A Formal Modeling Approach for Supply Chain Event Management

Emily (Rong) Liu and Akhil Kumar
Smeal College of Business
Penn State University
University Park, PA 16802, USA
{rull110,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 time Petri nets 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. Next, we show how to combine these patterns to build a complete Petri net and analyze it using reachability analysis, dependency graphs and simulation.

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

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 [4]. 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 delayed etc., and they are costly. Moreover, events tend to propagate in collaborative supply chains across partners, resulting in the well-known bullwhip effect [8]. Such risks have given rise to the new field 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" [3]. 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 [10]. 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. Through this approach, we can detect events, perform cause-effect analysis, forecast their consequences and prioritize them. More importantly, this approach can be used to create an event engine that can monitor the status of a supply chain and intelligently react to events.

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 can come from the following three sources: (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" or "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 events* as opposed to *composite events*. Composite events are *derived* from simple events by *event aggregation*. A composite event *A* is deduced when a group of simple events occurs [9]. 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 regulations [9]. *Event aggregation* is a mechanism to filter simple events and extract meaningful information from them by setting up alarms in advance and reacting to possible crises.

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, we need a rule to express the idea that the event "order delayed by *T*" is the 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 arises 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.

3. Petri nets Preliminaries

A Petri net is a directed graph consisting 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 [11]:

(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 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 will come later.

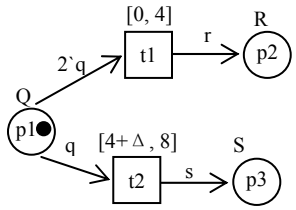


Figure 1: Colored time Petri net

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) [12]. If $I_1 = I_2$, we just associate one time value with each transition. If the interval is not specified, we assume $I_1 = I_2 = 0$. Moreover, tokens can be tagged with data values (or a color) to create a colored Petri net (CPN) [7]. For example, we use tokens of different colors (or values) for each order or product. In a CPN 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. Analysis techniques for TPNs are discussed in [1,5,12].

4. Event patterns to model Supply Chain Rules

We turn now to develop 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. Therefore, it is quite natural to model events as places of a Petri net. Thus, the terms *events* and *places* are used interchangeably. Moreover, time Petri nets offer an attractive choice for modeling the dynamic aspect in supply chains. Next we discuss the seven patterns mentioned before.

Pattern 1 (***cause-result pattern)***: A *simple 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].

Example 1: If an order is delayed (e_1), contact customer (e_2) before time *T1*.

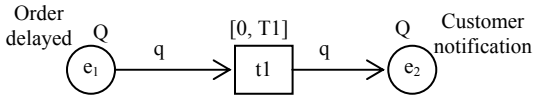


Figure 2: Petri net of Example 1

Figure 2 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. 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.

Example 2: If product *s* is out of stock (e_1) more than once within period *T2*, contact supply chain manager (e_2). (Note, *s* is the product ID)

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 3 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.

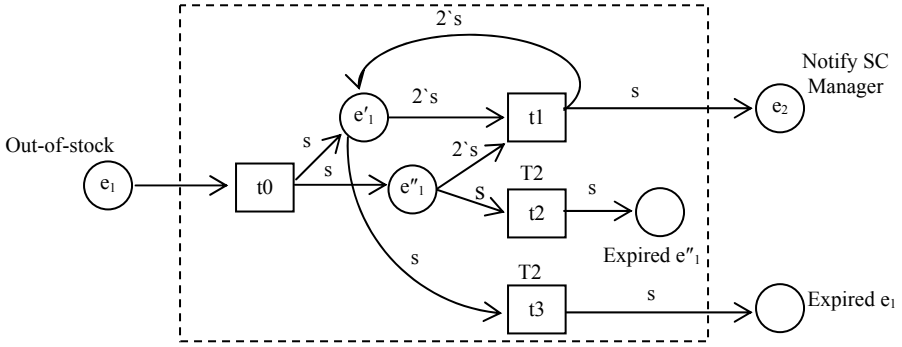


Figure 3: Petri net of Example 2

Pattern 3 (**Inclusive choice)**: The need for this construct arises when multiple, alternative events can occur based on temporal conditions. Example 3 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); 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*.)

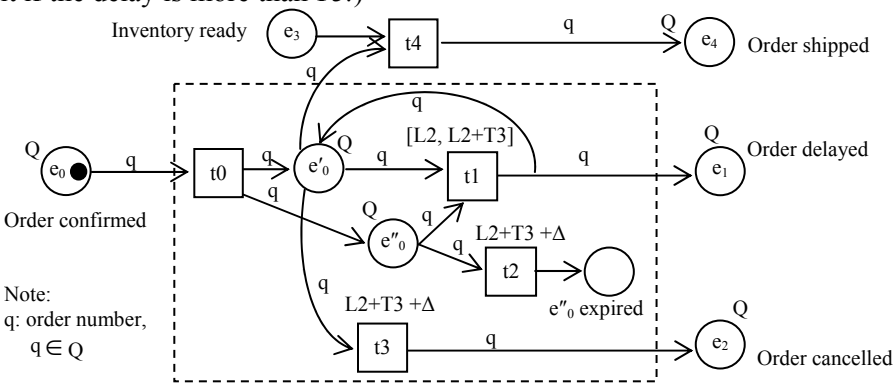


Figure 4: Petri net model of an order process

When an order is confirmed (see Figure 4), a token is placed in place e'_0 and e''_0 as well. Transitions *t1, t2,* and *t3* are enabled but do not fire at that moment. If this token is consumed by the shipment transition *t4* before time *L2* (relative to its arrival), transitions *t1* and *t3* are disabled, but transition *t2* will fire at time *L2+T3+Δ* 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 *t3*) may use it later. If there is still a token in e'_0 after *L2+T3*, transition *t3* 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 order cancelled, or only order delayed, depending upon the temporal relationships. One can see this rule actually has complex semantics, yet its Petri net model offers a relatively simple way for describing such temporal relationships.

Other patterns: Four other patterns are summarized, each with a brief description and an example, in Table 1. They represent other possibilities for modeling event relationships in supply chains.

5. Analysis, Discussion and Conclusions

We have developed an approach for modeling event relationships in a supply chain through Petri-nets. The formalism consists of 7 basic patterns that capture cause-effect relationships in Petri-nets, where the places or circles represent events and the transitions or boxes represent the (possibly delayed) effect of the events. The delay is captured by the time intervals on the transition boxes. Although these patterns are not exhaustive, they are sufficient for most situations and this formal approach, based on Petri-nets, allows new patterns to be constructed when necessary. Moreover, these patterns can be combined together as building blocks to create more complex Petri-nets. As real world events occur, tokens are placed to represent them. Then, we can use Petri-net algorithms for reachability analysis to predict the likely consequences of these events and build cause-effect dependency graphs. In one supply chain model created with 14 patterns, we were able to model many intricate interactions between different orders. For instance, product *A* was out of stock with the distributor and a rush supply order was issued, but this rush supply order was rejected by the first alternative vendor because of a production delay. Then another vendor was contacted; nevertheless, order *O1* became too late, and was eventually cancelled. However, order *O2* shipped on time. By modifying the scenario slightly, it was possible to ship both orders on time. Moreover, by focusing the analysis on the exception orders only, the complexity can be controlled.

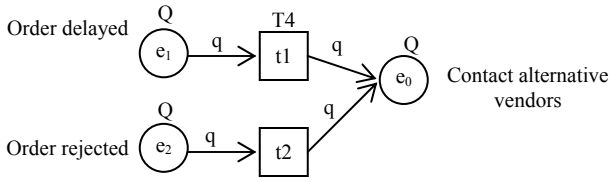
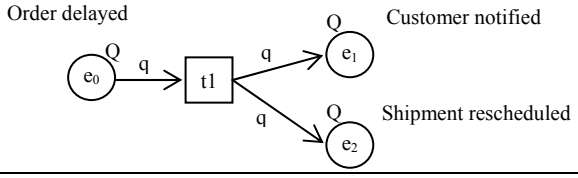
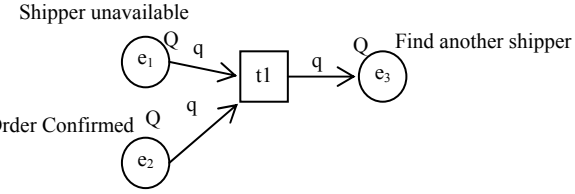
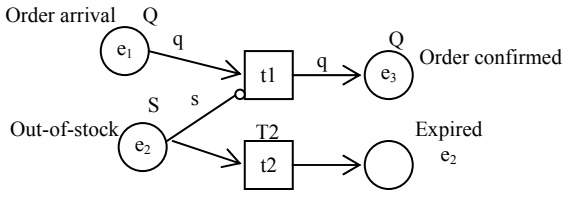
Pattern name	Example
Pattern 4: 1-of-N causes – single effect: Alternative causes produce one effect. Here, if the order is delayed by more than *T4* **or** rejected by one vendor, then alternative vendors are contacted.	
Pattern 5: 1 cause – N results: One cause can lead to multiple concurrent effects. If the order is delayed, then the customer is notified **and** the shipment rescheduled.	
Pattern 6: N causes – 1 result: Multiple causes in conjunction can produce one result. If the shipper of a confirmed order (e_2) is not available (e_1), find another shipper (e_3).	
Pattern 7: Non-occurrence of an event: A result is produced if an event does **not** occur. In this example, if the out-of-stock event does not occur (i.e., there is no token in e_2), then *t1* can fire upon an order arrival. Notice the *inhibitor* arc from e_2 to *t1*. e_2 expires after a time *T2*.	

Table 1: Patterns 4, 5, 6 and 7 (*name, description* and *example* for each pattern)

Petri net simulation offers another mature technique for analyzing the Petri net models. There are many available simulation software packages that facilitate the use of simulation for decision-making. By adjusting time intervals associated with transitions and other parameter values, it is possible to perform various types of scenario analyses. Moreover, we can simulate the effect of proposed changes in supply chains by adding new events or event rules.

Related research for detailed modeling of supply chains is still limited. In Casati, et al. [6], Time Petri nets are integrated into databases and used for semantic mapping of events in the computer networks domain. 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. Other approaches are discussed in [3, 9]. In addition, patterns have been studied systematically in the context of workflows [2]. These workflow patterns are somewhat similar to supply chain event patterns, but they do not address the complex temporal constraints involved in supply chain event patterns.

In summary, as supply chains become more tightly integrated across partners, it is becoming increasingly important to respond in real-time to events (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 a complete Petri-net. The Petri-net can be verified for correctness and algorithms can be used to perform cause effect analysis.

References

1. Aalst, W.M.P. van der. "Interval Timed Coloured Petri Nets and their Analysis," *Application and Theory of Petri Nets 1993*, Marsan, M. A. (ed.), Lecture Notes in Computer Science, 691:(453-472). Springer-Verlag, Berlin, 1993.
2. 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.
3. 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.
4. Asgekar, V. "Event Management Graduates with Distinction," *Supply Chain Management Review*, September/October, 2003, pp. 15-16.
5. 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.
6. Casati, F., Du, W. and Shan, M. "Semantic Mapping of Events," HP Labs Technical, HPL-98-74 980421.
7. Jensen, K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Volume 1, Springer-Verlag, Berlin Heidelberg, 1992, pp. 1-55.
8. Lee, H., Padmanabhan, V. and Whang, S. "The Bullwhip Effect in Supply Chains," *Sloan Management Review*(38), 1997, pp.93-102.
9. Luckham, D. *The Power of Events*, Addison-Wesley, Boston, 2002.
10. 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.
11. Murata, T. "Petri Nets: Properties, Analysis and Application," In *Proceedings of the Institute of Electrical and Electronics Engineers*, 77(4): 541-580, April 1989.
12. Wang, J. *Timed Petri Nets Theory and Application*, Kluwer Academic Publishers, Boston, 1998, pp. 63-123.