Proceedings of the International MultiConference of Engineers and Computer Scientists 2010 Vol II,
IMECS 2010, March 17 - 19, 2010, Hong Kong

A New Approach To Measure Software Security

Wei Han, Hairong Ye, Zuohua Ding *

Abstract—**A new method is proposed to measure software system security. Each component can be characterized by a composite net which combines the features of Finite State Machine and Operation Diagram. A fuzzy number is introduced to this net to represent uncertain elements that might affect the software security, and thus we construct Security Function for each component. The whole system Security Index is calculated based the Reliability Graph and the Security Function for each component, where Reliability Graph is used to reconstruct the component relationship of the system.**

Keywords: Software security, composite Petri net, fuzzy number, security measure function

1 Introduction

To ensure software security, we need to pay attention to at least three phrases in software development process: 1) design phrase, 2) implementation phrase, and 3) system testing phase. A lot of work has been done in the first phrase. The second phase is to realize the security properties in the design. Because of the diversities of the programming languages, in this phase, to minimize the risk of harm is mostly depending on the experience of the programmers, so far few works appeared in the literature. The third phase, also the last phrase, will determine the security quality of the software to some extent. It should check sever aspects such as interacting with system resource, responding to the inputs, cooperating with other subsystems, and surviving in the outside attack. It is a heavy task to test the security in this phrase. Fortunately, some tools have been developed which can statically scan the source code to catch security vulnerabilities [9] or employ a state based approach to detect anomalies in the program [8].

In this paper, we suggest a method to measure how secure of a developed software in the third phase. This method will provide us an index which can tell us in the running time that which part of the system is in normal state, warning state, critical state, and dying state. The motivation comes from a trivial example(actually some people use it as an example to study the security issues).

A user faces an Id Checking if he/she wants to access a database. The state design may be like that in Fig. 1. There are three states here: Logout, Login and Access. If Id Checking is successful, the state will change from Logout to Login; If other commands to database are correct, the state will change from Login to Access.

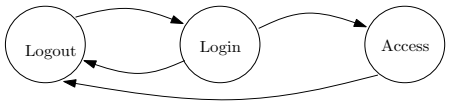

Figure 1: Login and access database

If Id Checking returns a wrong Id in the first time, we might say that the user made a mistake; If Id Checking returns a wrong Id again, we may say this user is so careless; If the Id Checking keeps return wrong ID in the third time, fourth time, etc, we have reason to say that this user might be a potential attacker. If by accident, in the tenth time, this user successfully passes the Id Checking, then the database is accessible to this user. However, this access is not normal. This access carries some unsafe elements to the database. If this user continues to access the critical part of the system, there is a risk to some extent that this user may screw the system. we do have the potential security issue here even through there is no mistake in the design, nor the coding.

Instead of studying audits of the system, in this paper we investigate software itself, since all the traces left in the system are the refections of the interactions of the software with the system and users. We build a model that combines finite state machine with function calls(method invocation) to form a Composite Petri Net. A fuzzy number is created for this net when it is running, which signals the failure possibility of the specified component of the system. A security function is defined based on this fuzzy number. The whole system security index is then defined based on this security function and the system structure. The advantage of our method is that we can determine the vulnerable parts of the software which is impossible by statically checking the source code.

This paper is organized as the following. Section II defines Composite Net. Section III constructs Security Functions for composite nets. In section IV, we define rules to translate components to composite nets. In Section V, we define Security Index of the system and study its properties. Section VI is to compute distribution func-

*The authors are with the Center of Math Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, Zhejiang 310018, P. R. China. E-mail: zouhuading@hotmail.com.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

tions for composite Petri net. Section VII is the summary and discussion of the paper.

2 Composite Nets

Definition 2.1 *A Petri Net is a tuple* $PN =< P, T, A_{pre}, A_{post}, w >$, *where*

1. $P = \{p_1, p_2, ..., p_n\}$ *is a finite nonempty set of places,*
2. $T = \{t_1, t_2, ..., t_m\}$ *is a finite nonempty set of transitions,*
3. $A_{pre} = \{p \to t\}$ *is a set of directed arcs which connect places with transitions,*
   $A_{post} = \{t \to p\}$ *a set of directed arcs which connect transitions to places,*
4. $w : A_{post} \to (0, \infty)$ *is a mapping to assign a weight to each arc,*

Definition 2.2 *A Timed Petri Net is a tuple* $TPN =< PN, d, r >$, *where*

1. *PN is a Petri net,*
2. $d : T \to 2^{R+}$ *is a mapping to assign a firing time interval to each transition.*
3. $r = \{r_1, r_2, \ldots, r_n\}$ *is a distribution function set with* r_i *defined on* $r(t_i)$ *and* $r_i : d(t_i) \to [0, 1]$.

Note that the Timed Petri net is equivalent to a Petri net if all the minimum time of the time intervals are 0 and all the maximum times are set to ∞. For each transition t, let $I(t)$ and $O(t)$ be the input and output places, respectively. For each place p, let $I(p)$ and $O(p)$ be the input and output transitions, respectively.

We will assign variables to Petri net and Timed Petri net to record the action accumulation which might lead to affect the security of the system.

Definition 2.3 *A Tagged Petri Net is a 2-tuple* (N, V) *where*
- N is a Petri Net,
- $V = (x_1, x_2, \ldots, x_n)$ *is a variable set, each of them has been assigned to a place* P_i *such that* $x_i : P_i \to [0, 1]$.

Definition 2.4 *A Tagged Timed Petri Net is a 2-tuple* (N, V) *where*
- N is a Timed Petri Net,
- $V = (x_1, x_2, \ldots, x_n)$ *is a variable set, each of them is time dependent and has been assigned to a place such that* $x_i : (P_i, t) \to [0, 1]$ *or simply* $x_i(t) \in [0, 1]$.

For a transition of t in (Timed) Petri net, it is enabled only all places in $I(t)$ have tokens. Only enabled transitions can be fired. For non-timed Petri net, we assume that as soon as a transition gets enabled, it starts to fire. For Timed Petri net, the transition can get fired at any time in the time interval associated to this transitions.

Firing is based on three indivisible primitives: 1) Removal of the tokens from the input places and insertion of the tokens in the output places (two transitions which do not share any places can be fired independently). 2) Determination of the new mark distribution in both input and output places after firing. 3) Each local variable will update its values.

The first two operations are the same as the regular Petri net processing. We omit the details and focus on the third operation. There are several situations here ($\wedge$ represents minimum and $\vee$ represents maximum):

1. 1 Non-timed Petri net.

   a) One input place. There is no time interval attached to the transition. Variable x_1 is with place P_1 and variable x_2 is with place P_2, respectively. A constant k is associated with the *arc* from transition t to place P_2. When token moves from place P_1 to place P_2, the value of variable x_2 is $x_2 = kx_1$. See Fig. 2(a).
   b). More input places. See Fig. 2(b). When firing, variable x_3 will update its value by picking the minimum of x_1 and x_2 as the following:

   $$x_3 = k \wedge_i x_i.$$

2. Timed Petri net.

   a) One input place. See Fig. 2(c). Assume an interval $[t_1, t_2]$ is attached to the transition t and a distribution function r is defined on this interval. Variable x_1 is with place P_1 at time α. If enabled, transition will be fired at any time between t_1 and t_2. Assume that the firing occurs at time $t_0 + t_1 \in [t_1, t_2]$, then the value of x_2 in place P_2 at time $t = \alpha + t_0$ is

   $$x_2(t) = x_1(\alpha) r(t_0).$$

   b) More input places. See Fig. 2(d). Assume that the firing occurs at time $t_0 + t_1 \in [t_1, t_2]$, then the value of variable x_3 of place P_3 at time $t = \vee\alpha_i + t_0$ is

   $$x_3(t) = \wedge_i x_i (\vee_i \alpha_i) r(t_0).$$

Since the firing rules are the same as those in ordinary Petri nets. The resulting tagged (timed) Petri net is live, safe, and/or reversible if and only if the original Petri net is live, safe, and/or reversible, because all these properties are dependent upon the firing mechanisms of transitions in the net. Noticing that the enabling conditions of

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

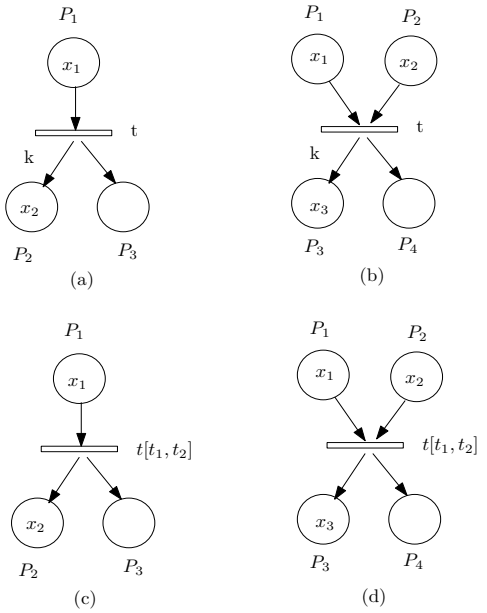

Figure 2: Tagged Petri net

transitions only depend on the markings of input places, the properties of liveness, safeness, and reversibility are unchanged while the tagged Petri net is mapped from ordinary Petri net or the tagged Petri net is mapped back to the ordinary Petri net.

Definition 2.5 *A Composite Net (CN) is a net that either the places of Tagged time PN contain Tagged PN or the places of Tagged PN contain Tagged time PN. The net inside is called local net. The net outside is called global net.*

For example Fig. 3 is a composite net, the global net is a Tagged TPN and the local nets are Tagged PN. The global net has variables y_1, y_2 and y_3. Its first state, *state1* contains a local net with variable x_1 and x_2, *state2* and *state3* also contain local nets with variables x_1 and x_2, respectively.

There are two properties for composite nets:

Property 1. The local net may or may not passes its tokens to the global net. In Fig. 3, the local net of *state1* will pass its token to *state2* through transition t_1, while the local net of *state2* will not pass its token to *state3*.

Property 2. The variable of global net will always take the most recent values from the variables in local net. For example, In Fig. 3, *state1* has a global y_1 and its local net has two variables x_1 and x_2. When the transition is enabled, the value of x_2 will get updated, meanwhile state changes from state1 to *state2*. At this moment, $t = 0$, x_2 will pass its value to y_1 as follows:

$$y_1(0) = x_2.$$

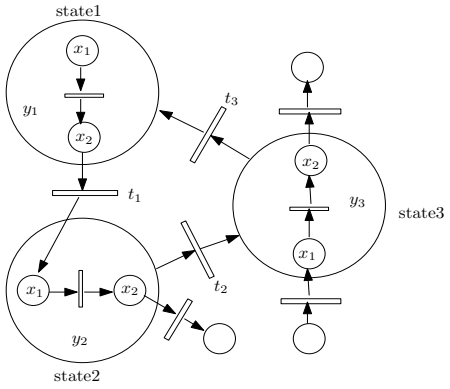

Figure 3: Composite Net

3 Security Function of Composite Net

In our discussion, we need some fuzzy concepts. A *fuzzy set* A in a universe of discourse $X = \{x\}$, written A in X, is defined as a set of pairs

$$A = \{(\mu_A(x), x)\}$$

where $\mu_A : X \to [0, 1]$ is the *membership function* of A and $\mu_A(x) \in [0, 1]$ is the *grade of membership* (or membership grade) of $x \in X$ in A. A *fuzzy number* is defined as a fuzzy set in R, the real line, which is assumed(usually) normal and convex(with a convex membership function). For more details about fuzzy numbers, we refer to [5]. Let E^1 be the set contains all the fuzzy numbers on R^+.

Definition 3.1 *Let* $A_i = \{(\mu_{A_i}(x_i), x_i)\} \in E^1$. *If* $\cup_{x_i \in A_i} x_i$ *forms an interval in* R^+, *then the fuzzy set*

$$T = \cup_i(\mu_{A_i}(x_i), x_i)$$

is called an extended fuzzy number. Let $\bar{E}^1$ *be the set contains all the extended fuzzy numbers on* R^+.

For a composite net, the extended fuzzy number $\mathbf{T}$ may take values from global net variables as its membership grade. We still use Figure 3 as the example. Assuming the time intervals attached to the transitions t_i are $[a_i, b_i], i = 1, 2, 3$ and their distributions are r_i. If the token leaves *state1* and takes $\tilde{t}_1^0$ to get *state2*. Then

$$\begin{aligned} \mu_{\mathbf{T}}(t) &= y_1(0),\ t \in [0, \tilde{t}_1^0],\ \tilde{t}_1^0 + a_1 \in [a_1, b_1], \\ y_2(\tilde{t}_1^0) &= r_1(\tilde{t}_1^0) y_1(0). \end{aligned}$$

Similarly, we obtain that

$$\begin{aligned} \mu_{\mathbf{T}}(t) &= y_2(\tilde{t}_1^0),\ t \in [\tilde{t}_1^0, \tilde{t}_2^0],\ \tilde{t}_2^0 + a_2 \in [a_2, b_2] \\ y_3(\tilde{t}_2^0) &= r_2(\tilde{t}_2^0) y_2(\tilde{t}_1^0) \\ \mu_{\mathbf{T}}(t) &= y_3(\tilde{t}_2^0),\ t \in [\tilde{t}_2^0, \tilde{t}_3^0],\ \tilde{t}_3^0 + a_3 \in [a_3, b_3], \\ y_1(\tilde{t}_3^0) &= r_3(\tilde{t}_3^0) y_1(\tilde{t}_2^0). \end{aligned}$$

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Generally, we have

$$\mu_{\mathbf{T}}(t) = y_1(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i)),$$
$$t \in [\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i), \sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1}],$$
$$\tilde{t}_1^{i+1} + a_1 \in [a_1, b_1],$$
$$y_2(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1})$$
$$= r_1(\tilde{t}_1^{i+1}) y_1(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i));$$

$$\mu_{\mathbf{T}}(t) = y_2(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1}),$$
$$t \in [\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1},$$
$$\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1} + \tilde{t}_2^{i+1}],$$
$$\tilde{t}_2^{i+1} + a_2 \in [a_2, b_2],$$
$$y_3(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1} + \tilde{t}_2^{i+1})$$
$$= r_2(\tilde{t}_2^{i+1}) y_2(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1});$$

$$\mu_{\mathbf{T}}(t) = y_3(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1} + \tilde{t}_2^{i+1}),$$
$$t \in [\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1} + \tilde{t}_2^{i+1},$$
$$\sum_{i=0}^{k+1}(\tilde{t}_1^i + t_2^i + t_3^i)], \tilde{t}_3^{i+1} + a_3 \in [a_3, b_3],$$
$$y_1(\sum_{i=0}^{k+1}(\tilde{t}_1^i + t_2^i + t_3^i))$$
$$= r_3(\tilde{t}_3^{i+1}) y_3(\sum_{i=0}^{k}(\tilde{t}_1^i + t_2^i + t_3^i) + \tilde{t}_1^{i+1} + \tilde{t}_2^{i+1});$$

$$k = 0, 1, 2, \ldots.$$

Definition 3.2 *For a given composite net CN, if the extended fuzzy number* $\mathbf{T}$ *takes values from its global net variables as its membership grade, then* $x : x(t) = \mu_{\mathbf{T}}(t)$ *is called the Security Function of CN.*

Let $\mathbf{T}$ be an extended fuzzy number with membership function $\mu_{\mathbf{T}}(t) \in [0, 1], t \in [0, \infty)$. Assume that this fuzzy number represents a fuzzy data on reliability of a composite net at time t. Therefore, at $t = 0$,

$$\mu_{\mathbf{T}}(t) = 1, \qquad \text{for} \quad t = 0.$$

Thus we can write a "law of possibility of failure" as

$$\Pi(t) = 1 - \mu_{\mathbf{T}}(t).$$

Here the possibility law $\Pi(t)$ gives the possibility of failure occurring at time t. One should note that $\mathbf{T}$ is not necessarily a unimodal fuzzy number, that is, $\mu_{\mathbf{T}}(t)$ may have a flat at the top for $\mu_{\mathbf{T}}(t) = 1$. Now consider θ as the time to failure; that is, the time when a composite net has failed. At this time $\Pi(t = \theta) = 1$ and, since $\mu_{\mathbf{T}}$ is not necessarily unimodal, θ may be over an interval $[\theta_1, \theta_2]$.

4 From Component To Composite Net

This section we will determine the state variable,or Survival Function, for each component. There are two concepts involved in this discussion: Finite State Machine and Operation diagram. The Finite State Machine represents the state-dependent aspects of the use case. The Operation diagram is to picture the active-passive relationship among function calls(method invocations) [4].

4.1 Finite State Machine

A **state** represents a recognizable situation that exists over an interval of time. Whereas an **event** occurs at a point in time, a **Finite State Machine** is in a given **state** over some interval of time. The arrival of an event at the Finite State Machine usually causes a transition from one state to another. Alternatively, an event can have a null effect, in which case the Finite State Machine remains in the same state. A component can be modeled by means of a Finite State Machine.

A Finite State Machine can be well characterized by a timed Petri Net based on the defined rules. We assign a time interval to the transition to indicate the time that needed for state changing and a distribution function corresponds to this time. For the resulted Timed Petri net, a variable is attached to each place representing state change. This variable will take values from 0 to 1.

4.2 Operation Diagram

If an event happens, at least a function will be called or a method be invoked. As long as a sequence functions(methods) have been called(invoked), data changes from one point to another point. Without exception, all these data are either the inputs or the outputs of functions(methods). Generally speaking, the parameters are the inputs and the return values are the output. By analyzing the source code, all these information can be pictured in a diagram, called Operation Diagram, on which the function(method) behaves like a bridge between input and output. We need to introduce two new concepts for Operation Diagram: Selection and Aggregation. Selection is used when an input serves different functions(methods) and aggregation is used when more than one inputs serve one functions(methods)at the same time. If we speak in programming language, the Selection is the statements such as if. . . else, switch, etc.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

In order to use Petri net to simulate the data changes, we define some basic rules to translate an Operation Diagram to a Petri net. There are three cases here: (a) One input to one function: this function will attached to the transition; (b) One input to more than one function: Selection operation is needed to attached for the transition; (c) More than one input to one function: An Aggregation operation is needed for this transition.

For the resulted Petri net, we will assign a variable to each place. All these variables will take crisp positive numbers as their values. Some pre-defined positive values are also attached to the arcs from transition to places based on our needs.

Now for each component we have two kinds of Petri net: The tagged timed Petri net from Finite State Machine and the tagged Petri net from Operation Diagram. It is naturally to combine these two together to form a composite net, with tagged Petri net as local net and tagged timed Petri net as global net. Thus, we may use component and composite net alternatively.

5 Security Index of The System

We assume that all communication between components are through message passing. The communication type can be classified as Synchronous and Asynchronous. Synchronous message passing means that the sending operation needs to wait until an acknowledgment is received. Asynchronous message means that the sending operation can proceed without waiting for the message to arrive at its destination. In both cases, the receiving side is blocked. Thus, the relationship of components can be classified as *sequential* and *parallel* corresponding to Synchronous and Asynchronous, respectively. See Fig. 4.

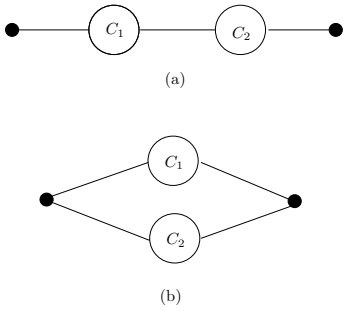

Figure 4: Component relationship type: (a) Sequential, (b) Parallel.

Let $\{\alpha_{lower}, \alpha_{up}, \alpha^i_{lower}, \alpha^i_{up}, i = 1, 2, \ldots, n\}$ be a set of positive numbers. Consider a system ξ composed of n components: $e_1, e_2, \ldots, e_n$, where component e_i is associated with a survival function x_i such that at time t, $x_i(t) \in [0, 1]$. If $x_i(t) > \alpha^i_{up}$, then we say that component e_i is in secure state at time t. If $x_i(t) < \alpha^i_{lower}$, then e_i in un-secure state at time t.

Now we define the secure state of two components. If they are sequential, then we define their secure state at time t as $(x \otimes y)(t)$:

$$(x \otimes y)(t) = x(t)y(t).$$

If they are parallel, then we define their secure state at time t as $(x \oplus y)(t)$:

$$(x \oplus y)(t) = 1 - (1 - x(t))(1 - y(t)).$$

Thus the secure state of the whole system is a function of $x_1, x_2, \ldots, x_n$. Let this function be γ, then for $t \in [0, \infty)$:

$$\gamma(x_1, x_2, \ldots, x_n)(t) = \gamma(x_1(t), x_2(t), \ldots, x_n(t)).$$

This γ is called Security Index of the system with n components.

Theorem 5.1 $\gamma(x_1(t), x_2(t), \ldots, x_n(t) \in [0, 1], \ t \in [0, \infty)$

For the given number $\alpha_{lower}, \alpha_{up}$, at time t, if $\gamma(x)(t) > \alpha_{up}$, then the system is in secure state; If $\gamma(x)(t) < \alpha_{lower}$, then the system is in un-secure state.

Sometimes we use positive integers to label the priority of the communication between components. The smaller the number, the higher the priority. Based on the component relationship, we can translate the component diagram to a new graph, called *Reliability Graph*. Based on this graph, we can calculate the survival index of the system.

For example, if a system consists of 3 components: e_1, e_2 and e_3. Component e_2 communicates with e_3 by both synchronous and asynchronous messages. These two components together then communicate with e_1 by synchronous message. Fig. 5(a) describes all these realtions of this system. We use number 1 to label the priority of the communication between component e_2 and e_3 and use number 2 to label the priority of the communication between component e_1 and e_2. Fig. 5(b) is the *Reliability Graph* of this system.

If the security functions for e_1, e_2 and e_3 are x_1, x_2 and x_3, respectively, then from Fig. 5(b) the security index of the system is

$$\begin{aligned}\gamma(x_1, x_2, x_3) &= x_1 \otimes ((x_2 \otimes x_3) \oplus (x_2 \oplus x_3)) \\ &= x_1(1 - (1 - x_2x_3)) \\ &\quad (1 - (1 - (1 - x_2)(1 - x_3))) \\ &= x_1x_2x_3 - x_1x_2x_3^2 - x_1x_2^2x_3 + x_1x_2^2x_3^2.\end{aligned}$$

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

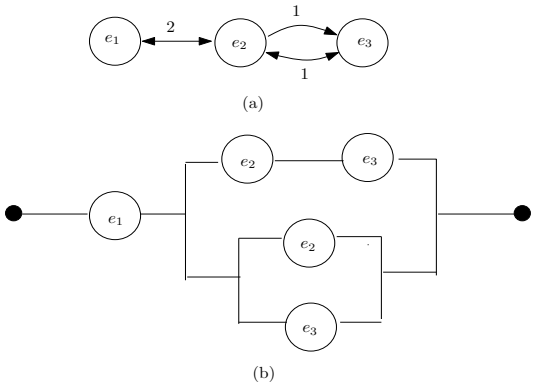

Figure 5: Example of a system: (a) System components, (b)Reliability Graph

6 Computing The Distribution Functions For Composite Petri net

In our composite Petri net, we need to compute the distribution functions for the transitions of the global net in the composite Petri net. These transitions are responsible to pass or receive messages between components. Thus for each transition, the distribution function is defined as the probability that this transition will be failure. To describe the failure behavior of the transition, we will employ enhanced non-homogeneous Poisson Process(ENHPP) model [3]. Each transition of the global net will be associated with a failure intensity function, which can be determined via the coverage testing and fault density analysis [6].

The ENHPP model is based on the following expression for its failure intensity function:

$$\lambda(t) = \frac{dm(t)}{dt} = a\frac{dc(t)}{dt} = ac'(t)$$

where $m(t)$ is the expected number of faults detected by time t, $c(t)$ is called coverage function and a is defined as the total number of faults which are expected to be detected given infinite testing time and complete test coverage, i.e., when $lim_{t\to\infty}c(t) = 1$.

To determine the faults for the reliability of each transition, we have used a method like that proposed by Delamaro et. a1 [1] for integration testing. The difference is that their method is used for the interfaces of function calls, reference to a value returned from a function, and reference to global variables shared by two or more functions, while ours is used for the interface of message passing. To obtain test suite, we developed a method called Condition Calculation in [2].

7 Summary and Discussion

A new method has been proposed to measure the security of software system. Our composite net has some similarities with Bigraph developed by Milner [7]. Bigraph is a graph whose nodes may be nested, representing locality, independently of the edges connecting them. It has been shown that Bigraph can represent dynamic theories for the π-calculus, mobile ambients and Petri nets in a way that is faithful to each of those models of discrete behaviour. In the future we may use Bigraph to model the mobil system and then study the security measure.

Acknowledgments

This work is partially supported by NSF of China under No.90818013, and Zhejiang NSF under Grant No.Z1090357.

References

[1] M. Delamaro, J. Maldonado, and A. P. Mathur, Integration testing using interface mutations, In *Proceedings of the Seventh International Symposium on Software Reliability Engzneering*, New York, pp.112-121, October 30-November 2, 1996.

[2] Z. Ding, K. Zhang and Jueliang Hu, A rigorous approach towards test case generation, *Information Sciences*, vol. 178, pp.4057-4079, 2008.

[3] S. S. Gokhale, T. Philip, P. N. Marinos, and K. S. Trivedi, Unification of finite failure non-homogeneous poisson process model through test coverage, *Proc. Intl. Symposium on Software Reliability Engineering (ISSRE'96)*, October 1996, pp.289-299, NY.

[4] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications With UML, Addison-Wesly, 2000.

[5] A. Kaufmann and M. M. Gupta, Fuzzy mathematical models in engineering and management science, North-Holland, 1988.

[6] M. Lipow, Number of faults per line of code, *IEEE Transactions on Software Engineering*, vol.8, no.4, pp.437-439, 1982.

[7] R. Milner, Bigraphs for Petri nets, in: *Proceedings of the Advanced course in Petri nets (Eichstaett'03)*, *LNCS* 3098, 2004.

[8] C. C. Michael and A. Ghosh, Simple, State-Based Approaches to Program-Based Anomaly Detection, *ACM Transactions on Information and system Security*, vol.5, no.3, pp.203-237, August 2002.

[9] J. Viega, J.T. Bloch, T. Kohno, and G. Mcgraw, Token-Based Scanning of Source Code for Security Problems, *ACM Transactions on Information and system Security*, vol.5, no.3, pp.238-261, August 2002.

ISBN: 978-988-18210-4-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)