Verification of Use Case with Petri Nets in Requirement Analysis*

Jinqiang Zhao[1,2] and Zhenhua Duan[1,**]

[1] Institute of Computing Theory & Technology, Xidian University, Xi'an, 710071, P.R. China

[2] State Key Laboratory of Software Engineering, Wuhan University, 430072, P.R. China

jqzhao1985@gmail.com, zhenhua_duan@126.com

Abstract. Requirement analysis plays a very important role in reliability, cost, and safety of a software system. The use case approach remains the dominant approach during requirement elicitation in industry. Unfortunately, the use case approach suffers from several shortcomings, such as lacking accuracy and being difficult to analyze and validate the dynamic behavior of use cases for concurrency, consistency, etc. This paper proposes an approach for overcoming limitations of the use case approach and applies the approach in Model Driven Development (MDD). Timed and Controlled Petri Nets are used as the formal description and verification mechanism for the acquired requirements. Use cases are used to elicit the requirements and to construct scenarios. After specifying the scenarios, each of them can be transformed into its correspondent Petri-net model. Through analyzing these Petri-net models, some flaws or errors of requirements can be detected. The proposed approach is demonstrated by an E-mail client system.

Keywords: use case; Model Driven Development; Petri net; requirement analysis.

1 Introduction and Related Works

Software development usually consists of the following stages: requirement analysis, design, code and testing. Many research studies have shown the considerable influence of early requirement analysis on the reduction of the unnecessary costs, confusion and complexity in the later phases of software development [1].

Therefore, high quality of requirement analysis can most likely reduce some potential risk occurred in later phases of software development.

* This research is supported by the NSFC Grant No. 60433010, NSFC Grant No. 60873018 jointly sponsored by Microsoft Asia Research Academy, Defence Pre-Research Project of China No. 51315050105, SRFDP Grant 200807010012 and SKLSE20080713.

** Corresponding author.

With its adoption into the UML, the use case diagram soon became widely adopted in industry, and it remains the dominant approach to requirements description and specification within the UML. The use case approach, based on the concept of scenarios, is commonly used as a requirements elicitation technique. Software requirements are stated as a collection of use cases. Each of them is written in the user's perspective and described by a specific flow of events in the system [2]. The use case approach offers several practical advantages during software development. On one hand, software users can understand whether the software system satisfies their need at the very beginning in software development when they see use case diagrams. Their complaints about a system can be directly reported to requirements developers and some necessary changes can be made in the requirements model accordingly. Use case diagrams make it possible for users to evaluate the system behavior before code is written. On the other hand, use case diagram can be used as blueprints during the whole software development [3].

However, some defects of the use case approach also exist. First, use cases are often described in informal language. Although readability of informal language is very important, one still needs to have precision in specification, as there are risks of ambiguity in natural language. Second, it's difficult to analyze and validate the dynamic behavior of use cases for concurrency, consistency, etc. To overcome the limitations of the use case approach, several researchers have tried to formalize the informal aspects of use cases. P. Hsia et al. [4] used a BNF-like grammar to formally describe use cases. The authors propose such a method and apply it to a simple PBX (private branch exchange) system. Their method has a formal mathematical base, generates precise scenarios, accommodates change, and keeps users involved in the process. The method is supported by developing a complex grammar. Andersson and Bergstrand [5] used Message Sequence Chart (MSCs). MSC provides several features aimed at enhancing the expressiveness of individual MSCs. Examples include constructs to specify the conditional, iterative, or concurrent execution of MSC sections. An MSC-based approach has advantages over the grammar-based approach in terms of scalability and understandability. Wuwei Shen et al. [6] presented HCL (High-Level Constraint Language) to formally and textually represent a requirement model, and any error or undesired result observed through the execution of HCL specification. Xiaoshan Li et al. [7] demonstrated a use case-driven, incremental and iterative requirement analysis supported by a simple formal semantic model. A use case is defined in terms of its pre and post conditions. The pre-conditions and post conditions of use cases and state constraints are written in the relational algebra.

In this paper, we propose an approach based on Petri nets, and apply the approach in MDD (Model Driven Development). In MDA (Model Driven Architecture), there are 4 types of models [8]: CIM (Calculation Independent Model), PIM (Platform Independent Model), PSM (Platform Specific Model), and ISM (Implementation Specific Model). So in our approach, use case diagram, as in level of CIM, is used to elicit the user requirements. For each use case, use case

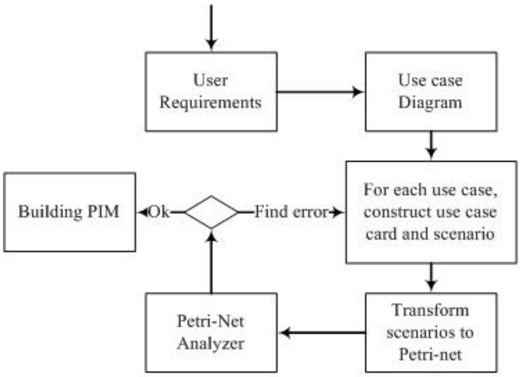
User Requirements
Use case Diagram
For each use case, construct use case card and scenario
Building PIM
Ok
Find error
Petri-Net Analyzer
Transform scenarios to Petri-net

Fig. 1. An overview of the approach

card is used to describe the properties (use case name, primary actor, pre condition, post condition, etc.) and some event flow of the use case. Through analyzing the use case card, scenarios are constructed. Each of the scenarios can be transformed into its correspondent Timed and Controlled Petri Net (TCPN). After the analysis and verification of these Petri net models, if some flaws or errors exist, the process will return back to reconstruct the use case description and scenarios, otherwise, models in the level of PIM will be built through analyzing these models in requirement analysis.

An overview of the proposed approach is shown in Fig. 1. This approach based on Petri nets is well-suited to overcome the limitations caused by the informal description of use case. Among many techniques, we choose Petri nets in our approach due to its mathematical simplicity, modeling generality, locality in both states and actions, and graphical representation, as well as well-developed qualitative and quantitative analysis techniques, and the existence of analysis tools[18].

Some researchers have tried to describe requirements using Petri nets, such as [2] [20], they do research on Modular Petri nets and Time Petri nets. However, they are different from the approach in this paper:

– The Research in this paper is based on the MDD perspective, and the approach is applied in MDA.
– The requirements we verified is the use case card, and event flow is written using the improved scenario language.
– Timed and Controlled Petri Net is used to make Petri net model be controlled enough to describe the selective behavior and message interaction between objects.
– The criteria for requirement quality assessment are defined and requirement model is verified by analyzing the Petri net models.

However, any techniques are not completely perfect. Timed and Controlled Petri Nets proposed in this paper will face challenges when event flow of the use case in industrial systems is quite complex. In that case, use case described using TCPN will be not only difficult to understand but also sensitive to changes. Fortunately, the 'include' relationship gives a good indication that some parts of event flows in use case can be extracted to compose a new use case which has 'include' relationship with initial use case.

The rest of this paper is organized as follows. Section 2 describes software requirement and use case diagram; for each use case, it describes the method of constructing use case card and scenario. Section 3 focuses on the definition of the Timed and Controlled Petri Nets. Section 4 introduces the transformation from use case descriptions to Timed and Controlled Petri Net models. The verification of TCPN models is discussed in Section 5. Finally, Section 6 presents the conclusion and future work.

2 Requirement Elicitation

During software development, requirements analysis and design is the first phase, in which software developers design a requirement model after they talk to users of a software system. The requirement quality has an influential impact on the whole system [6].

In this Section, an E-mail client system is used to illustrate how to elicit requirements and build requirement model using the method proposed in this paper. The e-mail client is software that provides the user interface which connects a user to the e-mail server, and allows individuals to send mail using the SMTP [9] protocol and to check mail from server using the POP3 protocol. The use case diagram of the E-mail client system is shown in Fig. 2.

In the use case diagram, each use case is described by use case card and scenarios are constructed. Here, we present a scenario language for describing scenarios [10]. The general advantages of using a scenario language mainly include:

- It is easily to be understood and conveniently employed by software users;
- Syntactic rules and event frame of each event enable the transformation from use case descriptions to TCPN models to be feasible;

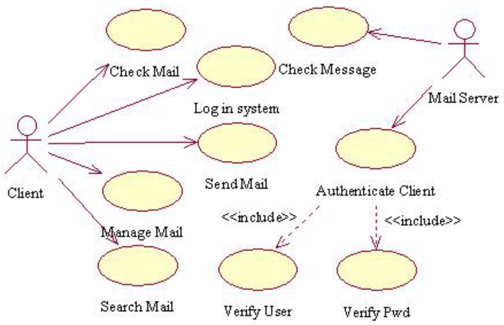

Fig. 2. Use case diagram of mail client system

- Use case card is improved using this scenario language, and method of using syntactic rules can detect the errors (lack of events, extra events, and wrong sequence among events);

The vagueness of the scenario written using this language can be reduced. In several papers such as [10] [11] [12], the similar scenario language has been already introduced, but some differences also exist. So, a brief introduction of this language will be given. A scenario can be regarded as a sequence of events. Events are behaviors employed by users or systems for accomplishing their goals. A set of syntactic rules will be followed while constructing scenarios.

Rule 1: Each event is written using simple sentence based on Subject + Predicate + Object, and each has only one subject and one predicate.

Example:
(1) Client send 'DATA' and 'QUIT' to SMTP server; (false)
Sentence (1) is replaced by some simple sentences:
(1)Client send 'DATA' to SMTP server; (true)
(2)Client send 'QUIT' to SMTP server; (true)
Rule 2: Use active rather than passive voice.

Example:
(1) 'DATA' is sent to SMTP server; (false)
(1)Client send 'DATA' to SMTP server; (true)
Rule 3: Use the same word for same description in different sentences.

Example:
(1)Client send 'DATA' to SMTP server; (true)
(2)E-Mail Client send 'QUIT' to SMTP server; (false)
(2)Client send 'QUIT' to SMTP server; (true)
'Client' and 'E-Mail Client' are the description of the same thing.
Rule 4: Use the name of an object to replace a pronoun of the object.

Example:
(1)Client send 'DATA' to it; (false)
(1)Client send 'DATA' to SMTP server; (true)
Rule 5: Each event has its own event frame.
The Example is shown in Table 2.
Rule 6: Each event ends with a semicolon.
Rule 7: The relations among single events are: sequence, selection, iteration, and concurrency.

Most events occur sequentially, so ordered events can be regarded as sequential events. For selective events, we use "IF-THEN-ELSE {Condition, F1, F2};",If 'Condition' return true, go to F1, otherwise, go to F2. For iterative events, we use "DO-WHILE {Condition, F1, F2 ...};", Do F1, F2 iteratively, until 'Condition' return false. For concurrency events, we use "And {F1, F2, F3 ...};".

We consider a use case card of sending mail using the SMTP protocol in E-mail client system. Table 1 shows the use case card that client sends a mail by SMTP protocol. In the use case card shown in Table 1,'Use case name' is

Table 1. Use case card of 'Send Mail'

Use case name	Send Mail
Primary actors	Client
Base use cases	null
Include use cases	null
Extend use cases	null
Pre-condition	Client click on 'send mail' button;
Post-condition	Client finished 'send mail';
Event flow	
(1) Client establish TCP connection with the SMTP server via port 25; (2) Client send 'HELO' to SMTP server; (3) SMTP server receive request of connection; (4) DO-WHILE{ Condition: username is ok, (4.1) SMTP server send request of username, (4.2) Client send username to SMTP server }; (5) SMTP server send request of password; (6) Client send password to SMTP server; (7) IF-THEN-ELSE{ Condition: password is ok, (7.1) SMTP server send 'OK' to Client, (7.2) SMTP server send message of error }; (8) Client send 'MAIL' and sender's address to SMTP server; (9) SMTP server send 'OK' to Client; (10) Client send 'RCPT' and receiver address to SMTP server; (11) IF-THEN-ELSE { Condition: SMTP server identify receiver address, (11.1) SMTP server send 'OK' to Client, (11.2) SMTP server refuse the request }; (12) Client send 'DATA' and mail content to SMTP server; (13) Client send 'QUIT' to close connection;	

the name of use case in use case diagram. 'Primary actors' means active objects such as human or system appearing in the scenario. 'Base use cases', 'Include use cases', and 'Extend use cases' mean use cases that have relation of 'generalization', 'include', and 'extend' with the use case described in the use case card. 'Pre-condition' specifies a condition that satisfies at the start of the scenario. 'Post-condition' specifies a condition that satisfies at the end of the scenario. 'Pre-condition', 'Post-condition', and 'Event flow' are written using the

Table 2. Example of event frame

Event	
Client send 'HELO' to SMTP server	
Object1	Client
Action	send 'HELO'
Object2	SMTP server
Message	'HELO'
Time Constraint	0

scenario language presented before. Furthermore, 'Event flow' in use case card is analyzed to extract object and message between objects. If needed, time delay of each event will also be added.

Example of event frame is shown in Table 2. 'Object1' and 'Object2' are objects extracted from the event sentence. 'Object1' and 'Object2' are elicited from subject and object of event flow respectively, 'Message' means message between 'Object1' and 'Object2'. 'Time Constraint' is the time delay.

3 Timed and Controlled Petri Net (TCPN)

Petri nets have been used extensively and successfully in various applications such as protocol or performance analysis.

The general characteristics of using Petri-nets mainly include:

- Petri net allows the modeling of concurrency, synchronization and resource sharing behavior of a system.
- Petri net has visual and easily understandable representation.
- Petri net has well-defined semantics and a solid foundation for mathematics.
- Petri net can offer variety of mature analysis techniques, such as reachability, deadlock, bounded, safety, invariant, etc.
- Some software tools are available to assist Petri net modeling and analysis.
- The integration of Petri nets with UML could provide a means for automating behavioral analysis and building platform independent model efficiently in Model Driven Development.

In order to describe the time constraint and to force the firing of one or more transition according to some given condition or system state, we present a Timed and Controlled Petri Net (TCPN), which adds time constraint to Controlled Petri Net. Controlled Petri net is a class of Petri nets with external enabling conditions called control places which allow an external controller to influence the progression of tokens in the net. Controlled Petri net was first introduced by Krogh [14] and Ichikawa and Hiraishi [15]. Following the formalism adopted in defining ordinary Petri nets, we present the definition of TCPN as follows:

Definition 1 (TCPN). *A Timed and Controlled Petri Net (TCPN) is an eight-tuple* $TCPN=\{P, C, T, F, M_0,\lambda,\mu,\gamma\}$ *where*

- $P = \{p_1, p_2, ..., p_n\}$ *is a finite set of state places or places;*
- $C = \{c_1, c_2, ... , c_n \}$ *is a finite set of control places;*
- $T = \{t_1, t_2, ... , t_k \}$ *is a finite set of transitions;*
- $F \subseteq (P\times T)\cup(C\times T)\cup(T\times P)$ *is the flow relation i.e. set of directed arcs;*
- $M_0 : P\rightarrow\{0, 1, 2, 3, ...\}$ *is the initial marking;*
- λ*:* $P\rightarrow \mathbb{N}$ *is the token function of state places;*
- μ*:* $C\rightarrow\{0, 1\}$ *is the token function of control places;*
- γ*:* $T\rightarrow \mathbb{R}^+$ *is the time delay function of transition;*

$P\cap C\cap T=\Phi$ *and* $P\cup C\cup T\neq\Phi$ *i.e. state places, control places, and transitions are disjoint sets;*

Definition 2. *For a transition $t \in T$, the set of input control places is $^{(c)}t = \{c | (c, t) \in F\}$; for a control place $c \in C$, the set of output transitions is $c^{(t)} = \{t | (c, t) \in F\}$; similarly, for a transition $t \in T$, the set of input and output state places are $^{(p)}t = \{p | (p, t) \in F\}$ and $t^{(p)} = \{p | (t, p) \in F\}$; for a state place $p \in P$, the set of input and output transitions are $^{(t)}p = \{t | (t, p) \in F\}$ and $p^{(t)} = \{t | (p, t) \in F\}$.*

Definition 3. *A transition t is state enabled if for all $p \in {}^{(p)}t$,$\lambda(p) \geq 1$; a transition t is control enabled if for all $c \in {}^{(c)}t$,$\mu(c)=1$; a transition t is enabled if and only if t is state enabled and control enabled.*

4 Transforming Use Case Description into Timed and Controlled Petri Net (TCPN)

Use case description including use case card and scenarios written using scenario language has been introduced in Section 2. In this section, we discuss transformation of use case description into Timed and Controlled Petri Net. A Timed and Controlled Petri subnet is defined for each part of the use case card (single event, IF-THEN-ELSE, etc.), and through the interconnection of every subnets, the net which models the whole use case is obtained. Detailed transformation rules are described below.

4.1 Expressing Single Event

Through analyzing the event frame (Table 2), and taking no account of message interactive between objects, single event is transformed into subnet shown in Fig. 3(a)(b). If time constraint of the event exists, timed transition is used (Fig. 3(b)).

The presence of a token inside P0 means that transition T0 is enabled and the action of single event is being executed. Finally, the presence of a token inside place P1 means that the action of this single event is ended.

4.2 Expressing Message between Objects

For some events of the event flow of use case card, message between objects exists, i.e. 'Message' in event frame of the single event is not null. The TCPN subnet used to describe message between objects is shown in Fig.3(c).

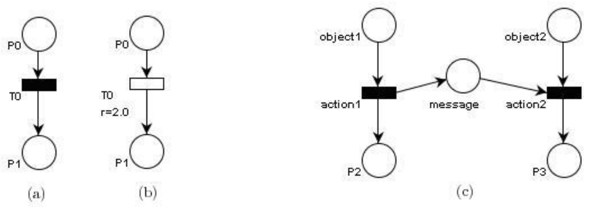

Fig. 3. Petri subnet with single event and message: (a) single event with immediate transition; (b) single event with timed transition; (c) subnet with message

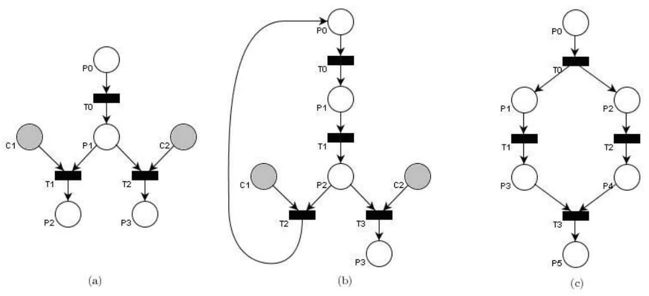

Fig. 4. Petri subnet with ‘IF-THEN-ELSE’ ‘DO-WHILE’ ‘AND’

Token exists inside object1 means that transition action1 is enabled. After firing the transition action1, a token is inserted into place named message. Transition action2 is enabled only if place object2 and message has a token respectively.

4.3 Expressing ‘IF-THEN-ELSE’

The Timed and Controlled Petri subnet which models the ‘IF-THEN-ELSE’ in the event flow described using the scenario language is shown in Fig.4(a).

The firing of transition T0 models the command of starting the check of condition. It inserts a token into place P1 and allows the beginning of checking. When the check ends, a token is automatically inserted into control place C1 or C2 (μ(C1) =1 orμ(C2) =1). It depends on the considered condition has been returned false or true respectively. According to the result of check, one between transition T1 and transition T2 is enabled, the token game of the subnet will then go on with a token inside place P2 (the check is ended and return true) or inside place P3 (the check is ended and return false).

4.4 Expressing ‘DO-WHILE’

The subnet which models the ‘DO-WHILE {Condition, F1}’ in the event flow is shown in Fig. 4(b). As soon as T0 is fired, execution of event flow F1 is started and a token is inserted into place P1. The firing of transition T1 models the command of starting check of the condition. It inserts a token into place P2. When the check ends, control place C1 or C2 will be inserted a token automatically, i.e.μ(C1) =1 orμ(C2) =1. The evolution of subnet is similar to what Fig.4(a) describes. The difference is the directed arc between transition T2 and place P0, when the check is ended and returns true.

4.5 Expressing ‘AND’

Fig. 4(c) shows the subnet which models ‘AND {F1, F2}’. The firing of transition T0 puts a token into both place p1 and place p2. In this way, event F1 and event F2 have to be executed concurrently. Transition T3 can only be fired if both

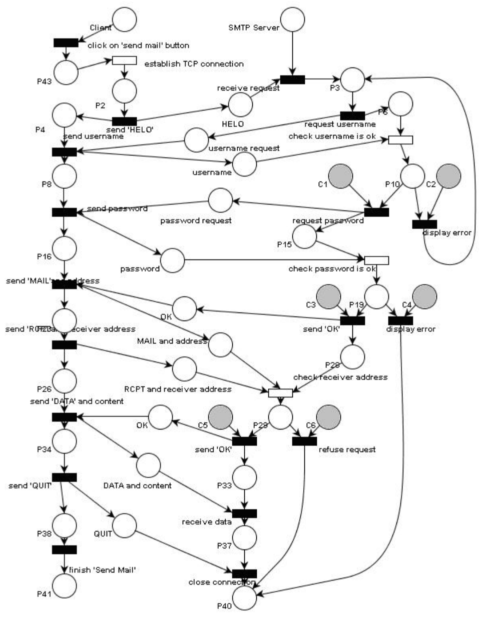

Fig. 5. TCPN model of 'Send Mail'

places (P3 and P4) have tokens. The presentation of a token inside place P5 means the concurrent event is ended.

4.6 TCPN Model of 'Send Mail'

According to the rules which precede this section, the TCPN model of 'Send Mail' (its use case card is described in Section 2) is obtained through the interconnection of some subnets.

The order which TCPN is constructed is: First, transforming 'Pre-condition' of use case description into a subnet; second, transforming 'Event flow' into subnets sequentially; third, transforming 'Post-condition' into TCPN subnet. The TCPN model of 'Send Mail' is shown in Fig. 5.

5 Verification of Use Cases by Analyzing Petri Net Models

A major strength of Petri nets is their support for analysis of properties and problems associated with the systems. So in order to verify the faults of use cases, we will analyze some behavioral properties such as Reachability, Boundedness, Liveness, Reversibility, etc. First of all, it is very necessary to understand what the high quality requirement is. The following criteria for requirement quality assessment are used when verifying requirements by analyzing TCPN model.

1. Completeness: The acquired requirements are completeness if the requirement model established has included all the functions of the system, and all significant requirements or software responses are included or defined.
2. Consistency: Two or more requirements are not in conflict with one another.
3. Correctness: The requirements elicited should satisfy user's demand, and the requirement itself should be correct.
4. Unambiguous: It is necessary to make sure that a requirement has only one meaning in particular context. Because our use case event flows are written following the rules described in Section2, the ambiguity of requirement descriptions can be reduced. Here we will not verify ambiguous information by analyzing Petri nets.

While analyzing the TCPN model for use case, a Petri net tool called PIPE2 (Platform Independent Petri Net Editor) is used for Petri net comparison and generating reachability graph [16]. Verification of use case is described in detail below.

5.1 Verification of 'Completeness'

The requirement is completeness, so Timed and Controlled Petri Net models generated by requirement descriptions should be completeness as well.

Definition 4 (Completeness). *The Timed and Controlled Petri Net is completeness, if*

– *All places and transitions are specified by particular names;*
– *No isolated subnet exists in the TCPN model of each use case;*

The approach for verifying 'Completeness' is: For a TCPN Q={P, C, T, F, M_0,λ,μ,γ}, traverse all places and transitions from initial place: if ∃(p∈P ‖c∈C ‖t∈T), p or c or t cannot be traversed, i.e. there exists an isolated subnet, display ("some message associate with the object is not identified"). If ∃p∈P, p does not have a particular name, display ("objects are not identified in event frame"). If ∃t∈T, t does not have a particular name, display ("'Action' in the event frame is not identified as a name").

5.2 Verification of 'Consistency'

The consistence of requirements is verified by analyzing some properties of TCPN.

Definition 5 (Consistency). *The TCPN models are consistence, if*

– *The TCPN model itself is consistency, i.e. the TCPN is live;*
– *TCPN models of related use cases are consistency, i.e. TCPN model of the use case U is consistent with that of U's include use case; TCPN model of the use case U is consistent with that of U's base use case.*

The approach for verifying ‘Consistency’ is to simulate a Petri net model using PIPE tool [16]: For a TCPN Q={P, C, T, F, M_0,λ,μ,γ}: If Q is dead (L0-live), display (‘some events never happen, deadlock exists’). For use case and ‘Include use cases’ in use case card: If name of ’Include use cases’ does not exist in transition names of TCPN model for a use case, display (“inconsistent between use case and ‘Include use cases’”). For use case and ‘Base use cases’ in use case card:

By comparing between TCPN model of use case and that of its ‘Base use cases’, If number of transitions in TCPN model of ‘Base use cases’ is more than that of the use case, or no identical transitions exist, display (“inconsistent between use case and ‘Base use cases’”).

5.3 Verification of ‘Correctness’

The Correctness of TCPN models for describing use cases is defined below:

Definition 6 (Correctness). *The TCPN models are correctness, if*

- *The reachability graph of TCPN model is correctness;*
- *The TCPN models is boundness;*
- *The time delay of the transition of TCPN models is valid.*

The method for verifying ‘Correctness’ is: For a TCPN Q={P, C, T, F, M_0,λ,μ,γ}: If the correct trace is a→b→c→d, errors of omission result in a→b→d or b→c→d; errors of extra event result in a→b→c→c→d or a→b→c→x→d; errors of reversal event results in a→c→b→d. these errors are verified by analyzing the reachability graph of TCPN after giving the initial marking of state places and the token number of control places. Fig.6(a) shows the reachability graph of TCPN model shown in Fig.5 when λ(Client) =1, λ(SMTP server) =1, μ(C2) =1, and Fig.6(b) shows the reachability graph of the TCPN model when λ(Client) =2, λ(SMTP server) =2, μ(C2) =1. If Q is not bounded, display (‘maybe overflow exists’). If $\exists$t$\in$T, γ(t)<0, display (‘time constraint identified in event frame is invalid’).

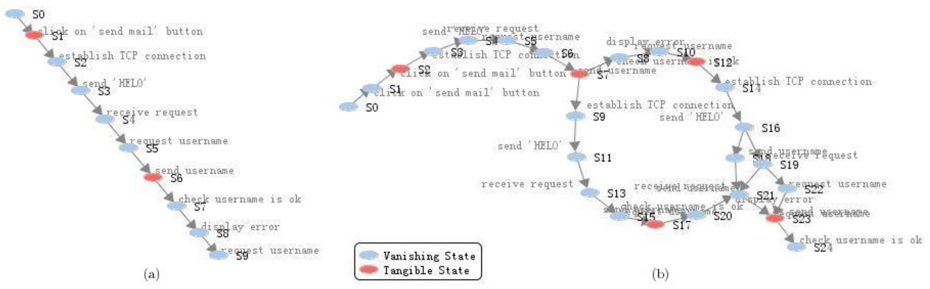

Fig. 6. Example of reachability graph of TCPN

6 Conclusion and Future Work

The use case approach, based on the concept of scenarios, is one of the best known and most widely used requirements elicitation techniques in the industrial application. In this paper, use cases in use case diagram are used to elicit the user requirements. Use case cards are used to describe use case in detail. Especially, the use case should be correct and valid. Therefore, it is important to enable verification of the use case. For instance, to detect some inconsistent information or incorrect fact that is hidden in use case description of use case. However, as lacking in formal syntax and semantics and being difficult to analyze the dynamic behavior, directly verification of use case is very difficult. So, Timed and Controlled Petri Net (TCPN) is proposed to formalize the use case description, then, use case can be verified by analyzing the TCPN model. According to criteria for requirement quality assessment, rules and methods are proposed as a verification mechanism to detect the faults or errors such as missing information, inconsistent information, incorrect fact, etc.

We demonstrate our approach using an E-mail client system and focus on the use case “Send Mail”, i.e. user send an email using the Simple Mail Transfer Protocol (SMTP). By analyzing the TCPN of use case “Send Mail”, faults and errors are detected. Experiments show that our approach can help developers to minimize discrepancy and reduce errors of requirement analysis.

Although our research have proposed an improvement in formalization and verification of use case, there also exist a lot of issues that are worthy of further research. First of all, use case is verified using only a few analyzing technique of Petri nets, so future research on analyzing other properties of Petri nets is needed to detect more faults. Secondly, our research here mainly focuses on modeling and verification of Functional requirements, by the research on extending of TCPN to modeling and verification of requirements appending nonfunctional requirements to functional requirements, software performance will be improved. At last, a case tool is needed to support the requirement elicitation, modeling and verification based on TCPN.

References

1. Bastani, B.: A Requirements Analysis Framework for Open Systems Requirements Engineering. ACM SIGSOFT Software Engineering Notes 32, 1–19 (2007)
2. Lee, W.J., Cha, S.D., Kwon, Y.R.: Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering. IEEE Transactions on Software Engineering 24(12), 1115–1130 (1998)
3. Shen, W., Liu, S.: Formalization, Testing and Execution of a Use Case Diagram. In: Dong, J.S., Woodcock, J. (eds.) ICFEM 2003. LNCS, vol. 2885, pp. 68–85. Springer, Heidelberg (2003)
4. Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y., Chen, C.: Formal Approach to Scenario Analysis. IEEE Software, 33–41 (March 1994)
5. Andersson, M., Bergstrand, J.: Formalizing Use Cases with Message Sequence Charts. Master’s thesis. Lund Inst. of Technology (1995)

6. Shen, W., Guizani, M., Yang, Z., Compton, K., Huggins, J.: Execution of A Requirement Model in Software Development. In: Proceeding of IASSE 2004, Nice, France, pp. 203–208 (2004)
7. Li, X., Liu, Z., He, J.: Formal and Use-Case Driven Requirement Analysis in UML. In: 25th Annual International Computer Software and Applications Conference, 2001. COMPSAC 2001, Chicago, IL, USA, pp. 215–224 (2001)
8. OMG. Model Driven Architecture (MDA)- document number ormsc/2001-07-01 (2001)
9. Postel, J.B.: RFC 821: Simple Mail Transfer Protocol (1982)
10. Toyama, T., Ohnishi, A.: Rule-based Verification of Scenarios with Pre-conditions and Post-conditions. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering 2005, Paris, pp. 319–328 (2005)
11. Ohnishi, A., Hui, Z.H., Fujimoto, H.: Transformation and Integration Method of Scenarios. In: Proceedings of 26th IEEE International Computer Software and Applications Conference (COMPSAC 2002), Oxford, England, August 26-29, pp. 224–229 (2002)
12. Li, L.: A Semi-Automatic Approach to Translating Use Cases to Sequence Diagrams. In: TOOLS 1999: 29th International Conference on Technology of Object-Oriented Languages and Systems, pp. 184–193. IEEE Computer Society, Nancy (1999)
13. Murata, T.: Petri nets: Properties, analysis, and applications. Proceeding of the IEEE 77(4), 541–580 (1999)
14. Krogh, B.H.: Controlled Petri Nets and Maximally Permissive Feedback Logic. In: Proceeding of the 25th Annual Allerton Conference, pp. 317–326 (1987)
15. Ichikawa, A., Hiraishi, K.: Analysis and Control of Discrete Event Systems represented by Petri Nets. In: Discrete Event Systems: Models and Applications. LNCS, vol. 103, pp. 115–134. Springer, New York (1988)
16. PIPE2, Platform Independent Petri net Editor 2, `http://pipe2.sourceforge.net`
17. Lanubile, F., Shull, F., Basili, V.R.: Experimenting with Error Abstraction in Requirements Documents. In: Proceeding of the 5th international symposium on software metrics, pp. 114–121 (1998)
18. Campos, J., Merseguer, J.: On the Integration of UML and Petri Nets in Software Development. In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 19–36. Springer, Heidelberg (2006)
19. Boccalatte, A., Giglio, D., Paolucci, M.: A CASE Tool for Information System Project and Development. In: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Tokyo, Japan, vol. 3, pp. 1042–1047 (1999)
20. Lee, J., Pan, J.-I., Kuo, J.-Y., Fanjiang, Y.-Y., Yang, S.: Towards the Verification of Scenarios with Time Petri-Nets. In: Proceeding of the 24th Annual International Conference on Computer Software and Application, Taipei, Taiwan, pp. 503–508 (2000)