interval is not specified, we assume $I_1 = I_2 = 0$

e_2 : output event class. For every rule, only *one* instance of e_2 is generated because it is not necessary to repeat supply chain events.

This event rule shows the semantics of event e_1 succinctly. Suppose e_1 continues to arrive at a system. If the number of its occurrences reaches a threshold, say n_1, and these events persist in the system long enough, event rule R can be triggered during interval $[I_1, I_2]$, and e_2 is then generated. I_0 is the expiration time of e_1. If rule R does not fire within the $[I_1, I_2]$ interval, then e_1 expires. After e_2 occurs, e_1 may normally be consumed by rule R. However, if e_1 is required by another rule, then a token should be returned to e_1. Hence, two representations are possible for event rules:

Representation 1 (consumption case - e_1 is consumed): This case can be modeled as a Petri net shown in Figure 2. This representation is useful when an event is not required by multiple rules.

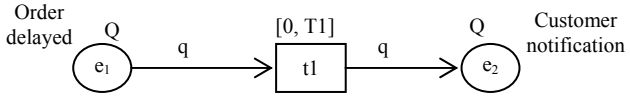


Figure 2: Petri net of Example 1 showing a rule R

Representation 2 (non-consumption case - e_1 is not consumed): Event e_1 is not consumed because it may be required by another rule. Nevertheless, event e_2 must not be generated multiple times from these occurrences of e_1. This case can be accurately modeled as a Petri net as shown in Figure 3.

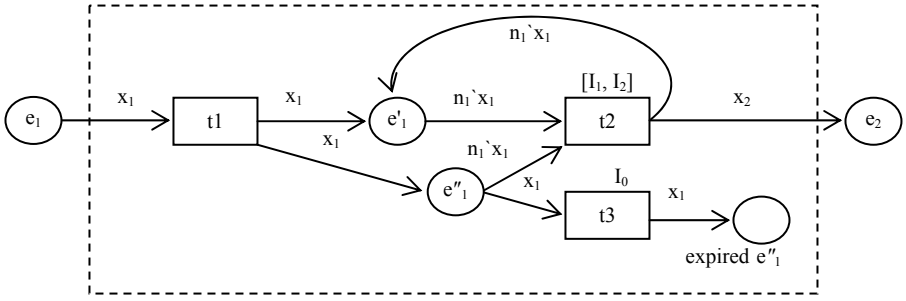


Figure 3: Petri net representation for non-consumption case

When comparing Figure 2 and Figure 3, one can note several differences. First of all, the representation chosen in Figure 3 abstracts from color sets, and focuses on timing issues and causalities. Second, events are not consumed. Third, we consider the situation where n_1 events need to occur to trigger another event. Since event e_1 is not consumed by rule R, we need a special mechanism to prevent event e_2 from being generated repeatedly. Therefore, as Figure 3 shows, Place e_1 is first *transformed* into two places, e'_1 and e''_1, through a transition *t1*. Tokens in e''_1 are consumed if transition *t2* fires, while $n_1 x_1$ tokens (denoted as $n_1`x_1$) are brought back to place e'_1. (Note that n_1 events are needed to enable transition *t2*.) Therefore, after the first firing, although there are $n_1 x_1$ tokens in place e'_1, transition *t2* cannot fire, and thus, at most one e_2 event is generated (with respect to $n_1 x_1$ tokens). If transition *t2* does not fire (because of insufficient tokens in e'_1), e''_1 expires at the end of expiration time by firing transition *t3*. The notion of expiration time will be discussed further in the next section.

These two representations are employed in our patterns in the next section.

4.2 Event patterns to model Supply Chain Rules

Next we will develop several patterns for constructing complex temporal event relationships and also give equivalent logical expressions for these patterns. In general, three logic connectives, OR ($\vee$), AND ($\wedge$), Negation ($\neg$), can be used on either the left or the right side of an event rule. Since modeling of time is crucial in understanding the behavior of our Petri net models, we call these patterns *temporal event patterns*.

We will show later that these patterns can be used as building blocks to create event networks in supply chains. We will demonstrate that these patterns allow us to capture sophisticated relationships involving multiple event instances, event expiration times and resolution times. Thus, this modeling approach can be used to model large varieties of typical supply chain events. We will also illustrate the patterns by examples.

Pattern 1 (**simple cause-result pattern)**: A *cause-result pattern* is the most basic pattern for describing event relationships. It shows that event e_1 can cause event e_2 within a time period [I_1, I_2]. More formally, this relationship is expressed as: $e_1 (x_1) \xrightarrow{[I_1, I_2]} e_2 (x_2)$.

Example 1: If an order is delayed (e_1), contact customer (e_2) before time $T1$, i.e., $e_1 (q) \xrightarrow{[0, T1]} e_2 (q)$ (Note: q is order numbers).

Figure 2 (in the previous section) shows the time Petri net model of this example. Note that *order numbers* can be considered as a color set here, i.e., each order has a different color. We use *Q* to denote this color set, and *q* is a variable for any order in *Q*. Transition *t1* must fire within time *T1* after it is enabled. Transition *t1* corresponds to the action "notify customer".

Pattern 2 (**Repeat_cause-one_effect pattern):** This pattern concerns the case where multiple occurrences of one event within a certain time period cause another single event to occur. Formally, this relationship can be described as: $e_1\ (n_1x_1,\ I_0) \xrightarrow{[I_1, I_2]} e_2\ (x_2)$, where n_1 occurrences of event e_1 for instance x_1 cause event e_2 to occur. There are numerous situations where this pattern is useful.

Example 2: If product *s* is out of stock (e_1) more than once within period *T2*, contact the supply chain manager (e_2). (Note, *s* is the product ID). Formally, this rule can be represented as $e_1\ (2s,\ T2\ ) \xrightarrow{[0,\ \Delta]} e_2\ (s)$.

This example introduces the notion of *expiration time* of events. If an event is not consumed (in this case, event e_1) by a rule, it may expire after a time interval. The Petri net model in Figure 4 represents the time constraints pertaining to these events. Whenever tokens arrive at place e'_1 and e''_1 (as a result of event e_1) transition *t2* and *t3* are enabled, but they cannot fire immediately. When there are two tokens arriving in place e'_1 and e''_1, transition *t1* fires immediately and produces the event e_2, "Notify SC Manager". After transition *t1* fires, two tokens are returned to place e'_1, because event e'_1 may be used by other rules. However, tokens in place e''_1 are consumed, so transition *t1* cannot fire repeatedly. Since transition firing takes no time, *t3* is still continuously enabled. If a token stays in place e'_1 for time *T2* after its arrival, *t3* fires and event e_1 expires. Thus, it is possible that event e'_1 expires without *t1* firing, if there is only one token arriving within interval *T2*. Simultaneously, transition *t2* fires so that e''_1 expires.

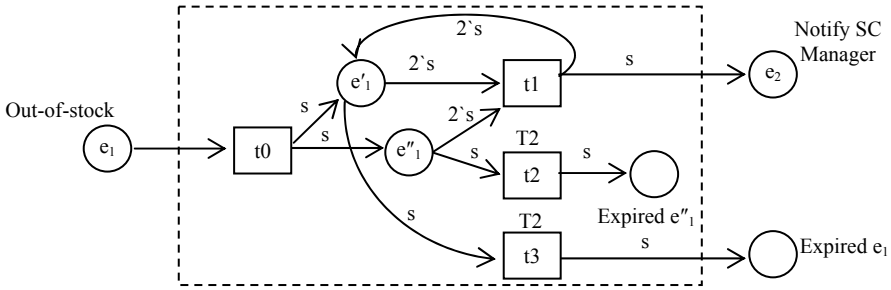


Figure 4: Petri net of Example 2 (Pattern 2)

Pattern 3 (**Inclusive choice)**: The need for this construct arises when multiple, alternative events can occur based on temporal conditions. Formally, this rule can be expressed as:

$$e_0\,(n_0x_0,\, I_{00})\ \{[\xrightarrow{[I_{11},\, I_{12}]}\ e_1\,(x_1)]\ \vee\ [\xrightarrow{[I_{21},\, I_{22}]}\ e_2\,(x_2)]\ \vee\ \ldots\ \vee\ [\xrightarrow{[I_{m1},\, I_{m2}]}\ e_m\,(x_m)]\}$$

Thus, in general, an event e_0 could produce one of many events ranging from e_1 to e_m based on the time intervals associated with these events. In general, these time intervals could overlap; however, by ensuring the intervals are non-overlapping it would be possible to make a deterministic choice based on time. The following example illustrates this pattern.

Example 3: If an order, with lead time *L2*, has not been shipped (i.e., not consumed by some other rule) within time *L2* after it is confirmed (e_0), the order is treated as delayed (e_1) (but e_0 is not consumed yet); however, if an order is delayed by more than time *T3*, it is treated as undeliverable and cancelled (e_2). (Perhaps the customer does not want it if the delay is more than *T3*. So e_0 is consumed at this time.) This example can be formulated as:

$$e_0\,(q,\, L2+T3+2\Delta)\ \{[\xrightarrow{[L2,\, L2+T3]}\ e_1\,(q)\ ]\vee[\xrightarrow{L2+T3+\Delta}\ e_2\,(q)\ ]\ \}$$

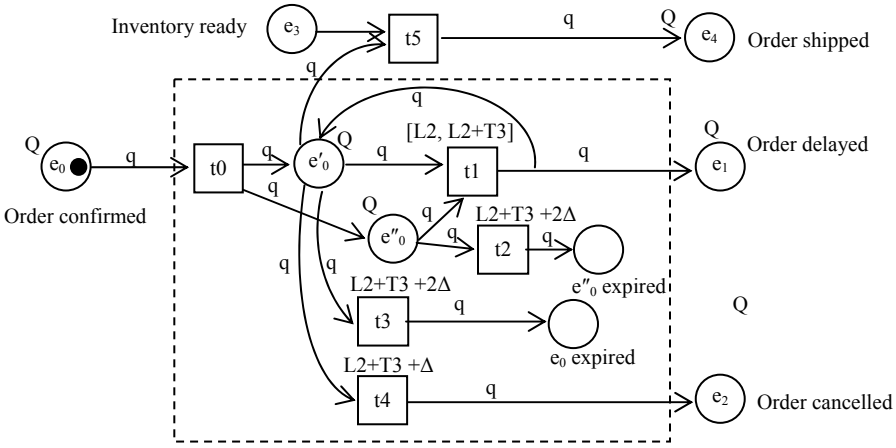


Figure 5: Petri net model of an order process (Pattern 3)

When an order is confirmed (see Figure 5), a token is placed in place e'_0 and e''_0 as well. Transitions *t1*, *t2*, *t3*, and *t4* are enabled but do not fire at that moment. If this token is consumed by the shipment transition *t5* before time *L2* (relative to its arrival), transitions *t1*, *t3*, and *t4* are disabled, but transition *t2* will fire at time *L2+T3+2Δ* after the token arrival. Otherwise, if during the time interval [*L2, L2+T3*] this token remains in place e'_0, transition *t1* will fire. After transition *t1* fires, this token is immediately brought back to e'_0 because some other rules (like *t4*) may use it later. If there is still a token in e'_0 after *L2+T3*, transition *t4* fires and produces event

"order cancelled". Thus, the token in e'_0 is consumed. In general, if this rule is triggered, it can produce two possible results: order delayed and cancelled, or only order delayed, depending upon the temporal relationships. One can see this rule actually has complex semantics, yet its Petri net model can precisely describe such temporal relationships. Note that in Figure 5, transition *t3* never fires and it can be removed. We keep this transition in the figure for consistency with event semantics.

Pattern 4 (**1 of N causes – single result Pattern):** A result can have multiple alternative (combination of one or more) causes. Hence, there is a need for this pattern, and its formal logical expression is as follows:

$$\{[e_1(n_1x_1, I_{10}) \xrightarrow{[I_{11}, I_{12}]}] \vee [e_2(n_2x_2, I_{20}) \xrightarrow{[I_{21}, I_{22}]}] \vee \ldots \vee [e_m(n_mx_m, I_{m0}) \xrightarrow{[I_{m1}, I_{m2}]}]\}\ e_0(x_0)$$

In this expression, the cause of event e_0 may be any one of e_1, e_2, ..., e_m. Typically, this structure could be used to indicate the cause for an event. Figure 6 is the Petri net presentation of this structure, if every source event e_1, e_2, …, e_m is consumed by this rule. Note that the notion of expiration times can be applied to this pattern. For simplicity, the expiration times and expiration transitions are not shown in Figure 6. For example, if transition *t1* does not fire (because of insufficient tokens), e_1 will expire at time I_{10} by firing an expiration transition. The same standard simplification is applied to the next three patterns. A specific example of Pattern 4 is given as below.

Example 4: When a rush replenishment order is rejected (e_1), or delayed (e_2) by more than time *T4* (if the delay is less than *T4*, the delayed time can be compensated by faster shipment), contact alternative vendors (e_0). Logical form: $\{[e_1(q) \xrightarrow{T4}] \vee [e_2(q) \xrightarrow{[0, T5]}]\}\ e_0$.

As the Petri net model in Figure 7 shows, in this example, if an order is rejected by a vendor, an alternative vendor must be contacted in a short interval, say [0, *T5*]. If the order is delayed, a token is put into place e_2 immediately. How long this token remains in place e_2 is exactly the order delay time. If the order is delayed for time *T4*, then transition *t2* has been continuously enabled for the same time, so transition *t2* fires immediately. The fired transition means that, in order to replenish inventory in time, alternative sourcing is required. If the delay does not exceed time *T4* and then the inventory is ready, a fast shipment (e_4) is used to compensate for this delay.

Therefore, through these examples, we see that colored time Petri nets not only present the transformation of events, but also simulate the underlying business activities. The latter advantage cannot be achieved by its logical formulations.

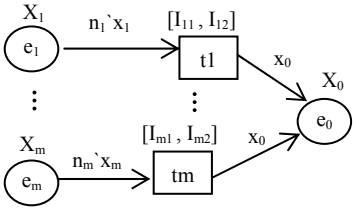


Figure 6: Petri-net for *1 of N causes – single result* (Pattern 4)

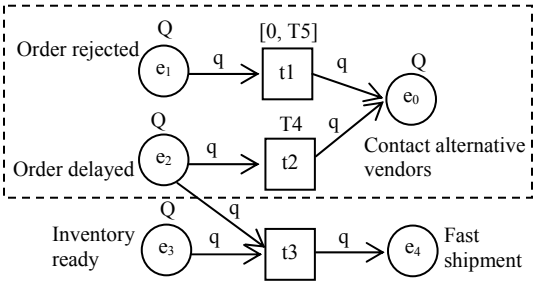


Figure 7: Petri net of Example 4 (Pattern 4)

Pattern 5 (**1 cause – N results Pattern):** This pattern recognizes that a cause may have multiple consequences and captures all *concurrent* consequences of a particular event. Logically, this is expressed as: $e_0\,(n_0x_0, I_{00}) \xrightarrow{[I_{01}, I_{02}]} \{e_1\,(x_1) \wedge e_2\,(x_2) \wedge \ldots \wedge e_m\,(x_m)\}$

In this expression, n_0 occurrences of event e_0 generate m different events concurrently. Figure 8 shows the Petri net representation of this rule. As Figure 8 shows, the m events are represented as output places of transition *t1*, which is enabled by n_0 occurrences of input event e_0. Transition *t1* fires within an interval $[I_{01}, I_{02}]$ after n_0 tokens are placed in e_0. In this case, this Petri net representation shows a *1 cause – N results* pattern.

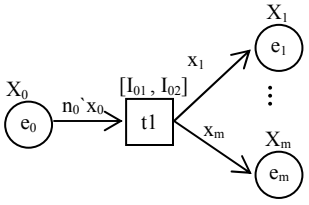


Figure 8: Petri net for *1 cause – N results* (Pattern 5)

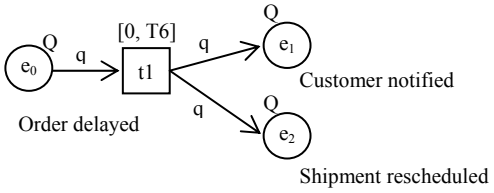


Figure 9: Petri net of Example 5 (Pattern 5)

Example 5: If an order is delayed (e_0), notify customer (e_1) *and* reschedule the shipment (e_2) immediately, i.e. $e_0\,(q) \xrightarrow{[0,\,T6]} [e_1\,(q) \wedge e_2\,(q)]$.

Figure 9 is the Petri net model of Example 5. If *T6* approaches 0, *t1* fires instantaneously. This simple example illustrates that this structure can be used to present the concurrent events that originate from the same cause.

Pattern 6 (**N causes – 1 result Pattern):** This pattern is the reverse of the above pattern, and it is used to model the concurrent causes of a particular event. The following formulation shows

that, there are *m* preconditions, e_1, e_2, ..., e_m, which occur simultaneously to arrive at event e_0. Similarly, assuming every sourcing event is consumed by this rule, this structure can be transformed into a Petri net as shown in Figure 10.

$$\{e_1 (n_1x_1, I_{10}) \wedge e_2 (n_2x_2, I_{20}) \wedge \ldots \wedge e_m (n_mx_m, I_{m0}) \} \xrightarrow{[I_{01}, I_{02}]} e_0 (x_0)$$

As the Petri net model, Figure 10 shows, each precondition can be modeled as an input place of transition *t0*, and the result e_0 is the output place of this transition. This Petri net exhibits an *N causes – 1 result* pattern; so does the Petri net representation of Example 6.

Example 6: When the shipper of a confirmed order (e_2) is not available (e_1), find another shipper (e_3) in a short time *T7*. This rule can be logically formulated as: $[\, e_1 (q) \wedge e_2 (q) \,] \xrightarrow{[0, T7]} e_3 (q)$.

The Petri net of Figure 11 shows the precise representation of this rule. In Figure 11, two events e_1 and e_2 in conjunction produce one result, event e_3.

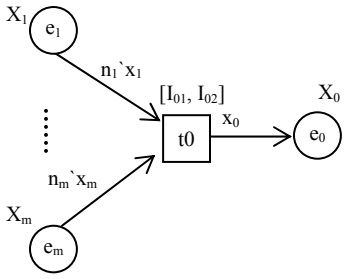


Figure 10: "*N causes – 1 result*" Petri net (Pattern 6)

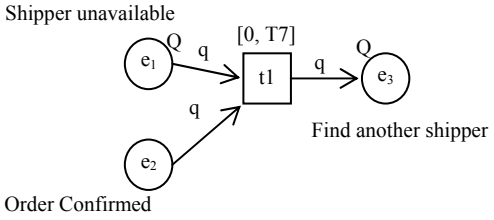


Figure 11: Petri net of Example 6 (Pattern 6)

Pattern 7 (non-occurrence of an event pattern): The above patterns were all based on the occurrence of events. However, non-occurrence of an event can also signal valuable information and hence we need a pattern for that. Negation is usually used to express the non-occurrence of a particular event. Typically, non-occurrence of an event and occurrence of some other events may, in conjunction, cause some other significant events to happen. The following logical formula describes this situation:

$$\{e_1 (n_1x_1, I_{10}) \wedge [\neg e_2 (n_2x_2, I_{20})]\} \xrightarrow{[I_{31}, I_{32}]} e_3 (x_3)$$

Event e_1 and non-outcome of e_2 cause e_3. Actually, if event e_1 is consumed by this rule, this formulation can be transformed into a Petri net similar to Figure 11, except the arc from place e_2 to transition *t1* replaced by an inhibitor arc, as Figure 12 shows. (See [7] for the semantics of inhibitor arcs in colored Petri nets.) In general, Figure 12 and the Petri net representation of Example 7 have a *non-occurrence* pattern.

Example 7: When an order arrives (e_1), if there is *no* out-of-stock (e_2) situation, the order is confirmed (e_3). Logically, this rule can be formulated as: $\{e_1(q) \wedge [\neg e_2(s, T2)]\} \xrightarrow{[0, T8]} e_3(q)$.

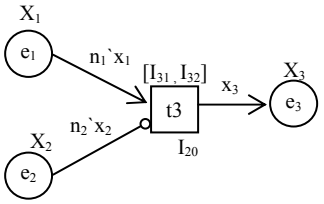


Figure 12: *Non-occurrence* pattern (Pattern 7)

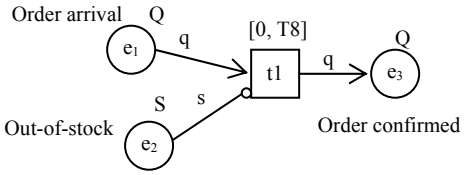


Figure 13: Petri net of Example 7 (Pattern 7)

Figure 13 is the Petri net model of Example 7. After the order arrives, a token is put into place e_1. At that time, if there is no token in place e_2 it means there is no out-of-stock event, and transition *t1* fires in a short time, say [0, *T8*], and puts a token in place e_3 to indicate the order is confirmed. Otherwise, if there is a token in place e_2, it inhibits the firing of transition *t1*.

However, some Petri net tools may not support inhibitor arcs. We can substitute inhibitor arcs with an equivalent structure that works for CPN tools [23] and is described in Appendix 1.

In this section, we have developed 7 basic patterns that capture cause-effect relationships in Petri-nets. Next, we will use an example to show that these patterns can be combined together as building blocks to create more complex Petri-nets.

4.3. Composing new patterns and creating user-defined patterns

Above we discussed 7 basic patterns to capture complex cause effect relationships. Now we demonstrate how they can be combined to create new user-defined patterns. In general, patterns can be combined (or composed) if they have common input or output events (i.e., places that have the same label). By superimposing common places shared by existing patterns, new patterns can be created. This approach has been used in modeling logic programs as Petri nets [25]. Obviously, if two patterns do not share any events then they cannot be directly composed. The possible scenarios for pattern combination are as follows:

(1) The output place of one pattern is the input place of another pattern (sequential)

(2) The two patterns have one or more common input places (Parallel 1)

(3) The two patterns have one or more common output places (Parallel 2)

(4) The two patterns have common input *and* output places (Parallel 3)

When two patterns are composed in sequence, they form more complex cause-effect chains. On the other hand, if they share common inputs, the patterns will compete for firing by taking tokens from the common event places and have exhibit more complex interactions. A detailed analysis of the various possible interactions for the different combinations is beyond the scope of this paper; however, we will illustrate our approach by showing how two new, non-trivial and useful, user-defined patterns can be created.

First, we create a new pattern to "initialize *B* when *A* occurs", as shown in Figure 14. Thus, when event *A* occurs, already existing occurrences of event *B* must be cleared. This *event initialization* pattern can be created by composing existing patterns as follows:

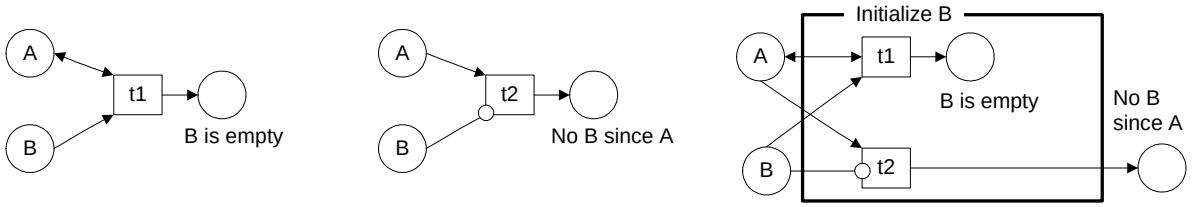
(1) Using Pattern 6, when event *A* happens, if prior *B* events exist, they are cleared (Figure 14(a)).

(2) Then, Pattern 7 is used such that transition *t2* fires when the place for event *B* is empty and puts a token in the place marked "No *B* since *A*" (Figure 14 (b)).

(3) Combine Pattern 6 and Pattern 7 by superimposing common places for events *A* and *B* (Figure 14 (c)).

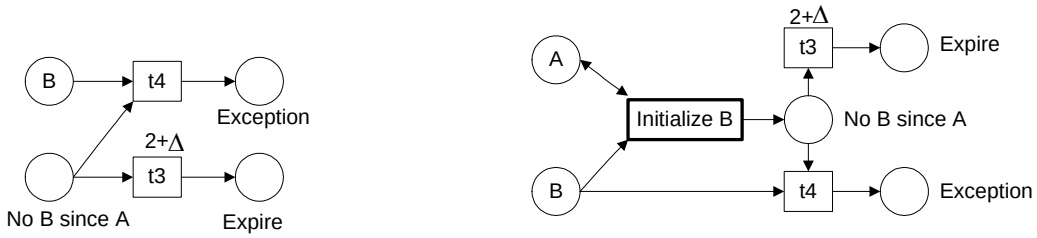
Similarly, in Figure 15 we show how another new pattern called *consecutive events* can be created using this new *event initialization* pattern as a building block. The *consecutive events* pattern generates an exception event when events *A* and *B* (initialized after *A*) happen within a time interval *T*. To create this pattern, we first apply Pattern 6 to places "No *B* since *A*" and *B*, as shown in Figure 15(a). If tokens do not arrive in *B* within the required interval, then tokens in places "No *B* since *A*" are said to expire. Later, we combine the new *event initialization* pattern with Pattern 6, as shown in Figure 15(b). We can foresee various applications for these new patterns. For example, the consecutive events pattern could be used in a supply chain to notify the manager if a "machine breakdown" and "no shipment arrival" events occurred within 1 day. Similarly, in telecommunication applications also such consecutive events would be useful [8].

Clearly, although the seven basic patterns are not exhaustive, the ability to compose them and create new patterns is a powerful feature that allows us to model most realistic situations. Moreover, if necessary, new primitive patterns can also be created from scratch by giving their Petri net description.



(a) Apply Pattern 6 (b) Apply Pattern 7 (c) Combine Patterns 6 and 7

Figure 14: Composing a new *event initialization* pattern by combining Patterns 6 and 7



(a) Apply Pattern 6 (b) Combine *event initialization* and Pattern 6

Figure 15: Composing a new *consecutive events* pattern from *event initialization* and Pattern 6

5. An Example of Event Causality Analysis Using Petri Nets

In this section, we will first show a Petri net that is built using the above seven patterns in the context of a realistic supply chain scenario. Subsequently, we will analyze event causality by simulation and dependency graphs.

5.1. Scenario of events and rules for a complete Petri net

First, we will give an example scenario description. Suppose there is a Vendor Managed Inventory (VMI) arrangement between a distributor and a vendor. In this arrangement, the vendor manages the inventory level for the distributor, proposes the new supply orders to the distributor, and ships them after the distributor's approval. The distributor sells products to its customers, and normally ships its customers' orders (for simplicity, we just call them orders) from stock, but whenever there is an out-of-stock situation, a rush supply order is placed with the vendor. When there is more than one out-of-stock event in a week at the distributor, this situation should be considered as a supply chain exception and reported to the supply chain manager immediately. The vendor would usually respond to rush supply orders as soon as possible, but they may be rejected if there is a serious production delay. Moreover, in case of production delay, all supply orders may be delayed. The distributor can contact an alternative vendor for replenishment in case that its normal or rush supply order is delayed or rejected. Figure 16 shows

all the trading partners in such a supply chain. *For simplicity, in this figure we assume there is one product, but one can similarly model multiple products also as we show in Section 6.* First, we need to identify the events and then write the rules that connect them together. The events of interest are summarized in Figure 17 and each event corresponds to a place in the Petri net.

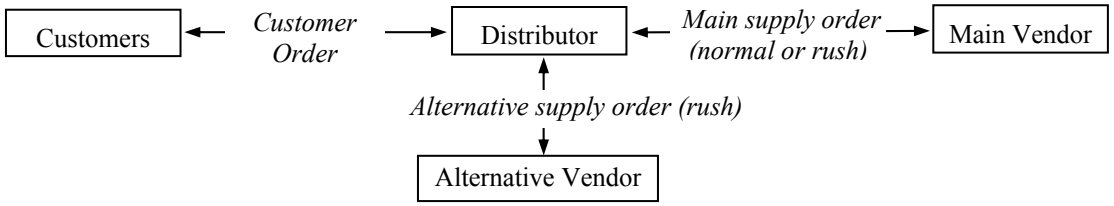


Figure 16: Interactions between trading partners in the example supply chain

Place (or event) description		
p1: Customer order arrival		p2: Out-of-Stock
p3: Back order		p4: Rush supply order
p5: Rush supply order confirmed		p6: Customer order confirmed
p7: Customer order delayed		p8: Notify customer of order delay
p9: Customer order cancelled		p10: Customer order shipped
p11: Out-of-Stock event expires		p12: Notify supply chain manager
p13: Rush supply order rejected		p14: Production delay
p15: Supply order delayed		p16: Contact alternative vendors
p17: Stock unavailable when delivery is due		p18: Rush supply order shipped
p19: Alternative sourcing failed		p20: Customer order rejected
p21: Production delay (p14) resolved		p25: Back order cancelled
p22: p2′ expired	p23: p3′ expired	p24: p5′ expired
p26: p5″ expired	p27: p6′ expired	p28: p2″ expired

Figure 17: Possible events in the supply chain

Next, we consider the rules that relate these events to one another, and also refer to the corresponding patterns used for modeling these rules (in parentheses).

1. When a customer order arrives (*p1*) and there is no out of stock (<u>not</u> *p2*), the order is confirmed (*p6*). (Pattern 7: *Non-occurrence*)
2. When a customer order arrives (*p1*) but there is an out-of-stock (*p2*), a back order (*p3*) is generated. (Pattern 6: *N causes – 1 result*)
3. When a back order occurs, a rush supply order with lead time *L1* is sent to the vendor (*p4*). (Pattern 1: *Simple cause-effect*)
4. When the rush supply order is confirmed by the vendor (*p5*), the back order is also confirmed to the customer (*p6*). A back order must be confirmed within *L2+T3*, where *L2* is the lead

time of the back order, and *T3* is the maximum allowed delay time; otherwise, it expires and is cancelled (*p25*) (Pattern 6: *N causes –1 result*)

5. If there is a production delay (*p14*), any incoming rush supply order is rejected (*p13*), because there is no production capacity left to fulfill any rush supply order in a short time. Otherwise, the rush supply order is confirmed. A production delay can be resolved in time interval [*a*, *b*]. (Pattern 6: *N causes – 1 result*; Pattern 7: *Non-occurrence*)
6. A rush supply order is shipped during time [0, *L1*] if there is no production delay (not *p14*). (Pattern 7: *Non-occurrence*)
7. A production delay (*p14*) can cause a supply order delay (*p15*). If a rush supply order is delayed (*p15*) for more than time *T4*, it leads to unavailable inventory when customer order delivery is due (*p17*). (Pattern 5: *1 cause – N results*; Pattern 1: *Simple cause-effect*)
8. If a rush supply order is rejected (*p13*) or delayed (*p15*) for more than time *T4*, contact alternative vendors for alternative sourcing (*p16*). (Pattern 4: *1 of N causes-single result*)
9. When a rush supply order is shipped (*p18*) by one of alternative vendors, the corresponding back order can be confirmed (*p6*) and shipped (*p10*); otherwise, the customer order can be rejected (*p20*). (Pattern 6: *N causes – 1 results;* Pattern 1: *Simple cause-effect*)
10. When a supply order is shipped from a vendor (*p18*), inventory is available for delivery (so if there is a token in *p17*, it is removed). (Pattern 6: *N causes – 1 result*)
11. When delivery is due, if inventory is available (not *p17*), the order is shipped (*p10*). (Pattern 7: *Non-occurrence*)
12. a. If an order (with lead time *L2*) has not been shipped in time *L2* after it is confirmed (p6), the order is delayed (*p7*).

    b. If an order is delayed (*p7*) more than time *T3*, then the order is cancelled (*p9*). (Pattern 3: *Inclusive choice*)
13. If there are two unresolved out-of-stock events (*p2*) during time *T2*, the supply chain manager is contacted immediately (*p12*). (Pattern 2: *Repeat_cause-one_effect*)
14. If the order is delayed (*p7*), notify the customer at time *T1* (*p8*). (Pattern 1: *Simple cause-effect*)

The above 14 rules can be easily formulated in terms of colored time Petri nets as shown in Figure 18. The darkened places in the figure are input events of this net. Place *p1* contains two different tokens representing the two order arrivals. Events which are not consumed by event

rules are transformed into multiple places, such as *p2*, *p2′*, and *p2″*, where *p2* holds tokens for events, and the others are special mechanism for preventing repetitive firing of transitions.[2]

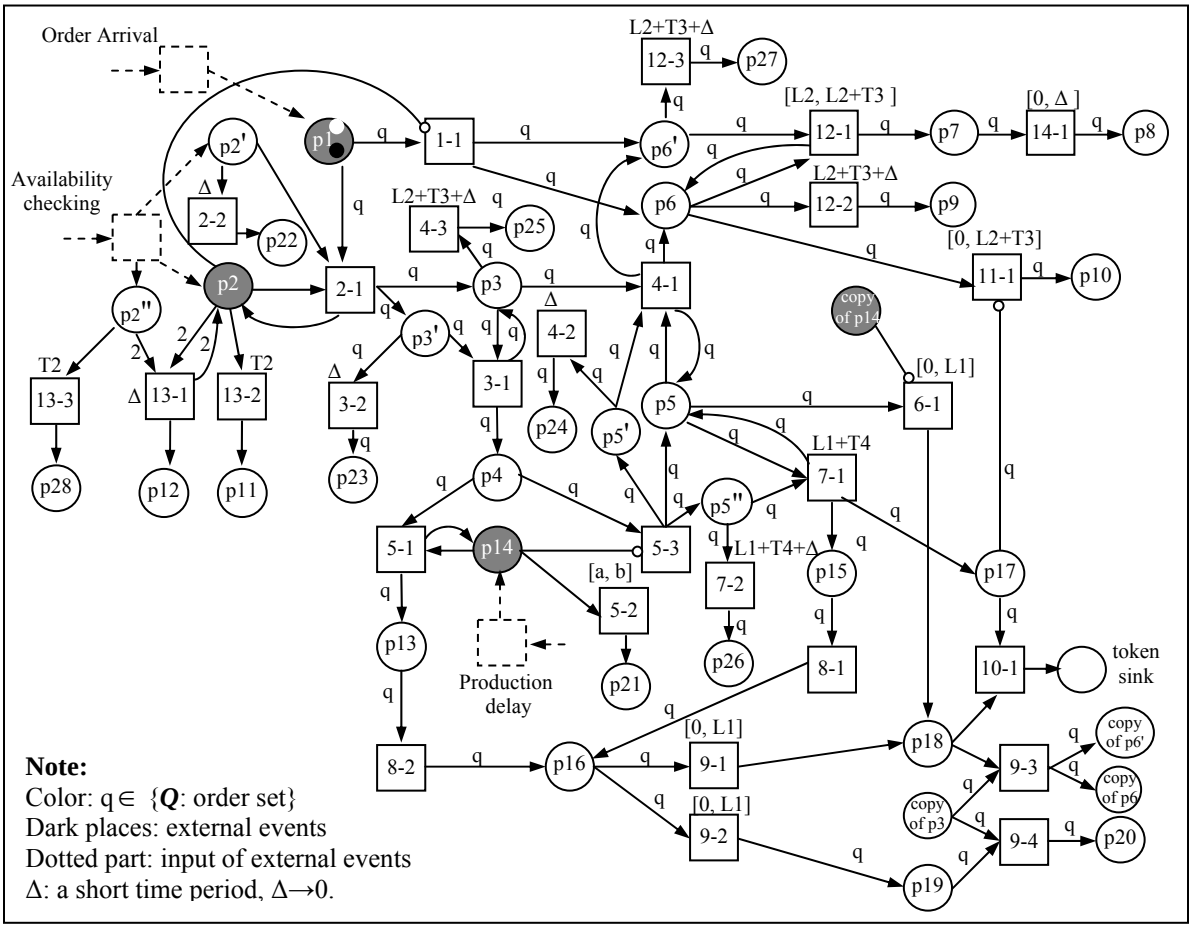


Figure 18: A supply chain event Petri net

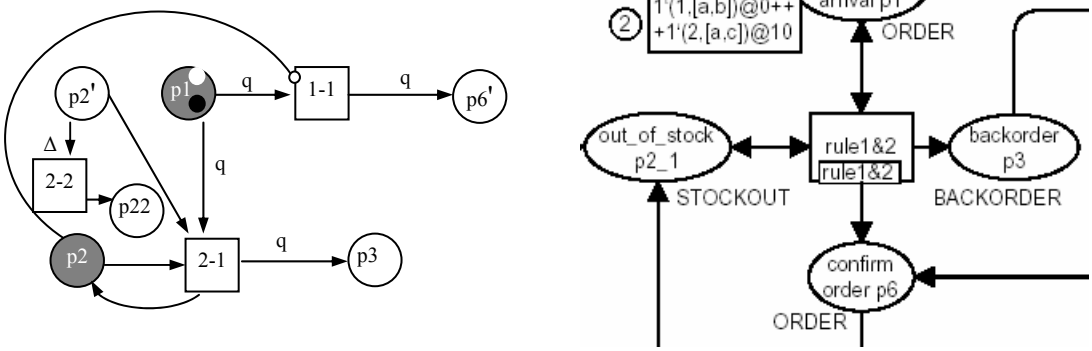
This Petri net was implemented using CPN Tools [23]. CPN Tools is a graphical computer tool supporting Colored Petri nets (CPN). The details of the CPN implementation are given in Appendix 1. In addition, since CPN Tools does not explicitly support time, a workaround is introduced to add temporal constraints to a transition.[3] Figure 19(b) shows an implementation of Rule 1 and Rule 2 from the detailed supply chain example above (See Figure 19(a)) in CPN Tools. In Figure 19(b), "ORDER", "STOCKOUT", and "BACKORDER" are color sets of different places. Place *p1* contains two tokens. For example, 1`(2, [a, c])@10 means there is one token (1`) with color (2, [a, c]) and timestamp 10 (@10). Note that "2" is the order number (of

[2] This point was explained in Section 4.1in the non-consumption case

[3] CPN Tools supports a notion of time. However, since it is an executable language allowing for automatic simulation it requires deterministic or stochastic time. Hence, the interval times are translated into guards based on an explicit clock.

integer data type) and [a,c] is a list of products (of list data type), which here denotes two products, "a" and "c" for order 2. In addition, Figure 19(b) is a hierarchical Petri net. The tag "rule1&2" attached to a transition shows this transition can be substituted with the sub-Petri net of Figure 19(a). It should also be noted that an inhibitor arc is substituted by an equivalent structure with normal arcs, so inhibitor arcs also reflect causal relationships between events. For example, in Figure 19(b), *p6* (confirmed order) depends on both *p1* (order arrival event) and *p2_1* (out-of-stock). More precisely, an occurrence of *p1* and non-occurrence of *p2_1* lead to *p6*.

Details of CPN Tools and hierarchical Petri nets can be found in [11]. Next, we will analyze events using *dependency graphs* based on running this model.



(a) Rules 1 & 2 (b) CPN implementation

Figure 19: Mapping rules 1 and 2 to CPN Tools (from the detailed example)

5.2. Dependency graph Analysis

The Petri net shown in Figure 18 (and its CPN Tool representation shown in Figure 22, Appendix 1) can be considered as an "event machine," i.e. when fed with input events, it will generate a set of composite events (both intermediate and final), and show the causal relationships between them. The behavior of this "machine" for the life of a particular instance or for a given time period can be represented by a simple dependency graph [9]. A *dependency graph* is a cause-effect graph of events produced from one or more Petri net instances (say, one or more orders) over a time period. The dependency graph is created from the Petri net by using the rule that *the output event*(*s*) *of a transition depends upon its input event*(*s*).

By executing the Petri net model with actual case data, we can create dependency graphs to show causal relationships that actually transpired between events. Table 1 describes the sequence of event occurrences and the transitions that fire when the events take place. The relationships are

reflected in Figure 20 that shows an event dependency graph generated based on the Petri net of Figure 18. Moreover, it also gives the correspondence between place numbers and event numbers. (Note that the events and the corresponding place numbers are not always the same.) The table also gives time values in the last column. These times are based on assigning suitable values for a hypothetical case to the parameters of Figure 18 as follows in time units (say, days): $L1 = 20$, $L2 = 50$, $T1 = 1$, $T2 = 50$, $T3 = 20$, $T4 = 10$, $a = 60$, $b = 80$.

Event	Description	Trans. fired	Place	Time
E1	Order *O1* arrival	-	p1	0
E2	Out-of-Stock of product *A* (forO1)	-	p2	0
E3	*O1* is on back order	1-1	p3	0
E4	Rush supply order *R1* is placed for *O1*	3-1	p4	0
E5	Supply order *R1* is confirmed to customer	5-3	p5	0
E6	Order *O1* is confirmed	4-1	p6	0
E7	Order *O2* is received	-	p1	10
E8	Product *A* is out-of-stock (forO2)	-	p2	10
E9	*O2* is placed on back order	1-1	p3	10
E10	Rush supply order *R2* is placed for *O2*	3-1	p4	10
E11	Contact supply chain manager	13-1	p12	10
E12	Product *A* production is delayed	-	p14	10
E13	Rush supply order *R2* is rejected	5-1	p13	10
E14	Alternative vendor is contacted for *R2*	8-2	p16	10
E15	Rush supply order *R1* is delayed for time *T4*	7-1	p15	30
E16	Product A is unavailable when *O1* is due	7-1	p17	30
E17	Alternative vendor is contacted for *R1*	8-1	p16	30
E18	Rush supply order *R2* is shipped from the alternative vendor (i.e., non-occurrence of event "product unavailable when *O2* due")	9-1	p18	30
E19	Order *O2* is confirmed	9-3	p6	30
E20	Order *O2* is shipped	11-1	p10	31
E21	Order *O1* is delayed	12-1	p7	50
E22	Alternative sourcing attempt for *R1* failed	9-2	p19	50
E23	Notify customer about order *O1* delay	14-1	p8	50
E24	Order *O1* is cancelled	12-1	p9	71

Table 1: A trace of possible event sequence generated from Figure 18

Figure 20 enables us to analyze the various events and their causes. The events that represent *exceptions* are shaded in this figure. The consequences of a particular event can be traced forward along this directed graph, while the causes of it should be traced backwards until one or more root nodes are reached. For example, it is not difficult to see that *E8* and *E12* are the main causes of exception *E13*, i.e., product *A* was out of stock with the distributor and a rush supply order *R2* was issued, but this rush supply order was rejected by the vendor because of a

production delay. Similarly, the graph shows that the ultimate exceptions resulting from *E12* are *E21*, *E22* and *E24*. The sequence of main events is as follows:

Production is delayed (*E12*) → *rush supply order R1 is also delayed* (*E15*) → *another vendor is contacted* (*E17*) → *alternative sourcing failed* (*E22*) → *Order O1 cancelled* (*E24*)

Moreover, notice that the exception *E11* (notify supply chain manager) happens because of two stock-out events of product *A* within 50 time units as denoted by events *E2* and *E8*. Thus:

Stock out of A for order O1 (E2) & Stock out of A for order O2 (E8) → Notify supply manager (E11)

Actually, Figure 20 only shows one possible scenario and gives the ultimate disposition of orders *O1* and *O2* (*O1* was cancelled, while *O2* was fulfilled). Figure 21 shows another out-of-stock situation during order fulfillment; however, now the outcome is different. Here, *E16* (Product *A* unavailable when *O1* is due) is resolved by *E18* (Rush supply order for *R2* shipped from the alternative vendor). Therefore, order *O1* is shipped (*E20*) within its lead-time. Later on, in spite of *E21* (alternative sourcing for *R1* fails), rush supply order *R1* is shipped (*E22*) from the main vendor after some delay. Eventually, order *O2* is also fulfilled (*E24*) by the incoming inventory from rush order *R1*. The modified events for this scenario are shown in Table 2 (events *E1* through *E17* are the same as in Table 1). Figure 21 shows the new dependency graph for these events. Nevertheless, *E11* (notify supply chain manager) still happens as before.

Event	**Description**	**Trans. fired**	**Place**	**Time**
… Events *E1* thru E*17* are same as in Table 1 …				
E18	Rush supply order for *R2* shipped from the alternative vendor.	9-1	p18	30
E19	Product available for *O1* (token in p17 removed) (i.e., non-occurrence of event "product unavailable when *O1* due")	10-1	p17	30
E20	Order *O1* shipped	11-1	p10	30
E21	Alternative sourcing for *R1* fails	9-2	p19	50
E22	*R1* Shipped from the main vendor	6-1	p18	80
E23	Order *O2* confirmed	9-3	p6	80
E24	Order *O2* shipped	11-1	p10	80

Table 2: An alternative scenario of events generated from Figure 18

These two dependency graphs show only two of many possible scenarios and serve to illustrate our approach. The advantage of this approach is that using a Petri net model as an event machine we can generate dependency graphs to predict and analyze different "interesting" scenarios. Moreover, by playing "token games", supply chain managers can explore a large number of possible event dependency graphs which lead to desirable results (e.g. order fulfilled successfully)

or significant exceptions (e.g., order cancellation). The design of an algorithm or heuristic that can automatically generate dependency graphs containing such events of interest is left as a future exercise. In this context it should be noted that it is possible to analyze all possible dependency graphs using reachability analysis techniques [4] for time colored Petri nets. However, as discussed there this is not very feasible for large problems for complexity reasons and heuristic techniques are required. Next, we provide a summary of simulation results and analyze their implications for supply chain management.

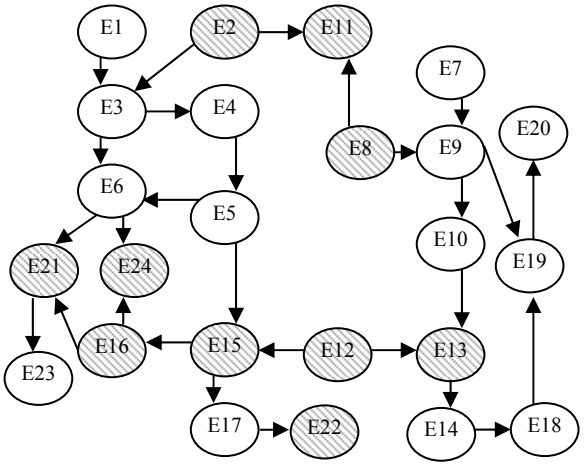


Figure 20: Dependency graph of Table 1 (exceptions are shaded)

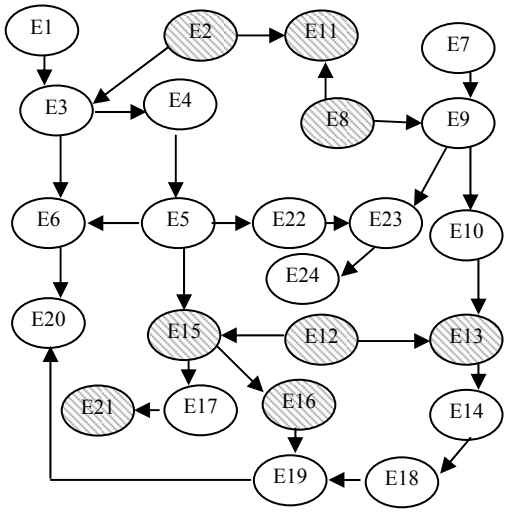


Figure 21: Dependency graph of Table 2

6. Simulation Results

To demonstrate the practical value of our approach, a detailed simulation experiment was conducted. In this simulation, we generated a large number of customer order arrival events and traced the order fulfillment process in terms of times of occurrence of each event. To make the simulation realistic, we assume there are three products, say *A*, *B*, and *C*. In general, more products can also be supported. Table 3 shows the parameters of our simulation experiment.

Parameter Name	Value or Distribution
Set of items in a customer order	Random selection from three products: A, B, and C
Customer order inter-arrival time	Exponential distribution with mean of 7 time units
Prob. of successful alternative sourcing (PSAS)	0.5
Inter-arrival time between production delayed events	Exponential distribution with mean of 100
Resolution time for production delay (RT)	Uniform distribution range [60, 80]
Normal supply arrival schedule	2 arrivals for each item every 30 time units

Table 3: Simulation parameter settings

The simulation runs for a period from 0 to 3500 time units. 500 customer orders are generated and processed. Among them, 445 orders were successfully shipped, and the other 55 orders were cancelled or rejected because of out-of-stock events and failures to find alternative sourcing. In Table 4, the "baseline case" column summarizes the number of main events generated during the simulation interval. Table 4 also shows that although about one quarter of customer orders (135 out of 500) occur in stock out situations, yet most of them (84 out of 135) can still be successfully fulfilled through rush supply orders. In addition, about 10% of customer orders (48 out of 500) are fulfilled by alternative sourcing, which shows that alternative sourcing is important.

Events	**Baseline case**	**Strategy 1**	**Strategy 2**
	RT = [60, 80], PSAS = 0.5	RT = [30, 50]	PSAS = 0.7
Order arrivals (p1)*	500	500	500
-- Customer order shipped (p10)	445	473	475
-- Customer order cancelled (p9)	4	3	1
-- Customer order rejected (p20)	47	21	20
-- Back order cancelled (p25)	4	3	4
Out-of-stock events (p2) *	182	182	182
Production delay (p14) *	35	35	35
Customer order delayed (p7)	4	4	1
Back order (p3)	135	135	135
rush supply order (p4)	135	135	135
rush supply order fulfilled	84	111	111
-- by main vendor	36	77	33
-- by alternative vendors	48	34	78
Rush supply order rejected by main vendor (p13)	102	60	102
Supply order delayed (p15)	8	16	6
Contact alternative vendors (p16)	110	76	108
Alternative sourcing failed (p19)	62	42	30
Performance Indexes			
Customer order fill rate	89%	95%	95%
Average customer order fulfillment time	28	27	28
Average replenishment time of rush supply orders (main vendor)	54	18	27
Average replenishment time of supply orders (alternative vendors)	10	10	11

*: These are input events. The three strategies have the same input events.

Table 4: Comparing different strategies in terms of events

Table 5 shows the detailed distribution of out-of-stock events by product. Each product accounts for about one third of these 182 out-of-stock events. In practice, it may be difficult for a supply chain manager to trace each one of these 182 events individually. Using Rule 13 (see Section 5.1), we can filter these events and reduce the number of events sent to the manager. Thus, the supply chain manager may be notified only when there are *two out-of-stock events* within a 50 time unit interval. Therefore, the number of events which needs management attention is reduced to 80, about 40% of the original number of events. Moreover, the manager can adjust Rule 13 to further reduce this number suitably.

Products	**A**	**B**	**C**	**Total**
Out-of-stock events	53	66	63	182
Notify supply chain manager of out-of-stock events	24	29	27	80

Table 5: Numbers of out-of-stock events

In addition, the events in Table 4 can be used to calculate key performance indexes of the supply chain. As Table 4 shows, the fill rate of customer orders is 89% and the average time between an order arrival and the shipment of the order is 28 time units. In addition, on average, it takes 54 time units for the main vendor to replenish rush supply orders, because production delays occur frequently (35 delay events) and they last a while before being resolved. In contrast, it takes a shorter average time (10 time units) to get supplies from alternative vendors. In general, since the customer order fill rate is somewhat low, the performance of this system may need to be improved. We show next how this can be done with our approach.

An important aspect of our approach is the ability to do sensitivity analysis. To show how such analysis can help to improve the performance of this supply chain, we alternately considered the effect on performance of changing two parameters: reducing the resolution time of production delays (Strategy 1), and increasing the probability of finding alternative sourcing (Strategy 2). Strategy 1 considers the possibility that a production delay can be resolved in a time interval [30, 50] instead of [60, 80]. For Strategy 2, another alternative vendor is introduced into the supply chain so that the probability of finding alternative sourcing is increased to 0.7. The simulation results of these two strategies are also shown in Table 4. Using Strategy 1, although there is a large number of back orders, more than a half of them (77 out of 135) are still delivered through successful rush supply orders from the main vendor (only 34 back orders are replenished by alternative vendors). For the second strategy, 70% of back orders (78 out of 111) are fulfilled by

alternative vendors. Both strategies lead to an increase in the fill rate of customer orders. Thus, compared with the baseline strategy, Strategy 1 and Strategy 2 can increase the fill rate to 95%. Similarly, other scenarios can be explored and analyzed in detail with this technique.

7. Comparison with Related Work

Related research for detailed modeling of supply chains is still limited. Active databases rely on event-condition-action (ECA) rules [18]. Such rules make databases "active" by allowing them to react to events, i.e., when an event occurs, if some conditions hold, an action (such as database update, insert, query) is taken. However, the drawback of ECA rules is that they cannot do event chaining in a natural way, and hence cannot easily facilitate the analysis of cause-effect relationships between events. Moreover, they are also unable to trace back the causes of events, or forecast future events. Finally, temporal attributes cannot be modeled explicitly.

One domain in which event management has been studied with considerable interest and success is the area of network management. Here the objective is to manage large number of low-level events that may be related and to extract high-level events that require management attention while ignoring the unimportant ones. Hasan et al [10] provide a conceptual framework for describing causal and temporal relationships between network events. In [9], Gruschke give a dependency graph based algorithm for event correlation in networks. This algorithm is used to map raw events in the network to faulty objects based on the links in the graph. These approaches are relevant in supply chains also, but they lack a precise representation of temporal constraints. In [6], Time Petri nets are integrated into databases and used for semantic mapping of events in computer networks. The transitions are associated with guard conditions expressed as database constraints. It is an interesting approach with possible applications in supply chains, but harder to implement and verify. In particular, there is no standard approach to transform an event rule to a Petri net and time constraints are captured in an ad-hoc way. Case-based approaches for event correlation in networks are given in [14]. These methods compare a new case against a database of cases and look for stored solutions; however, they require an application-specific model and are computationally complex. Rule-based or knowledge-based approaches are discussed in [8, 28]. Here the knowledge of the expert is described in rules and the rules are applied to diagnose a new problem. However, formal representations of rules are not provided, and hence it is difficult to extend those approaches to other domains.

Other related work includes a proactive SCEM system with agent technology discussed in [5]. While it focuses on event monitoring and alert generation, this system lacks the capability of analyzing events and suggesting solutions. Patterns have been studied in many domains, but the ones developed in the context of workflow management [1, 24] are the closest to our work; however, they do not address the complex temporal constraints. Classical Petri nets have been used to model rules in knowledge bases [15, 20, 29]. However, high-level time colored Petri nets are naturally more expressive because, besides capturing temporal constraints elegantly, they can support a rich vocabulary of event rules, such as sequence operators (*and*, *or*), modifiers (*last*, *nth*, *any*, *none*), and predicates [8]. We have modeled the part of this vocabulary that is relevant in supply chains, such as *and*, *or*, *any*, and *none*, in these seven patterns because our focus is on the most common patterns that arise in supply chains. The other part of the vocabulary consisting of modifiers can also be modeled as a further extension using the concepts of guards or multi-set colors in Petri nets (see [11, 12]).

8. Conclusions

We developed an approach for modeling event relationships in a supply chain through Petri-nets. The formalism consists of seven basic patterns that capture cause-effect relationships in Petri-nets. These patterns can be combined together as building blocks to create other patterns and also more complex Petri-nets. We used a very extensive example to illustrate this approach and showed in detail how dependency graph analysis can be used to determine causal relationships between events in a dynamic supply chain. It should be noted that these relationships are complex and depend upon the exact timing of events. We demonstrated that slight changes in temporal relationships can result in a very different dependency graph and also final outcome.

Petri net simulation offers a mature technique for analyzing the Petri net models, and the easy availability of many Petri net software packages is an asset. We implemented Petri net models using CPN tools and performed sensitivity analysis by simulation. By changing a specific event parameter, such as event resolution time, we can show how supply chain performance is affected. Such scenario analysis can suggest solutions to improve supply chain performance. Therefore, by managing events, we can actually manage supply chain performance. We ran comprehensive simulation experiments illustrated how this approach can help decision makers to improve supply chain performance.

In summary, as supply chains become more tightly integrated across partners, it is becoming increasingly important to respond in real-time to events (also called sense-and-respond capability). We described a novel approach to model event relationships in a supply chain using Petri-net patterns that can be combined to create realistic Petri-net models of supply chains. We further implemented a model in a Petri-net modeling and simulation tool, and ran simulation experiments with it. A unique feature of the approach is that the Petri-nets are constructed from patterns or building blocks which can be composed together and extended to create new user-defined patterns. In future work, we would like to develop more formal verification techniques for the supply chain models, and also develop heuristics for reachability analysis of dependency graphs to predict "interesting" events.

Acknowledgements

The work of the first two authors was supported in part by a grant from IBM.

References

1. Aalst, W.M.P. van der, Hofstede, A.H.M. ter, Kiepuszewski, B., and Barros, A.P. "Workflow Patterns," *Distributed and Parallel Databases*, 14(1):5-51, 2003.
2. Alvarenga, C.A. and Schoenthaler, R.C. "A New Take on Supply Chain Event Management," *Supply Chain Management Review*, March/April, 2003, pp. 29-35.
3. Asgekar, V. "Event Management Graduates with Distinction," *Supply Chain Management Review*, September/October, pp. 15-16, 2003.
4. Berthomieu, B., and Diaz, M. **"**Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Transactions on Software Engineering*, 17(3):259-273, 1991.
5. Bodendorf, F. and Zimmermann, R. "Proactive Supply-Chain Event Management with Agent Technology," *International Journal of Electronic Commerce*, 9(4): 57-89, Summer 2005.
6. Casati, F., Du, W. and Shan, M. "Semantic Mapping of Events," HP Labs Technical, HPL-98-74 980421.
7. Christensen, and S., Hansen, N.D. "Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs". *Application and Theory of Petri Nets 1993*, Marsan, M. A. (ed.), Lecture Notes in Computer Science, 691:(186-205). Springer-Verlag, Berlin, 1993.

8. Gardner, R. and Harle, D. “Pattern discovery and specification translation for alarm correlation,” In *proceedings of Network Operations and Management Symposium (NOMS’98)*, New Orleans, USA, February 1998, IEEE.
9. Gruschke, B. “Integrated Event Management: Event Correlation Using Dependency Graphs”, In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, October 1998.
10. Hasan, M, Sugla, B., and Viswanathan, R. “A conceptual framework for network management event correlation and filtering systems,” In M. Sloman, S. Mazumdar, and E. Lupu, editors, *Integrated Network Management* VI, pages 233–246, Boston, MA, May 1999.
11. Jensen, K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Volume 1, Springer-Verlag, Berlin Heidelberg, 1996.
12. Jensen, K. “An Introduction to the Practical Use of Coloured Petri Nets,” *Lectures on Petri Nets II: Applications*, Reisig, W and Rozenberg, G (eds.), Lecture Notes in Computer Science, 1492:(237-292), Springer-Verlag 1998.
13. Lee, H., Padmanabhan, V. and Whang, S. “The Bullwhip Effect in Supply Chains,” *Sloan Management Review*(38), 1997, pp.93-102.
14. Lewis, L. “A case–based reasoning approach to the resolution of faults in communication networks,” In *Proceedings of the IFIP TC6/WG6.6 Third International Sysposium on Integrated Network Management*, Hegering, H. G. and Yemini, Y. (eds.) San Francisco, USA, April 1993, pp. 671–682.
15. Liu, N.K and Dillon, T. “An approach towards the verification of expert systems using numerical Petri net,” *International Journal of Intelligent Systems*, Vol. 6, No. 3, pages 255-276. 1991.
16. Luckham, D. *The Power of Events*, Addison-Wesley, Boston, 2002.
17. Marabotti, D. “Information Technology Insights: Supply Chain Event Management Emerges in Enterprise Software,” *Chemical Market Reporter*, 262(9):21-22, September 2002.
18. McCarthy, D.R., and Dayal, U. “The Architecture of an Active Database System,” *Proceedings of ACM SIGMOD Conference on Management of Data*, J. Clifford, B. G. Lindsay, and D. Maier (eds.). ACM Press, New York, 1989, pp. 215-224
19. McCrea, B. “EMS Completes the Visibility Picture,” *Logistics Management*, 44(6):57-61, June 2005.

20. Meseguer, P. “A New Method to Checking Rule Bases for Inconsistency: A Petri Net Approach,” In *Proceedings of the 9th European Conference on Artificial Intelligence* (ECAI-90), Stockholm, August 1990.
21. Montgomery N. and Waheed, R. “Supply Chain Event Management Enables Companies to Take Control of Extended Supply Chains,” Report on European E-Business, AMR Research, September 2001.
22. Murata, T. “Petri Nets: Properties, Analysis and Application,” In *Proceedings of the Institute of Electrical and Electronics Engineers*, 77(4): 541-580, April 1989.
23. Ratzer, V. A., Wells, L., Lassen, M. H, Laursen, M., Qvortrup, F. J., Stissing, S. M., Westergaard, M., Christensen, and S., Jensen, K. “CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets,” *Applications and Theory of Petri Nets 2003*, W. van der Aalst, E. Best (Eds.), Lecture Notes in Computer Science, 2679: (450 - 462), Springer-Verlag GmbH, 2003.
24. Russell, N., Aalst, W.M.P.van der, Hofstede, A.H.M. ter, and Edmond, D. Workflow Resource Patterns: Identification, Representation and Tool Support. In O. Pastor and J. Falcao e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering* (CAiSE'05), volume 3520 of Lecture Notes in Computer Science, pages 216-232. Springer-Verlag, Berlin, 2005.
25. Shimura, T., Lobo, J. and Murata, T. “An Extended Petri Net Model for Normal Logic Programs,” *IEEE Transactions on Knowledge and Data Engineering*, 7(1): 150-162, 1995.
26. Strozniak, P. “Exception Management,” *Frontline Solutions*, 3(8): 16-24, August 2002.
27. Wang, J. *Timed Petri Nets Theory and Application*, Kluwer Academic Publishers, Boston, 1998, pp. 63-123.
28. Wu, P., Bhatnagar, R., L. Epshtein, Shi, Z. Alarm correlation engine (ace). In *Proceedings of the 1998 IEEE Network Operations and Management Symposium (NOMS'98)*, New Orleans, Louisiana, USA, 1998, pages 733-742.
29. Zhang, D and Nguyen, D. “PREPARE: A Tool for Knowledge Base Verification,” *IEEE Transactions on Knowledge and Data Engineering*, 6(6):983-989, 1994
30. Zuberek, W.M. “Timed Petri nets - definitions, properties, and applications,” *Microelectronics and Reliability*, 31(4):627-644, 1991.

Appendix 1: Simulation of Petri net in Figure 18

1. Hierarchical CPN mapping

Here we show how our Petri nets are implemented using CPN Tools. Figure 22 gives a Colored Petri net (CPN or CP-net) mapping from Figure 18. To make this mapping readable, a *hierarchical CPN* is used. In Figure 22, each transition with a small tag, called HS-tag, is a substitution transaction, and it is mapped onto a so-called subpage. This CPN has three levels. The first level is shown in Figure 22. The transaction "rule8&9" here can be expanded as a sub-page as shown in Figure 23 (Level 2). Furthermore, transaction "rule8" in Figure 23 can be substituted by a sub-page as shown in Figure 24.

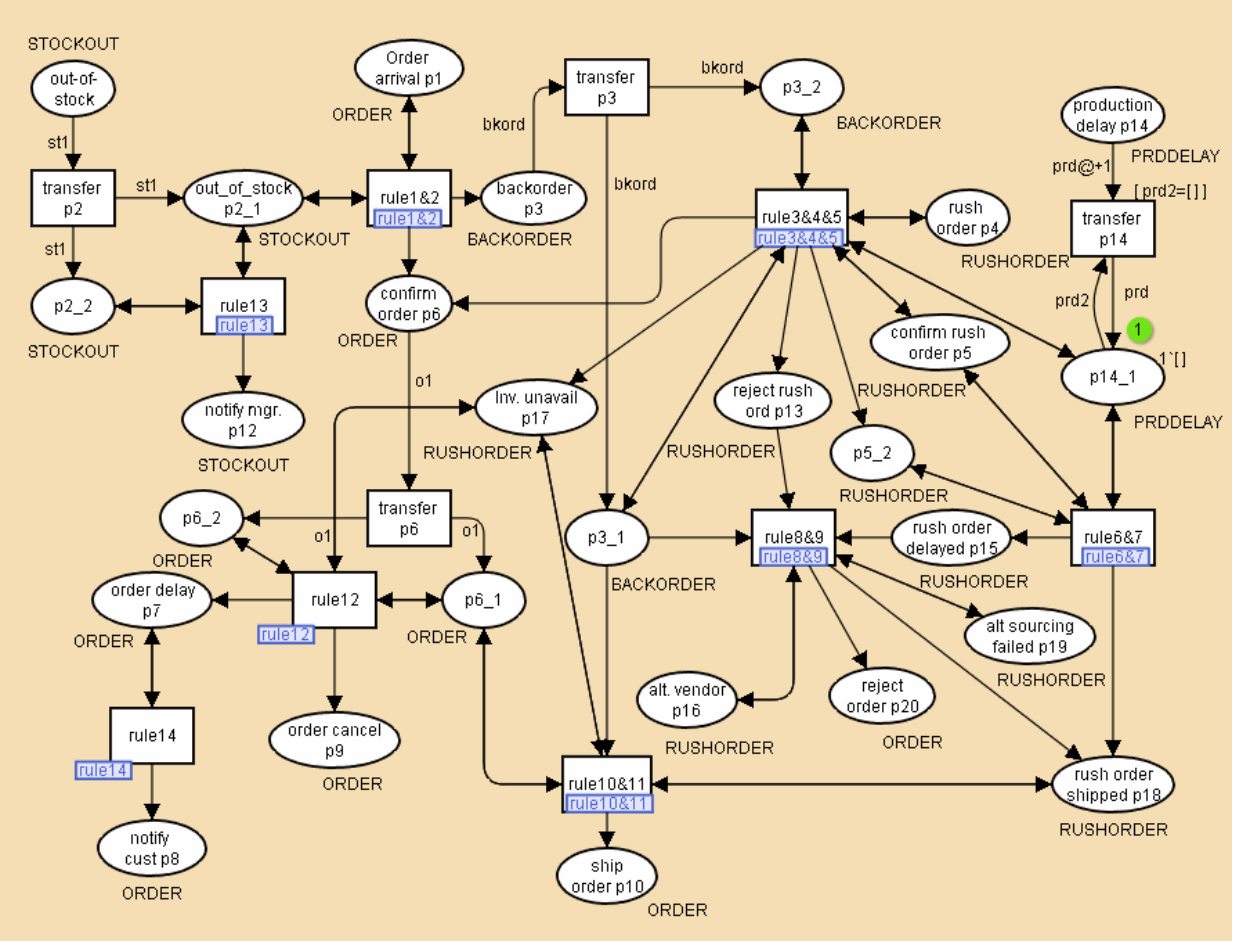


Figure 22: CP-net (Level 1)

2. Implementation of time constraints

CPN tools cannot directly support the subtle time semantics used in this paper, but it can support *timed CP-nets*. In a timed CP-net, a global clock is introduced and a token can carry a time stamp.

The time stamp describes the earliest model time when the token can be used. In addition, after firing a transition, the output tokens can be delayed for a *fixed time*. The details of timed CP-nets can be found in [12].

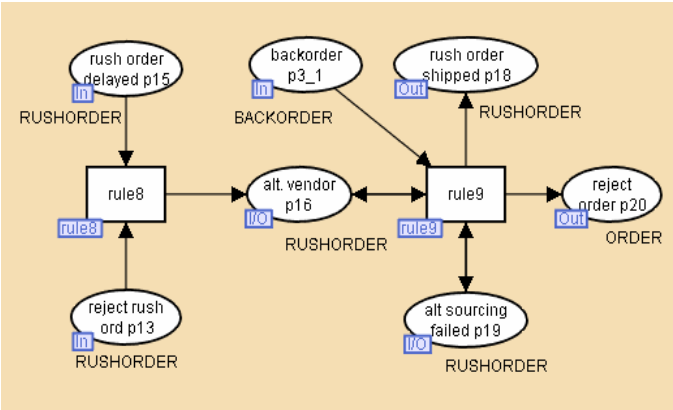


Figure 23: Subpage for "rule8&9" (Level 2)

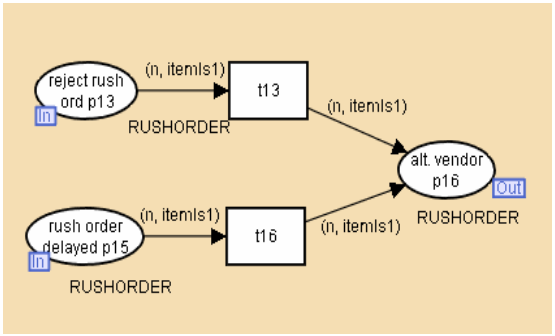


Figure 24: Subpage for "rule8" (Level 3)

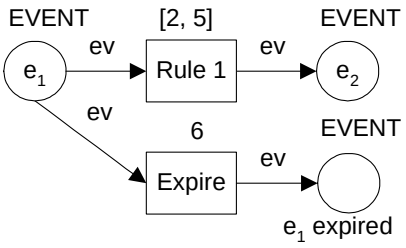


Figure 25: Time Petri net

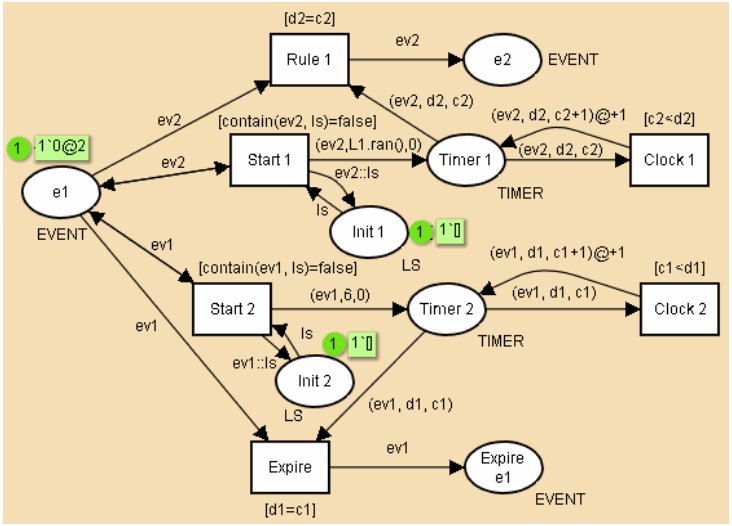


Figure 26: Implementation with CPN Tools

Since CPN tools are not designed for handling time intervals attached to transitions, we had to improvise and come up with a general approach for doing so. In mapping a time Petri net, a "clock" was introduced with each transition that has an associated time interval. Whenever this transition is enabled, the clock starts and then moves independently until timeout. Figure 26 is the mapping of the *time* Petri net of Figure 25, and shows a *timed* CPN with two "clocks". In Figure 26, transition "Start 1" fires whenever there is a token in *e1*. According to Figure 25, Rule 1 should fire in the [2,5] time interval *after* it is enabled. This interval is modeled by the L1.ran() function, which generates a random number between 2 and 5. Therefore, "Timer 1" has an initial

token with a color showing the ID of event *e1*, allowed waiting time, and actual waiting time 0. Then transition "Clock 1" fires. For each firing, the actual waiting time is increased by 1 and this token will be delayed for 1 time unit. Transition "Clock 1" fires until the allowed waiting time is reached. This is controlled by a *guard* "c2 < d2" placed on this transition. At that time, if the transition "Rule 1" is still enabled, it fires. The other "clock" controls the expiration of *e1*.

3. Implementation of inhibitor arcs

We can substitute inhibitor arcs with an equivalent structure as discussed in [7]. Figure 28 shows an equivalent Petri net of Rule 1 (see Figure 27). In Figure 28, each order arrival into *p1* sends an empty list to the place "stockout list". Then transition "makelist' fires if there is any token in place *p2* (i.e., out-of-stock event) and this list is appended with any item which is currently out-of-stock. In order to ensure this list is completely generated before it is actually used by Rule 1, Rule 1 always fires 1 time unit later than transition "makelist". This delay is achieved by the place "delay", where a token arrives one time unit after transition "start" fires. Later, this list is used to control the firing of Rule 1. Two functions are used in the guard conditions of the transitions. Function *contain(item1, itemls1)* checks whether *item1* exists in list *itemls1* to prevent duplicates. Function *containls(itemls2, itemls1)* returns true if any item of list *itemls1* is contained in list *itemls2*. Therefore, transition "rule 1" fires only if none of the ordered items is in the stockout list.

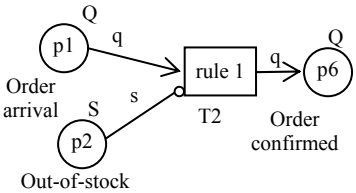


Figure 27: Rule 1 with an inhibitor arc

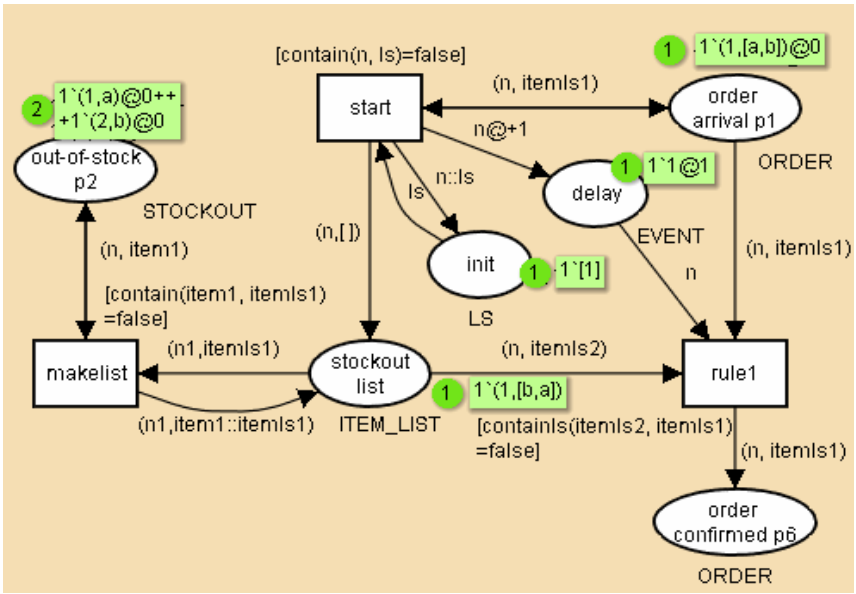


Figure 28: CPN implementation of Rule 1