Abstract—Recently, online shopping integrating third-party payment platforms (TPPs) introduces new security challenges due to complex interactions between Application Programming Interfaces (APIs) of Merchants and TPPs. Malicious clients may exploit security vulnerabilities by calling APIs in an arbitrary order or playing various roles. To deal with the security issue in the early stages of system development, this paper presents a formal method for modeling and verification of online shopping business processes with malicious behavior patterns considered based on Petri nets. We propose a formal model called E-commerce Business Process Net to model a normal online shopping business process that represent intended functions, and malicious behavior patterns representing a potential attack that violates the security goals at the requirement analysis phase. Then, we synthesize the normal business process and malicious behavior patterns by an incremental modeling method. According to the synthetic model, we analyze whether an online shopping business process is resistant to the known malicious behavior patterns. As a result, our approach can make the software design provably secured from the malicious attacks at process design time and, thus, reduces the difficulty and cost of modification for imperfect systems at the release phase. We demonstrate our approach through a case study.

Note to Practitioners—Online shopping with a third-party payment platform has rapidly evolved worldwide. New security issues have arisen due to the complex interactions among APIs, which can be illegally used by malicious users. Such use often results in loss of interests of legitimate users. The issue has become more and more serious these years. This work presents a formal method to deal with such problems. It adopts a novel formal model based on Petri nets. In the conceptual design stage, given the specifications of an online shopping business process and malicious behavior patterns, it can help designers analyze whether a business process is resistant to the malicious behavior patterns. Thus, the software design can be provably secured at process design time, and the difficulty and cost to modify imperfect systems are alleviated at the system release phase. The proposed method can be readily used in the software design of industrial online shopping business processes.

Index Terms—Business process, e-commerce, online shopping, software design, trustworthiness, verification.

I. INTRODUCTION

ONLINE SHOPPING with a third-party payment platform (TPP) has become the new frontier for doing business nowadays, and become increasingly popular in the global economy as more and more business transactions are conducted over the web. Its daily volume is sizable and continues to grow at a rapid pace [1]. It can be successful only if the general public trusts online trading systems. Even if the volume of transactions and the number of users are growing constantly, it appears that many users have not accepted online shopping as the main trading channel. Research has shown that insufficient trust represents a key reason for users to avoid making businesses over the Internet [2], [3]. However, online shopping systems are complex and difficult to be correctly designed. Design-level vulnerabilities are indeed a major source of security issues [4], [5]. For example, in Microsoft’s “security push,” about 50% of the security issues had been detected due to design-level flaws [4]. As a distributed application on the web, online shopping business processes are more complex and loosely coupled. Recently, online shopping systems have increasingly integrated TPPs such that a complete online shopping business process has three parties: Shopper, Merchant, and TPP. Their respective business processes construct the entire online shopping process. This integration introduces new security challenges due to the complexity for an application to coordinate its internal states with those of component services and web clients across the Internet [2]. The complex linkages of control and data flows in online shopping business processes may produce very serious problems, e.g., the violation of transaction properties and huge losses of users, and the loose coupling of the different parties results in the lack of mutual understanding among their business processes. Consequently, malicious users can call Application Programming Interfaces (APIs) in an arbitrary order by some special means, and even play several roles to achieve their malicious attack purposes.
As the new security challenges of online shopping business processes are at the application-level [6], the sufficient protection of online shopping systems from attacks is beyond the capabilities of network-level and operating system-level security approaches, e.g., cryptography, firewall, and intrusion detection. They lack knowledge of application semantics and cannot meet the needs from today’s distributed online shopping systems. Engineering software security is essential, and it is important to incorporate the use of assurance techniques throughout development and operation [7]. Thus, one needs to exploit the methodologies to verify the trustworthiness of online shopping business processes. Nevertheless, rigorous and formal methodologies for online shopping business process design remain unavailable. The most pressing challenge is how to verify the trustworthiness of an online shopping process in the conceptual modeling phase such that potential security issues can be addressed before the implementation of an online shopping system. At the requirement analysis and design levels, we need to explicitly identify whether the online shopping systems can be resistant to the possibly malicious behavior patterns. Note that such patterns can be found in many public threat libraries [2], [4], [8], [9].

We focus on the online shopping business process that consists of three parties: Shopper, Merchant and TPP, and verifies it by formal methods at the conceptual modeling phase from the application-level viewpoint. The basic idea is: initially, to construct the functional model according to design specifications; then, to choose one malicious behavior pattern and translate it to a malicious behavior model according to the functional model; next, to synthesize them for establishing an online shopping business process able to handle such malicious behavior scenario; at last, verify it and determine whether the online shopping business process can withstand such an malicious behavior pattern. The framework is shown in Fig. 1. Note that we need to verify the designed system subject to all malicious behavior patterns stored in the library according to the above process. After a malicious behavior pattern has been verified, choose another from the library and repeat the process until all are verified.

This work concentrates on the following respects:

a) E-commerce Business Process Net (EBPN). We extend and modify a traditional Petri net to an EPN by integrating both data and control flows to reflect the data and state information. Thus, data errors and nondeterminacy of data states during a trading process can be easily described with it. Data information is added, and data states are proposed to reflect the changes of transaction states.

b) Modeling methods of a functional model and malicious behavior model. Based on EPN features, we propose the methods to build up a functional model of an online shopping business process and those of malicious behavior patterns, and then, synthesize them to obtain a complete malicious behavior scenario.

c) Formal verification methods. First, we obtain the malicious behavior from the viewpoint of malicious clients according to the malicious behavior model. Next, we analyze the composed EPN with a malicious behavior sequence, and derive the relation graph of the malicious behavior sequence and legal transitions. At last, by using EPN’s dynamic properties, we determine whether an online shopping business process can withstand malicious behavior patterns.

Using the proposed methodology, designers can identify problems early in a design process and correct them before the system realization, and avoid losses caused by their solution procedure. Thus, one is able to generate more reliable systems faster and at lower costs with the proposed method.

The next section describes the related work. Section III introduces the basic concepts. Section IV describes how to model an online shopping business process and malicious behavior patterns using EPN. Section V presents the verification methods. Section VI gives a case study. Section VII concludes this paper.

II. RELATED WORKS

This study is primarily related to the work on the design and analysis of e-commerce systems and secure software systems, including e-commerce protocols and formal methods for software design. Formal methods are mathematical techniques for specifying and verifying correctness and trustworthiness of software systems. The U.S. Department of Defense Trusted Computer System Evaluation criteria require that the highest level of security classification (the A-class) use formal specification and verification techniques [10].

About trustworthy e-commerce systems, Tygar and Kalam have pinpointed the need for a systematic treatment of the correctness properties required in digital payment systems [11], [12]. Atomicity is one of the properties (Atomicity, Consistency, Isolation and Durability—known as ACID properties) of modern transactional information systems. Some solutions have been proposed to improve the performance of e-commerce services based on protocols, and many use a Communicating Sequential Process (CSP) system description and perform verification via the Failure Divergence Refinement (FDR) model checker [13]–[16]. Other similar methods also contain the model checking methods of e-commerce systems [17], [18]. Katsaros proposes a method to build and validate NetBill protocol models based on colored Petri nets [18]. Most of the
above researches concentrate on protocols of e-commerce and model checking of such transaction properties as atomicity, but the validations of protocols are not sufficient to ensure the integrity and reliability of e-commerce systems because there are still many defects and logic errors at the design level of business processes, which can be exploited by malicious clients. Business processes belong to the application level, and is a rather important part in an online shopping system. They include the business scenarios and applications. Many malicious behaviors in online shopping systems are exploited in business processes [2].

Additionally, the method of model checking cannot be understood easily by system designers and evaluators as it lacks an intuitive method of graphical modeling [19]. Petri nets are a suitable tool to illustrate true concurrency and model distributed systems, and important work has been done on supervisory control theory [20]–[28]. Supervisory control is a technology in which a controller is designed for a given physical system in order to observe the system behaviors and control it to achieve given specifications. It intends to resolve the issues related to non-blocking [23], [24], fault-tolerance [25], deadlock and liveness of the physical system [26]–[28]. It has gained various applications in such areas as flexible manufacturing systems and embedded systems.

Petri net-based approaches [29], [30] have been presented to model and verify correctness and soundness of workflows. A series of valuable work has been done on cooperative systems and interorganizational workflow based on Petri nets [19], [31], [32]. Some innovative works have been done on Petri net-based analysis and composition of web services [33]–[35]. However, most of the existing work concentrates on the soundness and correctness of workflow, cooperative systems, and composition of web services, but fail to consider malicious behaviors related to the financial security issues. These security issues may result in the financial loss of legitimate users.

Today’s businesses are inherently process-driven, and the security of business processes is increasingly important [36]–[39]. In [36], at the source code level, a method for statically checking the security and conformance of the system implementation is proposed. In order to rapidly implement new processes, research on the compliance of cross-organizational processes and their changes is performed [37]. Most of them focus on the security properties like Access Control and Confidential Information [36], [38] in enterprise business processes, and ensure the security of secret and sensible information that cannot be leaked to other parties. Other related studies refer to the process consistence in complex business processes [37], [39]. They are proposed to deal with the inconsistencies among business processes of different departments in a cross-organizational process, or the inconsistencies between real process executions and their designed model, and guarantee the consistence of these business processes when some have changed.

However, online shopping systems have their own security properties such as Atomicity [12] and Payment Completion Invariant [2]. Hybrid web applications that combine the APIs of multiple web services into integrated services like online shopping websites have rapidly developed, and caused new security concerns. The web programming paradigm is already under threat from malicious web clients who exploit logic flaws caused by improper distribution of the application functionality between a client and server [2]. Even if the security requirements of enterprise business processes are met [36]–[39], an online shopping business process may not be flawless, and malicious users can obtain additional benefits through a series of actions. Many accidents of existing online shopping systems are caused by data errors and state inconsistency as exploited by malicious users. Thus, both data properties and data state nondeterminacy must be depicted. Additionally, a formal model is needed to accurately depict the mainstream online shopping business processes consisting of data flow, control flow and three parties: Shopper, Merchant, and TPP.

The security analysis from the adversary’s perspective has been increasingly important in protocols, intrusion detection systems, and security testing [40]–[44]. At the requirement analysis and design levels, one can identify how the software can be attacked by malicious users. According to this idea, misuse or abuse cases [41], [42] and threat modeling [40] are studied. The threat driven system design derives system models from use and misuse cases, and evaluates whether they could mitigate the misuse threats. The threat modeling approach provides a structured way to design secure software systems, but due to the informal nature, most of the current threat modeling approach [8], [40], [43] does not support the verification of threat models. Based on Petri nets, a threat driven modeling and verification method specifies security threats according to misuse and anomalies of intended functions and handles security features for mitigating threats in an aspect-oriented paradigm [45]. In [46], a methodology has been proposed for utilizing threat modeling in building secure software architectures using Software Architecture Modeling (SAM) framework and verifying them formally using symbolic model checking. The above studies illustrate that security analysis from the adversary’s perspective has a great potential in software design. Nevertheless, sufficient formal methodologies for modeling and verification of malicious behavior patterns in online shopping systems remain to be seen. Without formal verification, it would be difficult to ensure that a system design is immune to some identified security threats. Existing formal methods tend to rely heavily on the formalization and verification of operating systems and general software systems, and should be significantly extended to the popular distributed online shopping systems including three parties: Shopper, Merchant, and TPP. Data errors, data states nondeterminacy, and complex malicious behaviors must be depicted by using formal methods. However, the existing formal models are insufficient to do so well.

Colored Petri nets (CPN) are a powerful tool for modeling concurrent systems, and it is a combination of Petri nets and programming languages according to [47]. The initial marking is some specific values of actual systems and every trigger of transitions needs to bind some specific values. However, it is hard to list all possible input values of an actual system. Thus, we need a model with a higher level abstraction to both reflect the data properties and cover as many actual situations as possible. The nondeterminacy of data states resulting from human factors such as data tampering in actual electronic trading systems cannot be depicted by CPN to the best knowledge of the authors.
We focus on the complex malicious behavior patterns that can be used to identity theft and illegitimate behaviors caused by out-of-order calls of the APIs to advance the state-of-the-art.

III. BASIC CONCEPTS

Petri nets are a graphical language for modeling and validating concurrent and distributed systems, and allow not only true concurrency but also interleaving. They have two types of nodes called places and transitions. Arcs are from places to transitions and vice versa. A state of a system is represented by a marking, and tokens are shown graphically by small black dots added to the places. The changes of the markings represent the dynamic behavior of the system simulated by a Petri net, which result from the firing of the transitions. The basic concepts of Petri nets are summarized in [48] and [49]. In order to describe the electronic trading process better, we modify and extend them with some new functions.

Definition 3.1: An E-commerce Business Process Net (EBPN) is a 7-tuple $EN = (P, T; F, D, W, S, G)$, where:

1) $P$ is a finite set of places;
2) $T$ is a finite set of transitions $T^i$ such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$;
3) $F \subseteq P \times T \cup T \times P$ is a set of directed arcs;
4) $D$ is a finite set of symbols with types denoting the types of tokens;
5) $W: F' \rightarrow \{a_1d_1, a_2d_2, a_3d_3, \ldots, a_ld_l\}, a \in \{0,1\}, d_i \in D,$ and $l > 0$ is the number of elements in $D$;
6) $S \subseteq D$ is a set of key token types; and
7) $G: T \rightarrow H$ is a predicate function that assigns a predicate to each transition $t \in T$, where $H$ is the set of logical expressions on $D$.

In Definition 3.1, an EBPN is a 7-tuple $EN = (P, T; F, D, W, S, G)$, in which $D, W, S,$ and $G$ are newly added to the traditional net definition, and $(P, T; F)$ is the basic structure of a traditional Petri net. EBPN has places, arcs, and transitions. An EBPN is a formal model used to portray an online shopping business process. It is a variant of traditional Petri nets. $T$ is used to depict APIs and operation events of an online shopping process that represents the process from the beginning to end of an electronic trading collaborated by all parties involved. An operation event means a client operation of users such as placing an order or clicking the payment button. $P$ is used to depict data channels which are the transmission channels of trading parameters among different parties that participate in online shopping processes. $D$ is the set of sets of tokens that depict the trading parameters used in an online shopping process such as “orderID,” “Price,” and “Status,” which are passed and exchanged among different parties. There are some key trading parameters in an online shopping business process. They are important ones related to the financial security such as “orderID” and “gross,” which are likely tampered at the client side. Other parameters like “status” and “result” that represent the states of an electronic trading are unimportant trading parameters, which are usually safe at the server side. Correspondingly, EBPN has some key token types denoted by $S$, and it is a subset of $D$. $S$ is used to depict the key trading parameters. The tokens belonging to such types have three fixed values, i.e., $T$ (true), $F$ (false) or $\emptyset$. $\emptyset$ means that the malicious users usually set the trading parameters as null values in the cases studied in [2]. In this paper, we also use $d_k \in D$ as a token with the type of $d_k$ in order to facilitate the expression. The weight function $W$ assigns a $k$-dimensional vector to each arc. Predicates are assigned to some transitions that are used to judge whether a validation result is true or false in a trading process.

For example, Fig. 2 (a) is a schematic example of EBPN depicting the paying operation of TPP. The operation needs two inputs, one of which is the current state of TPP, i.e., $TL\text{Listen}$, and the other one is the trading parameters including $\text{orderID}$ and $\text{gross}$. After the paying operation is finished, two data items are generated, one of which is $TP\text{Paid}$ representing that the money has been paid, and the other one is a transaction number, i.e., $\text{transactionID}$. Note that a predicate $\text{orderID} = T \land \text{gross} = T$ is assigned to $t_1$. The example in Fig. 2 is used to illustrate the new formal model-EBPN. It explains Definition 3.1 intuitively in the graphical form, and includes the characteristics of EBPN.

The above definition describes the static properties of an EBPN. Next, we define dynamic properties such as a marking function and data function.

Definition 3.2: A marking of an EBPN $EN = (P, T; F, D, W, S, G)$ is $M: P \rightarrow \{n_1d_1, n_2d_2, n_3d_3, \ldots, n_kd_k\}, n_k \in N, d_k \in D,$ and $k > 0$ is the number of trading parameters in $D$. If $p \in P$, then the multiset [46] of $k$-dimensional vector $M(p)$ is represented by $M_n(p)$.

A marking $M_n$ of an EBPN assigns $k$-dimensional vectors to places. The vector’s component $n_kd_k$ means that a place has $n_k$ tokens belonging to type $d_k$. Here, a token is a trading parameter that belongs to some type in an EBPN. For example, the marking of Fig. 2 (a) is $M = \{< T\text{Listen}, 0, 0, 0, 0, 0 >, < 0, 0, \text{orderID}, \text{gross}, 0, 0 >, < 0, 0, 0 >, < 0, 0, 0 >, < 0, 0, 0 >\}$. This means that $p_1$ has a token whose type is $\text{TL\text{Listen}}$ representing a state of TPP, and $p_2$ has three tokens representing the trading parameters, two of which are the type of $\text{orderID}$, and the other one is $\text{gross}$.

In this paper, for simplicity, the expression of a marking is $M = p_1(\lambda)p_1$ is the place that has tokens, and $\lambda = M(p_1)$. For example, the marking of Fig. 2 (a) is $M = \{p_1(\text{TL\text{Listen}}), p_2(\text{orderID}, \text{gross})\}$. $M(p_2) = \{\text{orderID}, \text{orderID}, \text{gross}\}$, and $M(p_1) = \{\text{TL\text{Listen}}\}$.

In an EBPN, every arc has a vector. If $p \in P$, and $t \in T$, then the weight of an arc $(p, t)$ or $(t, p)$ is represented by $W(p, t)$ or $W(t, p)$, and the trading parameter set of $k$-dimensional vector $W(p, t)$ or $W(t, p)$ is represented...
by \( \overline{W}(p, t) \) or \( \overline{W}(t, p) \). Satisfying \( W(p, t) \) is a requirement of enabling and firing \( t \) at the current marking. \( W(t, p) \) indicates what the output data is in \( p \) after firing \( t \). In Fig. 2(a), 
\[
\begin{align*}
W(p_1, t_1) = & \langle TListen, 0, 0, 0, 0 \rangle > \text{ means that firing } t_1 \text{ requires that } p_1 \text{ must have at least one token whose type is } TListen; \\
W(p_2, t_1) = & \langle 0, orderID, gross, 0, 0 \rangle > \text{ means that firing } t_1 \text{ requires that } p_2 \text{ must have at least two tokens whose types are, respectively, } orderID \text{ and } gross, \text{ and the tokens in } p_1 \text{ and } p_2 \text{ satisfy these conditions: } \\
W(t_1, p_3) = & \langle 0, 0, 0, TPaid, 0 \rangle > \text{ means that firing } t_1 \text{ deposits a token with the type of } TPaid \text{ to } p_3, \text{ and } \\
W(t_1, p_4) = & \langle 0, 0, 0, transactionID \rangle > \text{ means that firing } t_1 \text{ deposits a token with the type of } transactionID \text{ to } p_4.
\end{align*}
\]

In order to facilitate graphical expression, we simplify the vector \( \langle a_1 d_1, a_2 d_2, a_3 d_3, \ldots, a_k d_k \rangle \) as a set on the arc. For example, Fig. 2(a) and (b) represent the same transition of a paying operation in TPP. This is purely for graphical clarity, because an EBPN may have dozens of trading parameters, and the \( k \)-dimensional vector would be so long that it is impossible to represent it in a graph. In an EBPN, solid arcs represent the control flow depicting control structures and state transition relations, and dashed arcs represent the data flow among APIs. Two flows may overlap.

### Definition 3.3: A pair \( \Lambda = (M, \delta_D) \) is a data state of EN, if \( M \) is a marking of \( EN \), and \( \delta_D \) is a data allocation, where \( \delta_D \) assigns a value \( \text{T} \) (true), \( \text{F} \) (false) or \( \emptyset \) to each \( d \in \{M(p) : p \in P\} \) such that \( d \in (D - S) \rightarrow \delta_D(d) = \text{T} \), and \( \delta_D : S \rightarrow \{\text{T, F, } \emptyset\} \).

Hence, a data state can reflect both the current trading state and processed data information. \( \delta_D \) assigns a value \( \text{T} \) (true), \( \text{F} \) (false) or \( \emptyset \) to each token with some type in the current marking. \( \delta_D \) changes according to the change of markings. For trading parameters that are not related, their values are always \( \text{T} \). For some key trading parameters that are related, their values may be \( \text{T, F} \) or \( \emptyset \) during the implementation process of an electronic trading system.

In this work, if \( d \in \{\overline{M}(p) : p \in P\} \land S \) at a data state \( (M, \delta_D) \), and \( \delta_D(d) = \text{F} \), then we use notation “\( \delta_D(d) \)” to express its value. Likewise, if \( \delta_D(d) = \emptyset \), then we use notation “\( \delta_D(d) \)” to express its value. Otherwise, if \( d \in D \), and \( \delta_D(d) = \text{T} \), then its value would not be displayed for clarity. In Fig. 1, \( \delta_D \) and \( transactionID \) are two key trading parameters, and the data state is 
\[
\{p_1(TListen), p_2(orderID, gross)\} = \{p_1(TListen), p_2(orderID, orderID, gross)\}.
\]

**Definition 3.4:** \( \delta_G \) is a Boolean function that assigns a Boolean value \( \text{T} \) (true) or \( \text{F} \) (false) to each \( G(t) \) such that \( \delta_G : G(t) \rightarrow \{\text{T, F} \} \), \( \forall t \in T \).

In Fig. 2, \( t_1 \) has a predicate \( \langle orderID = \text{T} \wedge gross = \text{T} \rangle \), i.e., \( G(t_1) = \langle orderID = \text{T} \wedge gross = \text{T} \rangle \). \( G(t_1) \) is a logical expression; “\( orderID \)” and “\( gross \)” are variables in logical expressions, and their values are \( \text{T} \) (true), \( \text{F} \) (false) or \( \emptyset \). Note that “\( \wedge \)” means logical operator “AND” while “\( \vee \)” means “OR.” When \( \delta_G(G(t_1)) = \emptyset \langle orderID = \text{T} \wedge gross = \text{T} \rangle = \emptyset \langle \text{T} \rangle \) under a data state \( (M, \delta_D) \), \( t_1 \) can fire.

**Definition 3.5:** A transition \( t \in T \) is enabled at a data state \( \Lambda = (M, \delta_D) \) if
\[
1) \ p \in t \rightarrow M(p) \geq W(p, t) \text{; and } \\
2) \ \exists G(t) \rightarrow \delta_G(G(t)) = \emptyset \text{.}
\]

There are two necessary conditions that must be satisfied. The current marking satisfying \( W(p, t) \) which is a requirement of enabling and firing \( t \), and if \( t \) has a predicate \( G(t) \), then \( (M, \delta_D) \) can make it true. In case of conflicts between two transitions, either one can fire. The firing of either would make the other one lose the firing right. This is just like the conflict in traditional Petri nets.

Here, \( M(p) \geq W(p, t) \text{ means that } n_1, n_2, n_3, \ldots, n_k > a_1 d_1, a_2 d_2, a_3 d_3, \ldots, a_k d_k > \), and the arithmetic of \( M(p) \) and \( W(p, t) \) is based on a \( (n_1, n_2, n_3) > (a_1, a_2, a_3) \), and \( \delta_D \) is also used in a data state \( (M, \delta_D) \), i.e., \( (M, \delta_D) \rightarrow (M', \delta_D') \) describes that \( (M', \delta_D') \) is produced by firing \( t \) at \( (M, \delta_D) \).

Likewise, \( (M, \delta_D) \rightarrow (M', \delta_D') \) means that \( t \) is not enabled at \( (M', \delta_D') \). If there exists a transition sequence \( s = t_1 t_2 \ldots t_k \) and data state sequence \( (M, \delta_D), (M_1, \delta_D_1), (M_2, \delta_D_2), \ldots, (M_k, \delta_D_k) \) making that \( (M, \delta_D) \rightarrow (M_1, \delta_D_1) \rightarrow (M_2, \delta_D_2) \rightarrow \ldots \rightarrow (M_k, \delta_D_k) \) is reachable from \( (M, \delta_D) \), and this can be denoted by \( (M, \delta_D) \rightarrow (M_k, \delta_D_k) \). All the reachable data states from \( (M, \delta_D) \) are denoted by \( R(M, \delta_D) \).

The web programming paradigm of an online shopping system is already under threat from malicious web clients that exploit logic flaws caused by improper distribution of the application functionality between a client and server. The program logic of a hybrid web application is further complicated by the need to securely coordinate different web services that it integrates: failing to do so leaves the door open for attackers to violate security invariants by inducing inconsistencies among these services [2]. Numerous varieties of methods of malicious behaviors are emerging all the time, and clients (browsers) are controlled by users, or the data handled by them are not safe. For example, a malicious shopper can modify the augments in the http request or play more diverse roles than just the shopper, and thus to gain a deeper involvement in the checkout process. As a result, a malicious shopper can purchase an item at an arbitrarily low price, shop for free after paying for one item or even avoids payment [2]. A number of cases have been given in [2], [4], [8], and [9]. There may be some wrong and tampered data flowing in the system resulting in the violation of a transaction property and logic errors. Thus, we define key trading parameters in Definition 3.1 and key transitions next. The latter exist only in clients of an online shopping business process, i.e., handled by the users.

**Definition 3.6:** Given \( t \in T, t = T' \), and \( T' = T'' \), \( t \) is called a key transition if
\[
1) \ \exists s \in S \land \{W(t, p) : p \in P''\}; \text{ and } \\
2) \ \text{The token } s \text{ is produced by } t \rightarrow \delta_D(s) \in \{T, F, \emptyset\}.\]
Key transitions constitute a special part in $T$. Different from others in $T$, firing a key transition may produce several data states, and firing a non-key transition produces one data state only. If $t \in T$ is a key transition, then the token with the type of the key trading parameter produced by firing it has nondeterminacy values, i.e., $T$, $F$ or $O$, and this is expressed by the data state. The changing rules of data states are to be defined below. We use notation $T_X$ in an $EN$ to represent the set of key transitions, and $T_X \subset T$.

**Definition 3.7:** Let $EN = (P, T; F, D, W, S, G)$ be an EBPN, and $\Lambda = (M, \delta_D)$ be a data state of $EN$. A transition $t \in T$, which is enabled at $(M, \delta_D)$, can fire under $M(M \xrightarrow{t})$, and a new marking $M'(M \xrightarrow{t} M')$ is

$$M'(p) = \begin{cases} M(p) - W(p, t), & \text{if } p \in t - t' \\ M(p) + W(p, t), & \text{if } p \in t' - t \\ M(p) - W'(t, p) + W(t, p), & \text{if } p \in t \cap t' \\ M(p), & \text{otherwise.} \end{cases}$$

Meanwhile, if $t \notin T_X$, a new data state $\Lambda'$ is

$$\Lambda' = (M', \delta_D') = (\Lambda, \delta_D') = \left( \begin{array}{l} \forall d \in \{ W(t, p) | p \in t \} \\
\forall d' \in \{ W'(t, p) | p \in t \} \\
\{ W(t, p) | p \in t \} \cap S \\
n - \delta_D' (s) \in \{ T, F, O \} \\
\delta_D (d) \cap T_X \end{array} \right).$$

Else if $t \in T_X$, a new state set $\Gamma$ is

$$\Gamma = \left\{ (M', \delta_D') | M \xrightarrow{t} M', \forall s \in \{ W(t, p) | p \in t \} \cap S \rightarrow \delta_D'(s) \in \{ T, F, O \}, \forall d \in \{ W(t, p) | p \in t \} \cap S \rightarrow \delta_D'(d) = \delta_D(d) \right\}.$$

The occurrence rule mainly considers two changes after firing $t$. One is the change of the marking, while the other is the change of data allocation. There is only one marking that is newly produced, but data allocation is divided into two situations, i.e., whether $t$ is a key transition or not. If $t$ is not, only one data state is produced after firing $t$, because any token produced by firing $t$ is assigned with a fixed value. Otherwise, any token with the type of a key trading parameter is assigned with a fixed value. Note that the initial marking is some specific values of actual systems, and every trigger of transitions needs to bind them. We use notation $\text{EN}$ to represent the set of key transitions, and $T_X \subset T$.

According to the CPN definitions [47], we construct a CPN model depicting the paying operation of TPP in Fig. 3(a). First, we declare several colors (i.e., types) and variables used in a paying operation of TPP. In the CPN of Fig. 3(a), place “TPPState1” corresponds to $p_1$ in Fig. 2(a); Places “OrderID” and “Gross” to $p_2$ in Fig. 2(a); “TPPState2” to $p_3$, and “TransactionID” to $p_4$. The initial marking is assigned to “TPPState1,” “OrderID,” and “Gross,” i.e., $[\text{TPPState1}(1\text{''TPPaid}), \text{TransactionID}(1\text{''AB28397}), \text{OrderID}(1\text{''CD28306})]$, and $\text{Gross}(17.5)$. A marking in CPN is a function that maps each place into a multiset of tokens. Note that the initial marking is some specific values of actual systems, and every trigger of transitions needs to bind them. “AB28397” is an orderID, and 17.5 is the price. Firing “Paying” needs to bind them, i.e., (Paying, $<\text{Status} = \text{TLListen}, \text{order} = \text{AB28397}, \text{price} = 17.5>$). This binding satisfies the guard $[\text{order} = \text{AB28397}$ and $\text{gross} = 17.5]$. Then, “Paying” can fire, and only one marking [TPPState2(1TPPaid), TransactionID(TPAB28397)] is produced in Fig. 3(b), i.e., the changes of the states are determined in CPN. In Fig. 3, it is hard to list all possible follower states related to a transition occurrence instead of only one single follower state.

In this work, we define several basic structures for constructing an EBPN as Fig. 4 shows. Serial structure depicts the situation that several APIs or operation events fire one after another, and the output data of a transition is the input data of the subsequent one. When there exist selective routes in a business process, we use the conditional selection and join structure in Fig. 4. Parallel branch and join structures illustrate how to construct a concurrent scene. The combinations of the shown patterns are allowed. Fig. 4 shows the basic structures of EBPN, and they can be combined according to actual system design specifications. A design specification is a general statement, and it is derived from the design of a process or design of a system made by the system analyst in a software development process.
In workflow management [29], [30], process logic is not hard-coded in the system code. Workflow systems include an interpreter for business process models that allow flexible modifications of process models. However, through our cooperation and communication with the e-commerce enterprises, the distinction between design of a process and system is not so obvious. It is common that a business process is contained in the design of a system. Rather than on flexibility of workflow systems, e-commerce enterprises place more emphasis on security and dependability.

To build up a functional model, we elicit intended functions in terms of design specifications derived from the requirement analysis and system design phase, and construct the EBPN according to the following steps:

Step 1) Obtain the trading parameter set $D$ from the design specifications. Some key trading parameters are defined as $S \subset D$.

Step 2) Construct a data flow model. First, identify operators and APIs in the design specifications. They are represented as transitions. Second, identify the data flows and specify the input and output trading parameters of identified transitions. Third, connect the transitions according to data flow specifications by using places and arcs. The input and output trading parameters are represented as vectors on the arcs according to Definition 3.1.

Step 3) Add control flow. First, identify the order of APIs and operation events in the online shopping system according to key functionalities and design specifications. Second, establish some special control structures to depict the business functionalities according to Fig. 4. Third, add predicates to the corresponding transitions for describing the validation criteria.

Step 4) Set up the initial places. Add three places $p_i$, $p_j$, and $p_k$ as the initial places of three parties, $i, j, k \in \mathbb{N}^+$, which represent the initial states of Shopper, Merchant, and TPP. Then, link them with the first function transitions of three parties respectively by using arcs.

Step 5) Add the initial data state. The initial data state of the functional model is $(M_0, \delta_{POP}) = \{p_i(\lambda_i), p_j(\lambda_j), p_k(\lambda_k), \delta_{DPOP}\}$, where $\lambda_i, \lambda_j$, and $\lambda_k$ represent that Shopper, Merchant, and TPP are ready for a deal, and $\delta_{DPOP}(\lambda_i) = \delta_{DPOP}(\lambda_j) = \lambda_k$.

For an EBPN corresponding to an online shopping business process, the initial data state is unique, which represents that Shopper, Merchant, and TPP are ready for a deal.

For a malicious behavior model, we construct the EBPN depicting the malicious behavior process as follows:

Step 1) Identify transitions. Distinguish legal behaviors that exist only in a functional model and illegal behaviors that exist only in the malicious behavior pattern, and set up the corresponding transitions in accordance with the functional model and malicious behavior pattern.

Step 2) Identify the order and causal relationships among the transitions. In accordance with the input–output relationship of these transitions, as well as their order of execution, connect them by using places and arcs to construct an EBPN of a malicious behavior pattern. If needed, basic structures in Fig. 4 are used.

Step 3) Provide supplementary trading parameters. According to trading parameters required in the actual attack process, add the corresponding vectors to arcs. Some trading parameters that exist only in the malicious behavior model are supplemented.

Step 4) Unify the tags of transitions and places. Compare the functional model and malicious behavior model for determining their shared transitions and places, and rename the transitions and places according to the functional model.

After the functional model and malicious behavior model are constructed, we need to synthesize them to obtain a composed business process that contains the malicious behavior pattern in order to verify whether the malicious behavior pattern can be implemented successfully in the functional model.

Through the above steps, we know that malicious behavior model is constructed based on a functional model. The functional model and the malicious behavior model share certain transitions and places, and they bind the functional model and malicious behavior model with each other. We can synthesize the two models to obtain a composed EBPN with the initial marking of the functional model via a composition defined next.

**Definition 4.1:** Suppose that $E_{N_1} = \{P_1, T_1; F_1, D_1, W_1, S_1, G_1\}$ and $E_{N_2} = \{P_2, T_2; F_2, D_2, W_2, S_2, G_2\}$ are two nets satisfying Definition 3.1, $F_1 \cap F_2 \neq \emptyset$, $T_1 \cap T_2 \neq \emptyset$, $F_1 \cap F_2 \neq \emptyset$, $D_1 \cap D_2 \neq \emptyset$, and $S_1 \cap S_2 \neq \emptyset$. Their composition is $E_N = E_{N_1} \bigcirc E_{N_2} = \{P_1 \cup P_2; T_1 \cup T_2; F_1 \cup F_2; D_1 \cup D_2; W_1 \cup W_2, S_1 \cup S_2, G_1 \cup G_2\}$, in which for $\forall f_1 \in F_1 \subseteq F$, $W_1(f_1) = W(f_1)$; $\forall f_2 \in F_2 \subseteq F$, $W_2(f_2) = W(f_2)$; $\forall t_1 \in T_1 \subseteq T$, $G_1(t_1) = G_1(t_1)$; and $\forall t_2 \in T_2 \subseteq T$, $G_2(t_2) = G_2(t_2)$.

The functional model and the malicious behavior model share certain transitions $(T_1 \cap T_2 \neq \emptyset)$ and places $(F_1 \cap F_2 \neq \emptyset)$. Then, the arcs between these transitions and places are shared by them too, i.e., $P_1 \cap P_2 \neq \emptyset$. Meanwhile, the weights on these arcs are the same in the two models, and thus, the trading
parameters on $F_1 \cap F_2$ are the same, and $D_1 \cap D_2 \neq \emptyset$. As the malicious users are interested in the key trading parameters $(S_1)$ in order to implement their malicious purposes $S_1 \cap S_2 \neq \emptyset$.

Some of the decision processes rely on the experiences of modelers and domain experts. So does the modeling phase of most of the formal models. Note that this work does not restrict the structural properties like the soundness of a WF-net [29], [30]. In a WF-net, it is needed to verify the soundness of a workflow, i.e., whether it can execute from the beginning to end successfully without any deadlock and redundant messages in a workflow. Its soundness focuses on the structural correctness. From our cooperation and communication with the e-commerce enterprises, we know that limited structural concerns in actual online shopping businesses must be addressed, e.g., soundness and whether the shopping process can terminate successfully under malicious behavior patterns. Instead security is paid more attention in e-commerce enterprises. Different from a WF-net study, the emphases of this work are the proposed EBPN model and security verification, i.e., whether the malicious behavior patterns can succeed in an online shopping systems at the process design phase, rather than the soundness of online shopping processes.

V. VERIFICATION OF EBPN WITH MALICIOUS BEHAVIOR PATTERN CONSIDERED

First, according to the malicious behavior model, construct a malicious behavior sequence of attackers. As the malicious behavior sequence is not a complete executable sequence in the composed EBPN, thus, second, we analyze the composed EBPN with the malicious behavior sequence, and derive the relation graph of the malicious behavior sequence and its related transitions. Third, by using EBPN’s dynamic properties, we determine whether the sequence can be executed to the end. If yes, the malicious behavior pattern in the system is feasible. Otherwise, the system is able to withstand such an attack.

By analyzing many cases in online shopping systems, we conclude that malicious behaviors are usually made by one malicious client, as for different online shopping systems, there are different malicious behavior patterns in order to fulfill different malicious purposes. Up to now, we have not found the case that two aggressors cooperate in order to vulnerability a system or different behavior patterns are combined to yield a more complex behavior pattern. First, some definitions are given next.

**Definition 5.1:** Given $E N_1 = (P_1, T_1; F_1, D_1, W_1, S_1, G_1)$ is a functional model of an online trading system, and $E N_2 = (P_2, T_2; F_2, D_2, W_2, S_2, G_2)$ is a malicious behavior model constructed according to $E N_1$. A client malicious behavior sequence of $E N_2$ is a transition sequence $\sigma = t_0 t_1 \ldots t_k$, where $t_0, t_1, \ldots, t_k \in T_2$, $i, j \in \mathbb{N}^+$, $\tilde{\sigma}$ is the transition set of $\sigma$, and $|\tilde{\sigma}|$ is the length of $\sigma$.

The client malicious behavior sequence is derived from the specification of a malicious behavior pattern, and can be extracted as a transition sequence of EBPN.

**Definition 5.2:** Given $E N = E N_1 \cup E N_2 = (P, T; F, D, W, S, G)$ is a composed EBPN with malicious behavior pattern considered, where $E N_1$ and $E N_2$ are a functional model and malicious behavior model, and $\sigma = t_0 t_1 \ldots t_k$ is a client malicious behavior sequence in $E N_2$. A Transition Dependency Graph (TDG) of $E N$ and $\sigma$ can be defined as a three-tuple $TDG(E N, \sigma) = (B, E; L)$, where

1. $H$ is the set of nodes in $TDG(b, N)$, $B \subseteq T$, $\tilde{\sigma} \subseteq H$;
2. $E$ is the set of arcs, $E = \{(b_i, b_j)\} | b_i, b_j \in B, \exists p_k \in P \rightarrow (b_i, p_k), (p_k, b_j) \in F$ or $b_i, b_j$ is subsequence of $\sigma$; and
3. $L: E \rightarrow P$, where if $(b_i, p_k), (p_k, b_j) \in F$, $L(b_i, b_j) = p_k$; Else $L(b_i, b_j) = \emptyset$.

A TDG is used to intuitively portray the relationships among transitions. It is a directed graph reflecting the structural properties of an EBPN statically, and its construction steps are given as Algorithm 1.

**Algorithm 1:** Constructing a TDG

<table>
<thead>
<tr>
<th>Input:</th>
<th>The composed EBPN $E N = (P, T; F, D, W, S, G)$, client malicious behavior sequence $\sigma = t_0 t_1 \ldots t_k$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: TDG ($E N, \sigma$) = (B, E; L).</td>
<td></td>
</tr>
</tbody>
</table>

1. $E = L = \emptyset$;
2. $B = B_t = \{t_k\}$;
3. While $B_t \neq \emptyset$ and $|\sigma| \neq 0$ Do
4. Foreach $t_i \in B_t$ Do
5. If $t_i$ has a predecessor node $t_{i-1}$ in $\sigma$, and $t_{i-1} \notin \{t_i\}$ and $|t_i| = 1$ Then
6. If $(t_{i-1}, t_i) \notin E$ Then
7. $E = E \cup \{(t_{i-1}, t_i)\}$, $L = L \cup \{L(t_{i-1}, t_i) = p, p \in P, \{t_{i-1}, p\}, \{p, t_i\} \in F\};$
8. Endif
9. If $t_{i-1} \notin B$ Then $B = B \cup \{t_{i-1}\}$, $B_t = B_t \cup \{t_{i-1}\} \setminus \{t_i\}$; Endif
10. If $t_i \in \sigma$ Then Remove $t_i$ from the end of $\sigma$; Endif
11. Endif
12. Else If $t_i$ has the predecessor node $t_{i-1}$ in $\sigma$ and $t_{i-1} \notin \{t_i\}$ Then
13. If $(t_{i-1}, t_i) \notin E$ Then $E = E \cup \{(t_{i-1}, t_i)\}$, $L = L \cup \{L(t_{i-1}, t_i) = \emptyset\};$
14. If $t_{i-1} \notin B$ Then $B = B \cup \{t_{i-1}\}$, $B_t = B_t \cup \{t_{i-1}\} \setminus \{t_i\}$; Endif
15. If $t_i \in \sigma$ Then Remove $t_i$ from the end of $\sigma$; Endif
16. Endif
17. Else Foreach $b \in \{t_i\}$ Do
18. If $(b, t_i) \notin E$ Then
19. $E = E \cup \{(b, t_i)\}$, $L = L \cup \{L(b, t_i) = p, p \in P, \{b, p\}, \{p, t_i\} \in F\};$
20. Endif
21. If $b \notin B$ Then $B = B \cup \{b\}$, $B_t = B_t \cup \{b\} \setminus \{t_i\}$; Endif
22. Endforeach
23. If $t_i \in \sigma$ Then Remove $t_i$ from the end of $\sigma$; Endif
24. Endif
25. Endforeach
26. Repeat
Theorem 1: Algorithm 1 is correct and can be terminated.

Proof: Algorithm 1 starts from the last node \( t_k \) in \( \sigma \), and back traverses \( \mathcal{E}N \) according to the orders of transitions in \( \sigma \), i.e., the traversal algorithm is based on not only the structure of \( \mathcal{E}N \), but also the sequential relationships of \( \sigma \). Algorithm 1 absorbs the transitions that are related with the firing of \( \sigma \) into \( H \), thus \( H \subseteq \mathcal{T} \), and in the extreme cases, \( H = \mathcal{T} \). Steps 5–16 absorb all the transitions in \( \sigma \), and Steps 17–24 absorb the related transitions that are not in \( \sigma \). Thus, \( \bar{\sigma} \subseteq B \). If \( b_i, b_j \in B \), \( \exists p_k \in P \rightarrow \{b_i, p_k\}, \{p_k, b_j\} \in \mathcal{F} \), Steps 6–8 or 18–20 adds \( \{b_i, b_j\} \) to \( \bar{\sigma} \), and also adds \( p_k \) as the mark of \( \{b_i, b_j\} \). Otherwise, when \( b_i, b_j \in B \) and \( b_i \in \sigma \), \( b_i \) is a key transition, according to Definition 5.2, \( b_i \) is the set of nodes; \( b_j \) is refreshed, and \( \mathcal{F} \) is the mark of \( \{b_i, b_j\} \). Therefore, Algorithm 1 satisfies Definition 5.2 and is correct.

There are three decision conditions at Steps 3, 5, and 12 that are used to determine the situation of predecessor nodes of current node \( t_i \) in \( \mathcal{E}N \) and \( \sigma \). \( B_i \) is the set of current nodes that are ready to be processed. \( \sigma \) and \( T \) are finite sets, and \( B_i \) is refreshed by deducting the primary one after new produced nodes have been extended in Steps 9, 14, and 21. Then, the algorithm moves backward by adding new produced nodes to \( B \) and removing the current node from \( B_i \) until it becomes empty. Algorithm 1 also explains that if one element of \( T \) \( \mathcal{D} \mathcal{G}(\mathcal{E}N, \sigma) \) has existed in \( B \), it need not absorb it again. This means that the process of constructing \( \mathcal{D} \mathcal{G}(\mathcal{E}N, \sigma) \) would not be performed repeatedly many times. \( \sigma \) continues to decrease through Steps 10, 15, and 23, and thus \( \sigma \) must be empty at last. This also reflects the effect of \( \sigma \) in the process of constructing \( \mathcal{D} \mathcal{G}(\mathcal{E}N, \sigma) \). Hence, Algorithm 1 can be terminated \( \square \).

Algorithm 1 is the construction process of TDG, and TDG is defined in Definition 5.2. Here, the correctness means that the output of Algorithm 1 is indeed a TDG that is consistent with Definition 5.2. A possible input of Algorithm 1 is an arbitrary EBPN model and an arbitrary client malicious behavior sequence. Given such input, it always terminates.

Algorithm 1 is essentially a backward traversal algorithm of graphs, and its cost is bounded by the number of elements in \( EN \), i.e., its complexity is \( O(|\mathcal{P}| + |T| + |\mathcal{F}|) \), in which \( |\mathcal{P}| \) is the number of places, \( |T| \) is the number of transitions, and \( |\mathcal{F}| \) is the number of arcs in \( \mathcal{E}N \).

Special operations of key transitions in client malicious behavior sequences are considered according to the specific malicious behavior patterns, such as tampering some key trading parameters. If \( t_i \) is a transition in a client malicious behavior sequence, and \( t_i \) is a key transition, according to the specification of a malicious behavior pattern, it sets the value of \( d_i \) to \( \mathcal{T} \) or \( \mathcal{F} \) or \( \emptyset \). Then, we use the notation \( t_i \{d_i\}, t_i \{d_i, \mathcal{F}\} \) or \( t_i \{d_i, \emptyset\} \) to represent the value of the token with the type of \( d_i \) produced by it. Then, the complete client malicious behavior sequences attached with operation descriptions are obtained.

Definition 5.3: Given \( EN_1 = (P_1, T_1; F_1, \mathcal{D}_1; W_1, S_1(G_1)) \) is a functional model of an online trading system, and \( EN_2 = (P_2, T_2; F_2, D_2; W_2, S_2, G_2) \) is a malicious behavior model constructed according to \( EN_1 \). A client malicious behavior sequence of \( EN_2 \) is a transition sequence \( \sigma = t_1, t_2, \ldots, t_k \). Then, a \textit{complete client malicious behavior sequence} of \( EN_2 \) is a sequence \( \rho = (t_1, \Sigma_1(t_2), \Sigma_2, \ldots, t_k, \Sigma_k) \), \( \Sigma_i \) is called an operation of \( t_i \). If \( t_i \in \bar{\sigma} \) is a key transition, then \( \Sigma_i = \{s \in S' \subseteq S_2, \xi \in \{\mathcal{T}, \mathcal{F}, \emptyset\} \} \), else \( \xi = \emptyset \).

Here, \((M, \bar{\delta}_D) \xrightarrow{\bar{\delta}_D} \) means that \( t \) is fired according to its operation \( \Sigma \) at \((M, \delta_D) \). If \( t \) is a key transition and \( \Sigma = \{s \xi \in S' \subseteq S_2, \xi \in \{\mathcal{T}, \mathcal{F}, \emptyset\} \}, \) when \( t \) fires, the token with type of key trading parameters produced by firing \( t \) would be assigned with only one value, i.e., \( \xi \), and a data state \((M', \bar{\delta}_D') \) is produced, not a state set \( \Gamma \), because \( \Sigma \) defines the data allocation after firing \( t \), i.e.,

\[
(M', \delta_D') = (M', \forall s \in \{W(t, p)|p \in \bar{\delta}_D'(s) \in \xi \} \land s \xi \in \Sigma \)
\]

Without the restriction of \( \Sigma \), according to Definition 3.7, when \( t \) fires, the value of the token with a type of key trading parameters produced by \( t \) would be divided into three situations, i.e., \( \mathcal{T}, \mathcal{F}, \emptyset \), and a state set \( \Gamma \) is produced.

Definition 5.4: Let \((M_0, \delta_D)\) be the initial data state of \( \mathcal{E}N = (P, \mathcal{T}; F, D, W, S, G) \). Its Reachability Data State Graph \((\mathcal{R})\) can be defined as a three tuple \( \mathcal{R}(\mathcal{E}N) = (R(M_0, \delta_D), H, Y) \), where

1. \( R(M_0, \delta_D) \) is a set of nodes;
2. \( H \) is the set of arcs, \( H = \{(M_I, \delta_D), (M_J, \delta_D'), (M_J, \delta_D)| \in R(M_0, \delta_D); \forall t \in T; (M_I, \delta_D) \xrightarrow{\bar{\delta}_D} (M_J, \delta_D') \}; \) and
3. \( Y : H \rightarrow Y \{(M_I, \delta_D), (M_J, \delta_D')| = t_k \text{ if } (M_I, \delta_D) \xrightarrow{\bar{\delta}_D} (M_J, \delta_D') \}, \) and \( t_k \) is called the label of the arc between \((M_I, \delta_D) \) and \((M_J, \delta_D') \), \( (M_I, \delta_D) \) is the successor node of \((M_I, \delta_D) \), and \((M_J, \delta_D') \) is the predecessor node of \((M_I, \delta_D) \).

Due to the problem of state explosion [33, 34], \( \mathcal{R}(\mathcal{E}N) \) may be infinity as an EBPN with an initial data may have an infinite number of reachability data states. Hence, we construct \( \mathcal{R}(\mathcal{E}N) \) according to \( \mathcal{T}DG(\mathcal{E}N, \sigma) \) and \( \rho \). Its construction algorithm is as follows:

Algorithm 2: Constructing \( \mathcal{R}(\mathcal{E}N) \) according to \( \mathcal{T}DG(\mathcal{E}N, \sigma) \) and \( \rho \)

Input: The composed EBPN \( \mathcal{E}N = (P, \mathcal{T}; F, D, W, S, G) \), \( (M_0, \delta_D) \), the complete client malicious behavior sequence \( \rho = t_1, t_2, \ldots, t_k \), \( \mathcal{T}DG(\mathcal{E}N, \sigma) = (B; E; L) \).

Output: \( \mathcal{R}(\mathcal{E}N) \), “YES” or “NO.”

1. Let \((M, \delta_D)\) be the root node, and mark it with “New”;
2. \( B_1 = \{t_1\} \);
3. While “New” nodes exist and \(|\rho| \neq \emptyset \) Do
   Choose an arbitrary “New” node as \((M, \delta_D)\);
6. Marking \((M, \delta_D)\) as “Terminated node,” and go to step 2;
7. Endif;
8. If \( \exists t \in B_1 \) and \( t \) is a transition in \( \rho \) and \((M, \delta_D) \xrightarrow{\bar{\delta}_D} \) Then
9. \((M', \delta_D') = (M, \delta_D) \xrightarrow{\bar{\delta}_D} \)
10. If \((M', \delta_D')\) exist in \(\mathcal{R}(EN)\) Then
11. Create a directed edge from \((M, \delta_D)\) to \((M', \delta_D')\), and mark the edge labeled with \(t\);
12. Endif
13. Else Create a new node \((M', \delta_D')\), create a directed edge from \((M, \delta_D)\) to \((M', \delta_D')\), and mark the edge labeled with \(t\);
14. Endif
15. Mark \((M', \delta_D')\) with “New” and remove mark “New” of \((M, \delta_D)\);
16. \(B_t = B_t \cup \{t'|t'\text{ is successor node of } t\} - \{t\}\);
17. Remove \(t\)' from the beginning of \(\rho\);
18. Endif
19. Else Foreach \(\forall t \in B_t\) and \((M, \delta_D)\) Do
20. \((M', \delta_D') = (M, \delta_D) \xrightarrow{t} ;\)
21. Executing Step 10–16;
22. Endforeach
23. Endif
24. Repeat
25. If \(|\rho| = 0\) Then output “YES” Else output “NO”;

**Theorem 2:** Algorithm 2 can be terminated.

**Proof:** Algorithm 2 is essentially a construction algorithm of Reachability Data State Graph according to \(TDG(EN, \sigma)\) and \(\rho\). Due to the certainty of \(TDG(EN, \sigma)\) and \(\rho\), the process of constructing \(\mathcal{R}(EN)\) avoids much uncertainty, thereby reducing the number of generated states. The length of \(\rho\) is fixed, and \(\rho\) continues to decrease through Step 17. Thus, Algorithm 2 can be terminated. \(\square\)

The cost of Algorithm 2 is bounded by the number of nodes and arcs in \(TDG(EN, \sigma)\) with the length of \(\rho\), i.e., its complexity is \(O(B + |F| + |\rho|)\), where \(B\) is the number of nodes in \(TDG(EN, \sigma)\), and \(|E|\) is the number of arcs.

Given an EBPN \(EN = (P, T; F, D, W, S, G)\) and an initial data state \((M_0, \delta_{D_0}); \mathcal{R}(EN)\) as constructed directly may be infinite. However, Algorithm 2 constructs \(\mathcal{R}(EN)\) according to \(TDG(b, N, \sigma)\) and \(\rho\). \(\mathcal{R}(EN)\) produced by Algorithm 2 is a subgraph of the complete one (this complete one may have infinity data states). This is different from the relationship between Reachability Marking Graph and Coverability Tree in traditional Petri nets. Thus, \(\mathcal{R}(EN)\) produced by Algorithm 2 is just a part of all the reachability data states, and no information is lost for the reachability data states in \(\mathcal{R}(EN)\). Using Algorithm 2, we can determine whether a client malicious behavior sequence can be executed successfully. If the output is “YES,” the malicious behavior pattern in the system is feasible. Otherwise, the malicious behaviors of clients will not succeed, and the system is able to withstand such an attack.

VI. A CASE STUDY

We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.

As Fig. 5 shows, three parties including TPP, Merchant, and Shopper, as well as data exchange among them are described. The business process adopts an HTTP message that TPP uses to initiate communication against a typical malicious behavior pattern. We have applied our approach to the modeling and verification of a real-world online shopping business process. Based on the assumption that the business process is at the conceptual design phase, our purpose is to identify whether it is resistant to the known malicious behavior patterns.

In our procedure, at process design time, we need to verify the entire malicious behavior patterns in malicious behavior pattern library in order to make the software design provably secure from the known malicious behavior patterns at the conceptual design phase. Since the verification principles are the same, the specific malicious behavior pattern found in [2] will serve as the only case in this work. First, we briefly introduce the business process and then demonstrate a complete process of its modeling and verification against a typical malicious behavior pattern.
By replaying this message, the malicious user is able to obtain the illegal interests, and they will study such patterns appearing recently, and may be familiar with the business process to be invaded. Even if they do not know the details of a business process, they can constantly try to implement their malicious behavior patterns on a business process, successfully or not. They can play more diverse roles than just a shopper, and thus to gain deeper involvement in a checkout process than the latter could in a traditional client-server interaction [2]. Once they are successful, they would cause great damage.

The study in [2] has revealed numerous security related logic flaws in a variety of merchant systems, ranging from a high-quality open source software (NopCommerce), to a leading commodity application (Interspire) and high-profile merchant websites powered by closed-source proprietary software such as Buy.com and JR.com. Their attacker model is fairly simple—the attacker is a malicious shopper whose only capability is to call the web APIs exposed by the merchant and the TPP websites in an arbitrary order with arbitrary argument values. They show that everyone who has a computer and a small amount of cash is a qualified attacker. By exploiting the logic flaws, a malicious shopper is able to purchase at an arbitrarily-set price, shop for free after paying for one item, or even avoid payment.

In fact, process designers of the online shopping business process do not have to know the malicious behavior patterns as they are just business managers and system analysts in software engineering. Their task is to design the business process of an online shopping system because they are familiar with the business process to be invaded. Even if they do not know the details of a business process, they can constantly try to implement their malicious behavior patterns on a business process, successfully or not. They can play more diverse roles than just a shopper, and thus to gain deeper involvement in a checkout process than the latter could in a traditional client-server interaction [2]. Once they are successful, they would cause great damage.

The people who use the proposed methods in this work are the system verifiers or modelers. Our approach aims to ensure the security of a business process against the malicious behaviors, and to eliminate hidden dangers early at the design phase. Consequently, its use can greatly increase the difficulty for malicious users to materialize their malicious behaviors successfully at run time. Thus, by our methods, security is largely guaranteed for the online shopping business processes at the process design phase, and the implemented business processes are much more secure than those developed without using the proposed methods.

The study in [2] has revealed numerous security related logic flaws in a variety of merchant systems, ranging from a high-quality open source software (NopCommerce), to a leading commodity application (Interspire) and high-profile merchant websites powered by closed-source proprietary software such as Buy.com and JR.com. Their attacker model is fairly simple—the attacker is a malicious shopper whose only capability is to call the web APIs exposed by the merchant and the TPP websites in an arbitrary order with arbitrary argument values. They show that everyone who has a computer and a small amount of cash is a qualified attacker. By exploiting the logic flaws, a malicious shopper is able to purchase at an arbitrarily-set price, shop for free after paying for one item, or even avoid payment.

In fact, process designers of the online shopping business process do not have to know the malicious behavior patterns as they are just business managers and system analysts in software engineering. Their task is to design the business process of an online shopping system because they are familiar with the business process to be invaded. Even if they do not know the details of a business process, they can constantly try to implement their malicious behavior patterns on a business process, successfully or not. They can play more diverse roles than just a shopper, and thus to gain deeper involvement in a checkout process than the latter could in a traditional client-server interaction [2]. Once they are successful, they would cause great damage.

The people who use the proposed methods in this work are the system verifiers or modelers. Our approach aims to ensure the security of a business process against the malicious behaviors, and to eliminate hidden dangers early at the design phase. Consequently, its use can greatly increase the difficulty for malicious users to materialize their malicious behaviors successfully at run time. Thus, by our methods, security is largely guaranteed for the online shopping business processes at the process design phase, and the implemented business processes are much more secure than those developed without using the proposed methods.

Fig. 6 is the schematic diagram of the malicious behavior pattern. It has “begin” and “end” nodes representing a complete malicious behavior process. A rectangle and its text mean an operation. The directed arcs represent the orders of operations, and the labels on the arcs mean how many times the malicious user carries out the process. In this example, first, the malicious user changes the message in Step 2.a by setting its orderID to be empty and setting IPNHandler to be his/her. This change makes the TPP’s IPN message is delivered to the malicious user via Step 2.a.a. This action gives him/her an ipn message signed by the TPP, which consists of the argument list (orderID = empty, gross, merchantID, status) and a signature C°. By replaying this message, the malicious user is able to check out an arbitrary number of orders with the same prices. Each time, all he/she needs to do is to place a new order by Step 1.a, set the order’s ID as the browser cookie ORDER_ID, and finally call Merchant’s IPNHandler with the arguments in Step 2.a, and finally call Merchant’s finishOrder by Step 3.a. In this example, the malicious user plays all three roles: Shopper, Merchant and TPP. Note that the malicious user also changes his/her browser cookie before he/she calls Merchant’s handleIPN as a TPP. Therefore, it is a hybrid of TPP behaviors and browser behaviors [2]. This is a newly and complicated case in online shopping systems nowadays where an attacker plays three roles and call APIs in arbitrary orders, and results in the fund loss of Merchant. Therefore, we need a formal mechanism to determine whether an online shopping system is able to withstand this type of malicious behavior.
Using the proposed modeling methods and intended function specifications in Fig. 5 and Table I, we can model the business process using EBPN. The operation events of Shopper, as well as APIs of TPP and Merchant, are depicted by transitions in an EBPN. As Fig. 7(a) shows, the operation “Place an order” of Shopper is represented by \( t_1 \), and the phrase in it is used to signal its function. \( t_2 \) is an API of Merchant, representing the function of processing an order from Shopper. \( p_2 \) is a data transfer channel between them. In Fig. 7(a), \( p_1, p_3 \) and \( p_7 \) are the initial places of Shopper, Merchant, and TPP, respectively, and they are linked with three transitions: \( t_1, t_2 \) and \( t_3 \) via two contrary arrows, and this means that three parties are always ready to start a new transaction. The initial data state is \( M_0 = \{ \text{StopIdle}, \text{Merchant\textunderscore Order}, \text{TPP\textunderscore Order} \} \). Two predicates are added to \( t_5 \) and \( t_6 \) for describing the validation criteria in Table I, and different implementations caused by the validation criteria are depicted by the conditional selection and join structures through \( t_5 \), \( t_6 \), and \( t_7 \). In Fig. 7(a), \( t_8 \) is the key transition, and \( \{ \text{orderID}, \text{gross}, \text{IPNhandle} \} \) is the set of key trading parameters. The malicious behavior model corresponding to the malicious behavior pattern in Fig. 6 is expressed in Fig. 7(b). \( t_1, t_5 \), and \( t_6 \) are legal transitions in the functional model, while \( t_{10} \) and \( t_{11} \) are the malicious behaviors that are used to play Merchant and TPP by malicious users. Note that the marks of legal places and transitions in Fig. 7(b) are consistent with them in Fig. 7(a). In Fig. 7(b), \( t_2 \) and \( t_{15} \) are the key transitions, and the key trading parameters are the same as those in Fig. 7(a).

After the functional model and malicious behavior model are constructed, we need to synthesize them in order to verify that whether the malicious behavior pattern can be implemented successfully in the functional model. According to Definition 4.1, we synthesize the two models of Fig. 7(a) and (b) to obtain the composed EBPN, as shown in Fig. 8.

For the malicious behavior pattern of Figs. 6 and 7(b), it is easy to construct the client malicious behavior sequence \( \sigma = t_1 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{11} t_{12} t_{13} \). Then, according to Definition 5.3, the complete client malicious behavior sequence of Fig. 8 is \( \rho = t_1 t_3 \{ \text{orderID}, \text{merchantID}, \text{gross}, \text{IPNhandle} \} t_10 \{ \text{IPNhandle} \} t_11 t_12 t_13 t_{14} t_{15} t_{16} t_{17} \). Using Algorithm 1, the TDG of the model in Fig. 8 is showed in Fig. 9. The based on TDG and is shown in Fig. 10. \( \rho \) can be executed successfully, and the output is “YES.” Thus, we know that the online trading system cannot withstand an attack of multiple checkouts with one payment. By analyzing based on TDG and \( \rho \), a business process and the corresponding malicious behavior model, we find that after “Receive data from Merchant and tamper orderID and IPNhandle,” TPP accepts the argument \((\text{orderID} = \text{empty}, \text{gross}, \text{merchantID}, \text{status})\) and transfers it to handleIPN at Step 2.a.a. Then, handleIPN of Merchant accepts it and returns a result. In fact, that is the problem as handleIPN of Merchant should not accept it. It is worth noting that the problem is caused by transitions \( t_5 \) and \( t_6 \) corresponding to the function LoadOrderByID in Table I. It is heavily used utility functions in the online shopping systems, and is usually called in many situations, e.g., when handling a TPP’s request or handling a browser’s request. Therefore, it is designed to be generic: when handling a TPP request, e.g., in handleIPN, the
Fig. 8. The composed EBPN integrating malicious behavior pattern.

Fig. 9. The TDG of Fig. 8 and \( \sigma \).

function is called with an explicit \( \text{orderID} \) [2]. However, a typical request from the browser does not contain the \( \text{orderID} \) field in the request URL. In this situation, loadOrderByID(\( \text{empty} \)) would be called, and the \( \text{orderID} \) is retrieved from a cookie named \( \text{ORDER\_ID} \). Therefore, this generic design turns out to be problematic. Changing this generic design can resolve such a security issue in the process of Fig. 5.

Traditional security technologies include encryption, decryption [50], protocol verification [13]–[16] and testing [44], in which testing is an engineering mean for quality assurance of software, rather than a formal method. It aims to find bugs by executing a program with test cases after the software has been implemented. Testing mainly includes white box technology in which testers do know the inner details of the software and black one in which they do not know them. Test cases are different according to the two technologies. Protocol verification is a traditional security technology at the network level. It is a formal method used to verify the protocols of distributed systems such as Security Socket Layer (SSL) and Secure Electronic Transaction (SET) protocols. Different from them, this work represents a novel attempt to model and verify online shopping business processes by considering malicious behavior patterns at the design level and application level, and determines whether an online shopping business process can withstand typical malicious behavior patterns by formal methods. Thus, the security of systems is guaranteed in the requirement analysis phase instead of that the systems are modified at the implementation and running phase, which is known to be extremely costly.

Malicious behavior patterns in online shopping systems are some behaviors that comply with the traditional security requirements (privacy, integrity, etc.) of the systems, but result in malicious activities. This is beyond the scope of traditional security technologies. Using the same procedure above, we can also verify whether such a system can resist other malicious behavior patterns at the process design phase. We have modeled and verified some typical malicious behavior patterns [2], [8], [9], and our approaches can well deal with such type of issues. The result is shown in Table II. In comparison, traditional methods cannot do these. However, due to different emphasis of different methods, the proposed methods in this work cannot deal with many issues in traditional security technologies.
In addition, our methods cannot deal with the system exception at run time after the system is implemented. Our proposed approach works at the time of process design, but the malicious behavior patterns are derived from the issues appearing at the run time. Our ultimate goal is to guarantee that the design of an online shopping business process is immune to the malicious behavior patterns. If the design has any security issue, the implemented system would be worse. Thus, guaranteeing the security of a process design is necessary. On the basis of the security of a process design, the implementation of a business process can resist the malicious behavior patterns that have been verified through our proposed methods, instead of finding the problems at run time and then repair them, which bears high cost. The occasional exceptions, such as network latency or a system crash at run time which results in a security accident, are not the focus of this work.

VII. CONCLUSION

This work’s contribution is a systematic approach for modeling and verification of online shopping business processes against some specific malicious behavior patterns. The modeling process is done in a step-by-step manner, and the

<table>
<thead>
<tr>
<th>Malicious behavior patterns</th>
<th>Our methods</th>
<th>Encryption, decryption</th>
<th>Testing</th>
<th>Protocol verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Paying to the attacker himself to check out from the victim”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“Paying for a cheap order to check out an expensive one”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“A customer purchases products at a reduced price”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“Sealing a payment notification and replaying it many times”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“Adding items into the cart after the checkout button is clicked”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“A customer pays for another customer’s purchase”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>“A customer cancels another customer’s order”</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
composed EBPN by considering the malicious behavior pattern is built through composing function and malicious behavior models. By analyzing the models through two special procedures, we can verify whether such a process is resistant to some specific malicious behaviors. Through a case study, we illustrate how to model and verify an online shopping system at the design level.

The proposed methodology can also be used in other online shopping business processes and malicious behavior patterns that have three parties through defining different business processes and data sets. It is a basic and generic method of modeling and analyzing systems. In fact, there may be another party who participates in an online shopping process sometimes, e.g., banks, but in the trading process, it is common that users transfer a sum of money from bank to TPP such as PayPal or Alipay, and then pay for goods with the money in their account in TPP in a relatively long time, and do not use the bank transfer. Therefore, the opportunities for banks to involve in a trading process are limited. In the cases in [2], [4], there are hardly any other parties involved. In addition, the online shopping process with another party (except Shopper, Merchant, and TPP) is more complex, and there must be more security issues that we do not know. Thus, our future work will be devoted to extend our work to multiparty cases.

Since the existing malicious behavior patterns and threats lack any standards, another important work is to establish a public malicious behavior pattern library by using formal methods. It is also an important task to study the structural properties of EBPN and build its timed version for performance analysis [53]–[60].

REFERENCES


Wang-Yang Yu received the M.S. degree from Shandong University of Science and Technology, Qingdao, China, in 2009, and the Ph.D. degree from Tongji University, Shanghai, China, in 2014.

He is currently a Lecturer with the College of Computer Science, Shaanxi Normal University, Xi’an, China. His research interests include the theory of Petri nets, formal methods in software engineering and trustworthy software.

Chun-Gang Yan received the Ph.D. degree from Tongji University, Shanghai, China, in 2006.

She is currently a Professor with the Department of Computer Science and Technology, Tongji University, Shanghai, China. Her current research interests include concurrent model and algorithm, Petri nets theory, formal verification of software, trusty theory on software process. She has published more than 30 papers in domestic and international academic journals and conference proceedings.

Zhi-Jun Ding received the M.S. degree from Shanghai University of Science and Technology, Taian, China, in 2001, and the Ph.D. degree from Tongji University, Shanghai, China, in 2007.

Currently, he is a Professor with the Department of Computer Science and Technology, Tongji University. He has published more than 50 papers in domestic and international academic journals and conference proceedings. His research interests are in formal engineering, Petri nets, services computing, and workflows.

Chang-Jun Jiang received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1995.

He is currently a Professor with the Department of Computer Science and Technology, Tongji University, Shanghai, China. He has published more than 100 publications. His current research interests include concurrency theory, Petri nets, formal verification of software, cluster, grid technology, program testing, intelligent transportation systems, and service-oriented computing.

Meng-Chu Zhou (S’88–M’90–SM’93–F’03) received the B.S. degree in control engineering from Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 1990, and is a Distinguished Professor of Electrical and Computer Engineering. His research interests are in Petri nets, sensor networks, web services, semiconductor manufacturing, transportation and energy systems. He has over 560 publications including 11 books, 270 + journal papers (majority in IEEE TRANSACTIONS), and 18 book-chapters.

Prof. Zhou is a Life Member of the Chinese Association for Science and Technology-USA and served as its President in 1999. He is a Fellow of the American Association for the Advancement of Science (AAAS) and Fellow of International Federation of Automatic Control (IFAC). He is the founding Editor of IEEE Press Book Series on Systems Science and Engineering, an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS: SYSTEMS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.