IEICE TRANS. INF. & SYST., VOL.E97–D, NO.9 SEPTEMBER 2014

PAPER

Investigating System Survivability from a Probabilistic Perspective

Yongxin ZHAO[†], **Yanhong HUANG**[†], **Qin LI**[†a)], ***Nonmembers***, **Huibiao ZHU**[†], ***Member***, **Jifeng HE**[†], **Jianwen LI**[†], ***and*** **Xi WU**[†], ***Nonmembers***

SUMMARY Survivability is an essential requirement of the networked information systems analogous to the dependability. The definition of survivability proposed by Knight in [16] provides a rigorous way to define the concept. However, the Knight's specification does not provide a behavior model of the system as well as a verification framework for determining the survivability of a system satisfying a given specification. This paper proposes a complete formal framework for specifying and verifying the concept of system survivability on the basis of Knight's research. A computable probabilistic model is proposed to specify the functions and services of a networked information system. A quantified survivability specification is proposed to indicate the requirement of the survivability. A probabilistic refinement relation is defined to determine the survivability of the system. The framework is then demonstrated with three case studies: the restaurant system (RES), the Warship Command and Control system (LWC) and the Command-and-Control (C2) system.

key words: *survivability, probability programs, probabilistic refinement, survivability specification*

1. Introduction

The term *survivability* employed by the industrial community emerges with the development of the networked information systems [28]. The notion of survivability is originated from the weapon system engineering [2], [34] and then regarded as a crucial property for the networked information systems. The survivability property is analogous to the dependability characteristics such as reliability, availability, and security described by Avizienis et al. [1]. Informally speaking, the notion of survivability captures the requirement that the system could perform harmoniously with various operating environments in face of external attacks, failures, accidents or internal errors. Furthermore, a survivable system may sacrifice some secondary functionalities temporarily to retain the essential primary functionalities.

The dependability specifies the requirements in terms of availability, reliability, safety, confidentiality, integrity and maintainability. Like the dependability, the survivability also describes the requirement of a system to avoid failures. However, the dependability only focuses on the behavior of the system in a certain and normal environment. It does not consider the behavior of the system under different environments. The notation survivability emphasizes that the survivable system can provide different forms of services and each service meets the corresponding dependability requirements [10], [15] under different environments. Essentially, a complete and precise definition of survivability should consider three critical characteristics [16], i.e., various environments, alternate forms of services and the probability of availability of each service.

Knight made a comprehensive discussion about system survivability in [16], [17]. He presented a survivable specification involving three characteristics mentioned above and gave a rigorous definition of what a survivable system is, i.e., a system is survivable if it complies with its survivability specification. This precise definition of the survivability above is a significant and elegant evolution. But there are still some problems need to be settled.

- In Knight's work, the environment is classified to several factors such as political climate, conflict dispersion and security threat status in LWC systems. This classification, however, is not necessary and rises the complexity of reasoning about the system behavior. In fact, a correspondence between a certain environment and the configuration of the system is sufficient for the analysis.
- In Knight's specification, a service can satisfy a requirement with a probability. But this probability is not computable without a behavioral model of the service.
- Knight defines the survivability specification but does not provide a computable approach to determine whether a system is survivable with respect to a series of operating environments, i.e., how to determine a given system model complies with the survivability specification.

The need for a precise and adequately comprehensive formal definition of survivability as well as a computable approach for verifying survivability is critically essential not only from an academic view point but also in an engineering sense. In this paper, we revise the survivability specification with a formal framework. The system behavior is specified with a formal probabilistic model. A probability refinement theory [21] is employed to verify whether a system complies with the survivability specification. With the refinement theory, one can compare two similar services and indicate how a system satisfies a survivability specification, i.e., a system is survivable if it is a refinement to a survivability

Manuscript received September 24, 2013.
Manuscript revised April 16, 2014.

[†]The authors are with the Shanghai Key Laboratory of Trustworthy Computing, Software Engineering Institute, East China Normal University, 3663 Zhongshan Road (North), Shanghai, China, 200062.
a) E-mail: qli@sei.ecnu.edu.cn (Corresponding author)

DOI: 10.1587/transinf.2013EDP7339

specification. The various operating environment is modeled with a probabilistic distribution over a succinct environment set so that the concrete and complicated environmental factors can be abstract away. The service model also contains a set of core functionality to specify the concept of an acceptable service, i.e., the core functionalities should be preserved under every operating environment.

According to Knight's specification, the model of a survivable system is divided into two layers, i.e., the function layer and service layer. The functions provides specific functionalities while a set of alternative services provides the possibility to preserve core functionality under various operating environments. Based on this perspective, we also formalize the system as two layers. The functions are specified with guarded probabilistic programs and a service combines a set of functions with alternative implementations. The system behavior is constructed by a probabilistic choice from a set of accepted services regarding to a various operating environment.

In summary, the main contributions of our work include:

- **Probabilistic Computation Model.** Modeling system behavior with probabilistic programs [24] facilitates the model to express the probability within the distribution of the operating environments and its inferences to the dependability of the services. The probabilistic model provides a quantified computation approach to reason about the survivability property and can further support the expression of randomization in terms of efficiency and simplicity [32]. With this model, one can specify the behavior of a networked information system containing alternative services and functions under a various operating environment complies with a probability distribution.
- **Quantified Survivability Specification.** Proposing a survivability specification which specifies the required probability of the availability of each acceptable service. The specification contains a core function set and a distribution of the environment configuration. The required probability indicates the quantified survivability requirement of the system design.
- **Computable Verification using Probabilistic Refinement.** With the probabilistic computation model, the probability of availability of each service in the system can be figured out. The survivability of a system can be verified by checking the probabilistic refinement between the computation model and the survivability specification. Some important inference involving the probabilistic refinement are presented for the reasoning and proof.

The remainder of the paper is organized as follows. Section 2 introduces the fundamental of probabilistic programs and its denotational semantics. Section 3 gives the two-layer model of system and researches the divergence and failure semantics of the functions and services. The general refinement is extended into the probabilistic refinement to reason about the system's survivability. In Sect. 4 the survivability specification is presented to design and analyze the survivable system. The survivability can be verified by checking the probabilistic refinement between the system model and its survivability specification. Section 5 demonstrates the formal framework with two case studies: the Command-and-Control case study given by Knight [16] and a hypothetical Warship Command-and-Control system. At last, Sect. 6 concludes the paper.

2. Fundamental of Probabilistic Programs

In this section, we present our language of the survivability system which is an extension of Dijkstra's language of guarded commands [12], [13] by including the program $WAIT$ and probabilistic choice $P\,{}_r\oplus\, Q$. The syntax of our programming language is given below.

$$P, Q ::= ABORT \mid SKIP \mid x := e \mid P \lhd b \rhd Q \mid P; Q \\ \mid P\,{}_r\oplus\, Q \mid P \sqcap Q \mid X \mid \mu X \cdot P(X) \mid WAIT$$

Where P and Q are programs, b is boolean expression and r is a real number over the interval $[0, 1]$. Assume that x stands for a list of distinct variables, and e a list of expressions.

Program $ABORT$ stands for the worst program and $SKIP$ represents a program which is always idle. The assignment $x := e$ updates variable x with the value of expression e. Conditional choice $P \lhd b \rhd Q$ behaves like P if boolean expression b is satisfied, otherwise like program Q. The sequential $P; Q$ first executes program P and when P is finished, it executes program Q. The program $P \sqcap Q$ executes P or Q nondeterministically. Probability choice $P\,{}_r\oplus Q$ chooses between programs P and Q with probabilities r and $1 - r$ respectively. Program $WAIT$ always blocks the progress of the whole program. Program $\mu X \cdot P(X)$ defines the recursion program and X is recursive identifier.

In this paper, we will propose a probabilistic relational model to specify the denotational semantics of the language. We first turn to notations and definitions which are necessary for the probabilistic model for our language. we employ the state variable s including data variables and a special control variable *wait* to record the observable system state. The set St collects all possible states, which is formally defined as follows.

$$St =_{df} (Var \to Val) \times (\{wait\} \to \{true, false\})$$

Where Var is the set of programs variables and Val is the Integer set (not affect the generality of the model). In our model, the execution of the program may be blocked in some states which we call a failure. The failure state can be observed with the control variable $wait = true$.

Definition 2.1: A probability distribution $prob$ is a function from St to the interval $[0, 1]$, such that $\sum_{s \in St} prob(s) \leq 1$.

We say that a probability distribution refines another if it assigns higher probabilities to all states.

Definition 2.2: For probability distribution $prob_1, prob_2 \in Prob_{St}$,

$$prob_1 \leq prob_2 =_{df} (\forall s \in St \cdot prob_1(s) \leq prob_2(s))$$

where $Prob_{St} =_{df} St \to [0,1]$ contains all the probability distributions $prob$ over St.

In our framework, a program P can be specified by a relation from St to $Prob_{St}$, i.e.,

$$P \subseteq St \leftrightarrow Prob_{St}$$

Note that the execution of a sequential program either terminates with the observable final state or diverges without any meaningful final state. A program may terminate at different states because of nondeterminism [9] and probabilistic choice [31]. Accordingly we utilize a probability distribution to characterize the possible final states of the program execution. In design calculus [4], an additional control variable *ok* is introduced to indicate whether the program terminates or diverges. In our model, we only sum up the probability of the terminated behaviors and leaves the rest probability to the divergent behavior so that the control variable *ok* is not necessary. As a result of this modeling, the overall distribution of the final state may less than 1 for a program which is possible to diverge, i.e., $\sum_{s \in St} prob'(s) < 1$, where the dashed distribution $prob'$ indicates the observation occurs after the program executes. In our paper, we follow the convention that the undashed and dashed variables record the observation before the execution and after the execution respectively.

The refinement relation over the probabilistic program can be defined as follows.

Definition 2.3: Let P and Q be probabilistic programs. We say P is a refinement of Q, denoted as $P \sqsupseteq Q$, if and only if

$$\forall s, prob' \cdot (P(s, prob') \Rightarrow Q(s, prob'))$$

In UTP theory [4], Healthiness conditions regulate the behaviors of programs so that the model can reflect the reasonable properties of the real program. For the probabilistic model, we list the healthiness conditions a program is supposed to satisfy. Healthiness conditions [35] are defined as equations in terms of an idempotent function ϕ on predicates. Every healthy program represented by predicate P must be a fixed point under the healthiness condition of its respective UTP theory, i.e., $P = \phi(P)$.

H1: A nondeterministic choice made on two identical programs is void.

$$P = P \sqcap P$$

H2: If a program results in a probability distribution (say p), then a better distribution $q \geq p$ should be allowed to be observed in the execution of the program. In other terms, the healthiness condition requires that every probabilistic program satisfy *up-closed* property.

$$P = P; (prob \leq prob')$$

H3: A program would act like $WAIT$ if its predecessor is blocked so that it cannot be executed.

$$P = P \lhd wait = false \rhd WAIT$$

We define a healthiness function $\mathcal{H}$ to mapping an ordinary program to a healthy program which satisfies the healthiness condition. And the rest of the paper only refers to healthy programs.

$$\mathcal{H}(P) =_{df} (P; (prob \leq prob')) \lhd wait = false \rhd WAIT$$

The healthiness function $\mathcal{H}$ is idempotent and monotonic [4] and all healthiness programs constitutes a complete lattice.

In summary, the semantics of our language can be defined as follows.

(1) $ABORT =_{df} \mathcal{H}(True)$

The behavior of the chaotic program $ABORT$ is totally unpredictably.

(2) $SKIP =_{df} \mathcal{H}(prob' = \eta_s)$

Where $\eta_s(t) =_{df} \begin{cases} 1, & t = s \\ 0, & t \neq s \end{cases}$

The program $SKIP$ terminates immediately and changes nothing.

(3) $x := e =_{df} \mathcal{H}(prob' = \eta_{s[e/x]})$

The assignment terminates and changes the value of variable x to e.

(4) $P \,{}_r\!\oplus\, Q =_{df} \exists prob_1, prob_2 \in Prob_{St} \cdot P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge prob' = r * prob_1 + (1-r) * prob_2$

The probabilistic choice $P \,{}_r\!\oplus\, Q$ chooses between programs P and Q with probabilities r and $1-r$ respectively.

(5) $P \lhd b \rhd Q =_{df} (b \wedge P) \vee (\neg b \wedge Q)$

The program of the conditional choice acts like P if $b = true$ and like Q otherwise.

(6) $P; Q =_{df} P; (\uparrow Q)$

Where $\uparrow Q =_{df} \exists G \in St \to Prob_{St}, \forall s \in St \cdot Q(s, G(s)) \wedge prob' = \sum_{t \in St} prob(t) * G(t)$

The sequential program passes the final state of program P on to the program Q as its initial state.

(7) $P \sqcap Q =_{df} \exists prob_1, prob_2 \cdot P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge \exists r \cdot prob' = r * prob_1 + (1-r) * prob_2$

The behavior of $P \sqcap Q$ is like P or Q nondeterministic.

(8) $\mu X \cdot P(X) =_{df} \sqcap \{X | X \sqsupseteq P(X)\}$

The recursive program is defined as the least fix point of the recursive function.

(9) $WAIT =_{df} \mathcal{H}(prob'(s[true/wait]) = 1)$

The program $WAIT$ blocks the execution of the program.

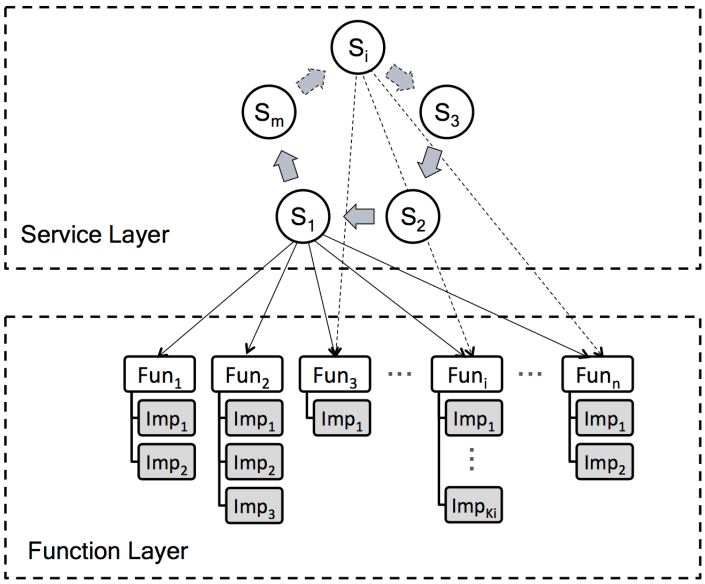


Fig. 1 The two layer model of survivable system.

In the following part of the paper, we will employ a unified syntactic form (α, g, P) to represent a probabilistic program. α is the alphabet of the program (In the rest of the paper, the programs would have the same alphabet if we don't mention it deliberately). $g(s) : St \to \{0, 1\}$ is the guard condition and P is a predicate over the alphabet indicating the semantics of the program execution. We employ the notation $pGCL$ to define all the programs with the syntactic form (α, g, P).

3. Probabilistic Model of Survivable System

In this section, we present a denotational semantical model of the survivable system. The model shown in Fig. 1 is divided into two layers, i.e., function layer and service layer. The system is constructed by the probabilistic choice of services it could provide combined with the operating environment.

3.1 Function and Service

Our model of the system is divided into two layers, function and service. The function is defined by the probabilistic program above while the service consists of functions. Alternative forms (degraded) of services are needed to adapt various environments. The service allows one function to have multiple implementations achieving specific functionalities. The same function of different services can also has different implementations so that the degraded service can afford some (worse) functions to make a tradeoff between the performance and the resources.

Assume that all the functions here completely implement the corresponding functionalities the users expect respectively and the parameters including import and export are concealed for the succinctness of the model. The assumption about functions is not strictly necessary. It can be extended to the form of function with parameters. The analogous extension is in [22], [23].

In our setting, the *signature* of the function is determined by its name (identifier) and the set FN collecting the names of functions. The name merely identifies the function from others but offers no implementation of the function specification. The function $Spec$ maps each function in FN to its specifications, i.e., $Spec : FN \to \mathcal{P}(pGCL)$. In the function layer, the specifications $Spec(f)$ of function f describes all the potential implementations of the function. Obviously the set $Spec(f)$ should be *up-closed* and *convex-closed* due to the semantics of the probabilistic programs.

In the service layer, the service is identified by a triple $(FDec, Imp, Prot)$, where $FDec \subseteq FN$ lists all the functions which the service could provide to the users. The specification $Imp : FDec \to pGCL$ depicts the corresponding implementations of the functions in $FDec$. The protocol $Prot$, the set of sequences of function names, specifies the patterns the service engages in the functions. To describe the property in the process of calling the functions, the set should be prefix closed.

The dynamic behavior of a service could be described by a sequence of a particular triple (tr, div, ref). The trace tr is included in $Prot$. The probability div is the possibility the service may be chaotic after engaging in tr. For any set X of functions, $ref(X)$ gives the probability with which the system may refuse X after engaging in tr.

Not all the sequences lead to a satisfying state and the execution of service could lead to divergent or deadlock/wait state. The probabilistic model itself has the expressive power to depict the divergence [3], [6], thus we needn't employ the additional control variable to denote the divergent state. The control variable *wait* and the guarded conditions indicate which service functions could engage and which could not. Moreover, the model has the ability to suggest how much probability the system (service) could lead to a

divergent state or refuse to perform in some sets of functions after engaging a trace.

Definition 3.1 (Probabilistic Divergent Trace): For the service $(FDec, Imp, Prot)$, a trace $\langle f_1, f_2, \ldots, f_n\rangle \in Prot$ is called probabilistic divergent trace with probability α if the service can lead to a divergent state with probability α after engaging in the trace, i.e.,

$$\exists prob_1 \cdot (Imp(f_1); Imp(f_2); \ldots; Imp(f_n))[prob_1/prob'] \wedge \sum_{s \in St} prob_1(s) \leq 1 - \alpha$$

Definition 3.2 (Probabilistic Failure): Given that the service $(FDec, Imp, Prot)$, it can refuse to perform the set X of functions with probability α after engaging in trace $\langle f_1, f_2, \ldots, f_n\rangle \in Prot$ if

$$(\exists\, prob_1, s_1 \cdot (Imp(f_1); Imp(f_2); \ldots; Imp(f_n))[prob_1/prob'] \wedge (prob_1(s_1) \geq \alpha \wedge \forall f \in X \cdot \neg g(s_1) \vee s_1(wait) = true))$$

or $\langle f_1, f_2, \ldots, f_n\rangle$ is a divergent trace with probability α.

Definition 3.3 (Semantic Definition): The formal semantics $Traces(S)$ of a service $S = (FDec, Imp, Prot)$ can be specified as a set $\{(tr, div, ref) \mid tr \in Prot\}$, where the trace tr is the probabilistic divergent trace with probability div and could refuse to perform the functions in set X with probability $ref(X)$.

After achieving the precise understanding of service, we utilize the definition to introduce the degraded services. Degraded services is tradeoff between performance and resources. The tradeoff is exploited by constructing service with less provision of functionality and worse function implementation for coping with faults and attacks in some environment.

Definition 3.4: Let $S_1 = (FDec_1, Imp_1, Prot_1)$ and $S_2 = (FDec_2, Imp_2, Prot_2)$ be services, S_2 is called a degraded service of S_1 if

(1) $FDec_1 \supseteq FDec_2$.

(2) $\forall f \in FDec_2, Imp_1(f) \sqsupseteq Imp_2(f)$.

In this subsection, our two-layer model consists of the features which the survival system performs. The system can be expressed as a collection of different services. Every service have its own functionals. And for the same functionality, different services would have a different implementation. In this sense, the degraded service can be obtained by using a worse implementation for tradeoff. In summary, the model accurately describes the survival system component, as well as the relationship between function and service.

3.2 Probabilistic Refinement

In modern software engineering, software developers apply software refinement [17], [25] in order to proceed from a high-level abstract model to a final executable software system by adding more details stepwisely. The software system would becomes larger and more complicated during this development process [5]. A service contains a set of functions. A refinement relation is introduced to reason about the correctness of the services.

Definition 3.5: The service $S_1 = (FDec_1, Imp_1, Prot_1)$ is a refinement of the service $S_2 = (FDec_2, Imp_2, Prot_2)$ denoted by $S_1 \sqsupseteq_S S_2$ if

(1) $Prot_1 \supseteq Prot_2$.

(2) $Traces(S_1 \downarrow Prot_2) \subseteq Traces(S_2 \downarrow Prot_1)$.

Where, given that a service $S = (FDec, Imp, Prot)$, the notation $S \downarrow Prot1$ defines a degraded service $S' = (FDec, Imp, Prot')$ whose protocol $Prot'$ is the subset of protocol $Prot$, i.e., $Prot' = Prot \cap Prot1$. We generalize the probabilistic choice between probabilistic programs in terms of the operator $S[r]T$. It chooses between S and T with probability r and $1-r$ respectively. It offers probabilistic choice between services. In our model, a system would be regarded as a probability choice of all the services that it could provide later. The semantics of $S[r]T$ is given below:

$$\begin{aligned} Traces(S[r]T) =_{df} \{\, (tr, div, ref) \mid \exists d_1, d_2, f_1, f_2 \cdot \\ (tr, d_1, f_1) \in Traces(S) \wedge (tr, d_2, f_2) \in Traces(T) \wedge \\ div \leq (r \times d_1 + (1-r) \times d_2) \wedge \\ ref \leq (r \times f_1 + (1-r) \times f_2) \wedge tr \notin dom(Traces(T)) \wedge \\ (tr, div, ref) \in Traces(S) \wedge tr \notin dom(Traces(S)) \wedge \\ (tr, div, ref) \in Traces(T)\, \} \end{aligned}$$

We generalize the probabilistic refinement relation from the service refinement relation. Survivability analysis is based on probabilistic refinement.

$$P \sqsupseteq_r Q =_{df} P \sqsupseteq_S (Q[r]\perp_S)$$

where $\perp_S$ indicates a service which only has a function which is defined by the unpredictable program $\perp$. The relation $P \sqsupseteq_r Q$ reads as 'P refines Q with probability r'. In particular, the probabilistic refinement degenerates into the general refinement if $r = 1$ and the relation suffices trivially if $r = 0$.

The probability refinement relation provides a precise interpretation about the meaning of one program refines another program with certain probability. The concept of probabilistic refinement could be used in other model.

We list some important properties about probability refinement in Table 1.

The probability refinement is based on the probabilistic programs, with which we can describe the actual situation accurately. In real world, the system will behave as a certain service at a time. As time goes by, with the environment changes, the system will evolve into another service. So the actual system can be a series of combinations of different services. In order to compare the behavior of the system, we give the definition of probabilistic refinement.

3.3 System with Operating Environment

The operating environments could affect the performance of

Table 1 The laws for probabilistic refinement.

$P \sqsupseteq_r P$	**reflexivity**
$\frac{P \sqsupseteq_r R \quad Q \sqsupseteq_r S}{P \lhd b \rhd Q \sqsupseteq_r R \lhd b \rhd S}$	**$\lhd\rhd$-monotonicity**
$\frac{P \sqsupseteq_r Q \quad Q \sqsupseteq_s T}{P \sqsupseteq_{r*s} T}$	**quasi-transitivity**
$\frac{P \sqsupseteq_r Q \quad r \geq s}{P \sqsupseteq_s Q}$	**probability-monotonicity**
$\frac{P \sqsupseteq_r Q}{P; R \sqsupseteq_r Q; R}$	**;-left-monotonicity**
$\frac{P \sqsupseteq_r Q}{R; P \sqsupseteq_r R; Q}$	**;-right-monotonicity**
$\frac{P \sqsupseteq_r Q \quad R \sqsupseteq_r T}{P[s]R \sqsupseteq_r Q[s]T}$	**[r]-monotonicity**

the service and the survivable system must have the power to withstand certain types of faults and security attacks. Thus we should describe the environment factors and how it affects the behavior of a survivable system. We employ the notion Env to define the set of all operating environment conditions and

$$E_p : Env \to [0, 1]$$

where E_p satisfies $\sum_{\theta \in Env} E_p(\theta) = 1$
to define the probability distribution over E_p respectively. Each of the environment conditions is stable and predictable, which is obtained by a long time observations. The probability distribution E_p precisely captures the meaning of the environment variety and stability from a quantitative point of view [8], [14].

To model the affection to the system caused by the environment, we add a guarded condition merely involving the environment condition for the succinctness ahead of each service the system could afford. Thus the system could be defined as a guarded choice between services:

$$Sys =_{df} if\ (b_1(\theta) \to S_1, b_2(\theta) \to S_2, \cdots\cdots, b_n(\theta) \to S_n)$$

where $S_1, S_2, \ldots, S_n$ are services of the system Sys. $\theta \in Env$ is the environment condition variable. b_i is the guard condition satisfying $\bigvee_i b_i = True$ and $b_i \wedge b_j = false$.

Definition 3.6: A system $Sys =_{df} if(b_1(\theta) \to S_1, b_2(\theta) \to S_2, \cdots\cdots, b_n(\theta) \to S_n)$ with the probability distribution E_p could be regarded as a probability choice of services, i.e.,

$$Sys = \sum\{p_i \& S_i | p_i = \Sigma_{\theta \in Env} E_p(\theta) * b_i(\theta), 1 \leq i \leq n\}$$

where the notation $\Sigma\{p_i \& S_i | 1 \leq i \leq n\}$ is defined recursively:

$$\textstyle\sum\{1 \& S\} =_{df} S \text{ and}$$
$$\textstyle\sum\{r \& S\} \cup \{p_i \& S_i | i \in I\} =_{df} S[r]\{(p_i/1 - r) \& S_i | i \in I\}.$$

The system is defined as a probabilistic choice of the services with respect to the operating environment. With the change of the environment, there is an acceptable service of the system adapting the environment. However the situation is not so optimistic. When the best choice of service is unavailable, an alternative (degraded) service would take over.

4. Survivability Specification

Now, we will devote ourselves to defining the survivability property. The precise and adequately comprehensive definition of the survivability must have three essential characteristics, i.e., various operating environments, alternate acceptable forms of service and the probability of the availability of each service [16]. John C. Knight presents a six-tuple survivability specification and he indicates that a system is survivable if it complies with its survivability specification. However he does not provide a computation model for the system. A verification approach is also missing to determine whether a system satisfies the specification.

We present a more elegant and abstract survivability specification in our framework and reveal how our model of system satisfies the new survivability specification. First, the survivability specification should list all the services that it can provide. In different environment, the system can perform different forms of service to ensure that the functionalities expected by the users can be met. The specification also introduces the concept of core functions to define primary service in a survivability system with respect to the operating environment. The relative service values offer an ordering on the user's perceived service value from their requirement under reachable environmental state. The value is represented by a natural number and the ordering is total. It indicates the evaluation of each service in the view of the user. The probability distribution over operating environment describes the condition to which the system is subject in a period and the environment also includes certain specific failure modes or threats. But for each function in the system, its functionality is certain. At last, a service probability is specified to define the probability that service must meet the corresponding dependability requirement, i.e., the

probability gives an assurance that the various different acceptable services will be provided at some adequate level.

Definition 4.1 (Survivability Specification): A survivability specification is a five-tuple, $(\mathbb{S}, F, E_p, V, P)$ where

(1) $\mathbb{S}$ is a set of acceptable forms of services in the system. Each service consists of a set of the functions it can provide.
(2) F is the core function set which defines the core functions of the survival system. A service provides all the functions in the set F is called essential service.
(3) $E_p : Env \rightarrow [0, 1]$ is the environment distribution. The notion indicates the probability distribution of environment states. The set E contains all the possible environment states and satisfies $\sum_{\theta \in E} E_p(\theta) = 1$
(4) $V : \mathbb{S} \times Env \rightarrow N^+$ is the relative values. The value given by the user indicates the order of the services which the user would prefer in different environment conditions.
(5) $P : \mathbb{S} \rightarrow [0, 1]$ is the probabilistic requirement on the operation of the acceptable forms of service.

The services in the specification should at least include an essential service to meet the expectation of the user with respect to the normal environment. The relative service values indicate which service satisfying the user has a more evaluation. But the ordering may violate the implementation decided by the engineering. For the trusted requirement, we insist that the order be consistent with the refinement relation between services if the latter exists. Thus the consistent specification is outlined below:

Definition 4.2 (Consistent Specification): A survivability specification $(\mathbb{S}, F, E_p, V, P)$ is consistent if

(1) $\exists S \in \mathbb{S}, S.FDec \supseteq F$. We can at least find a service is an essential service in our survivable system.
(2) $\forall S_1, S_2 \in \mathbb{S} \wedge e \in Env, S_1 \sqsupseteq_S S_2 \Rightarrow V(S_1, e) \geq V(S_2, e)$. The service which refines another service should receive a higher relative service value given by the user.

The system we constructed is survivable if complies with its consistent specification defined above. The viewpoint is coincident with Knight's but we give a formal method to determine how a system complies with the specification. The system should ensure the probability requirement is satisfied for each of the service of the specification.

Definition 4.3 (Survivability System):
A system $Sys = \sum\{p_i \& S_i | p_i = \Sigma_{\theta \in Env} E_p(\theta) * b_i(\theta), 1 \leq i \leq n\}$ is survivable if it complies with its consistent specification $(\mathbb{S}, F, E_p, V, P)$, i.e., $\forall S \in \mathbb{S}, Sys \sqsupseteq_{P(S)} S$.

The probability refinement in the definition of survivability system not only indicates the system could provide the service expected by the user but also means the system ensures that the system meet the dependability requirement of the service in the specification with the corresponding probability. Moreover, we can define the weakest system which satisfies a given specification. Thus the probability refinement could be used to develop the survivable system and all systems satisfying the specification must be the probability refinement of the weakest system. We could also define another refinement relation over specifications with respect to the given environment conditions, the relative service values and core functions in the future work.

Now, we will compare our specification with Knight's survivable specification. In our setting, we simplify the definition of the environment to make it more abstract. We concentrate on the probability distribution of the environment other than the discussion of various concrete environment factors. The service based on the probabilistic model could be reasoned about and therefore our model is computable. To avoid the gap between the system designers and the users on perceived service value, we emphasize the consistency between the V table and the service refinement. The valid transitions T could be inducted from our specification since the services given in the specification are constructed from the probabilistic programs. The core function is introduced to define the primary service satisfying the primary requirement. A survivable system should at least provide the primary service to the user despite of the environment changes.

5. Case Study of Survivable System

In this section three examples of survivable systems, i.e., the RES system, the LWC system and the C2 system, will be given to illustrate our approach described above. We formalize these three systems in our framework and give the corresponding survivable specifications. Thus the survivable system could be developed by refining the survivable specifications step by step. But we cannot list the specified and exhaustive specifications of all the functionalities for a huge system, so we assume that all the functions are regarded as the basic and appropriate modules in our examples.

5.1 The RES System

The first example introduces the restaurant system which provides two kinds of functionalities: *hall food* and *delivery*. For hall food, it provides three different forms of services, i.e., NSA, SA, $EatOutside$. The first two are both settled in the inside of the restaurant; NSA indicates that the customers cannot smoke in the non-smoking while SA permits the customers smoke in the smoking area. The two areas are separated by a solid door and the layout will not make secondhand smoke from the smoking area to impact the non-smoking area. If the weather is awesome, $EatOutside$ is a good choice and the customers are arranged at the outside of the restaurant. Besides, the restaurant may provide the delivery when the weather is not so heavy. The weather conditions have a strong influence on the functionalities provided by the restaurant. When the weather is good, the restaurant may provide the functionalities hall food and delivery under most conditions. But when the weather is not so good, the

Table 2 The satisfied V table of the RES system.

V	S_1	S_2	S_3
e_1	3	1	2
e_2	3	2	1

restaurant may only provide NSA and SA with a big possibility.

For the RES system described above, we first give a possible survivable specification $(\mathbb{S}, F, E_p, V, P)$ and then design a possible survivable system Sys in our framework. At last, we check the satisfiability of the system and the specification based on the probabilistic refinement. If the RES system is a survival system, it should provide one of the following services in different environments. Here, we regard NSA, $NSA\&SA$, and $NSA\&EOut$ as different implementations of functionality hall food.

- Service S_1: the customers are served inside and not allowed to smoke in the restaurant. It also provides the functionality delivery, i.e., $S_1 = \{NSA,\ delivery\}$.
- Service S_2: the customers are served inside, but which of NSA and SA is not determined. Besides the delivery is provided, i.e., $S_2 = \{NSA\&SA,\ delivery\}$.
- Service S_3: the customers are served in non-smoking area or in outside area and the delivery is provided, i.e., $S_3 = \{NSA\&EOut,\ delivery\}$.

Here, we regard a variant of $FDec$ as a service declaration, where the function name are substituted with its some implementation. Thus the set $\mathbb{S}$ is defined as $\{S_1, S_2, S_3\}$ and the core function F is $\{hall\ food\}$. In this example, we employ two kinds of environment states e_1 and e_2 to represent different weather conditions. The state e_1 indicates the weather is good while e_2 indicates the weather is not good. The notation E_p indicates the probability distribution of environment states. Here, we assign $E_p(e_1) = 0.8$ and $E_p(e_2) = 0.2$. The relative service values are listed in Table 2, the ordering of that indicates which service the user would prefer in the environment conditions. The probabilistic requirement P for the accepted forms of services is defined as $P = \{(S_1, 0.59), (S_2, 0.78), (S_3, 0.75)\}$.

In order to explain how the formal language defined in Sect. 2 facilitates the check of the system survivability, we just give a sketch of the definition of functions and services.

Let α the alpha of the program. Define variables $hfood$ and $deli$ to denote the implementation form of functionalities hall food and delivery. In detail, $hfood = 1$, $hfood = 2$ and $hfood = 3$ indicate the customers are served in non-smoking area, smoking area and outside area respectively. $deli = 1$ and $deli = 0$ state whether the delivery is available or not. Thus some function definitions are described as below.

$$NSA : (\alpha, true, hfood := 1)$$
$$NSA\&SA : (\alpha, true, hfood := 1 \sqcap hfood := 2)$$
$$NSA\&EOut : (\alpha, true, hfood := 1 \sqcap hfood := 3)$$
$$delivery : (\alpha, true, deli := 1)$$

In summary, a survivable specification $(\mathbb{S}, F, E_p, V, P)$ is given. Obviously, for each service S in $\mathbb{S}$, F is the subset of $S.FDec$ since all the service provide the core functionality hall food. Note that S_1 is the service refinement of both S_2 and S_3 according to the definitions involved and refinement calculus. In consideration of the Table 2, the relative service value of S_1 is bigger than one of S_2 and S_3 in whichever environment, which indicates the given V table is coincident with the service ordering. That is, the specification $(\mathbb{S}, F, E_p, V, P)$ is a consistent specification.

Now, we will construct a system in our framework.

$$Sys = \{(e = e_1) \rightarrow S_1', (e = e_2) \rightarrow S_2'\}.$$

Where $S_1' = \{f_1, g_1\}$ and $S_2' = \{f_2, g_2\}$, the details of f_1, g_1, f_2, g_2 are interpreted as below.

$$f_1 = (\alpha, true, (((hfood := 1_{0.75} \oplus hfood := 2)$$
$$_{0.8} \oplus hfood := 3)_{0.95} \oplus ABORT))$$
$$g_1 = (\alpha, true, deli := 1_{0.9} \oplus ABORT)$$
$$f_2 = (\alpha, true, (((hfood := 1_{0.8} \oplus hfood := 2)$$
$$_{0.99} \oplus hfood := 3)_{0.9} \oplus ABORT))$$
$$g_2 = (\alpha, true, deli := 1_{0.4} \oplus ABORT)$$

The function f_1 indicates that the customers are served in the non-smoking area with probability 0.57, the smoking area with probability 0.19, the outside area with probability 0.19 and the customers can not eat in the restaurant due to its capacity with 0.05.

Considering the given E_p, the system can be described as $Sys = 0.8\&S_1' + 0.2\&S_2'$. Now, we compare the system Sys and the specification in terms of the probabilistic refinement. For service S_1, in the environment e_1, the system Sys may provide NSA with probability 0.57 and $delivery$ with probability 0.9; in the environment e_2, the system Sys may provide NSA with probability 0.7128 and $delivery$ with probability 0.4. Thus the system Sys may provide NSA with probability 0.59856 and $delivery$ with probability 0.8. In other words, $Sys \sqsupseteq_{0.59856} S_1$, thus $Sys \sqsupseteq_{0.59} S_1$.

Similarly, we can conclude that $Sys \sqsupseteq_{0.7862} S_1 \sqsupseteq_{0.78} S_1$ and $Sys \sqsupseteq_{0.75236} S_1 \sqsupseteq_{0.75} S_1$. Therefore, the system is a survivable system with respect to the given specification.

5.2 The LWC System

The second example introduced below is one of the most important services in a Warship Command-and-Control system. The system which is called LWC system can provide three kinds of functionalities: *Location*, *Weather query* and *Communication*. Two kinds of data transmission channels are proposed to protect the safety and privacy of the communication. One is called the private channel, which is used to transfer the data from the system only, while the other is called the public channel, which is used to transfer the encrypted data from both the system and the internet. The functionality *Location* can get the location information of the warship from the local device without using any channel. But the other two functionalities *Weather query* and

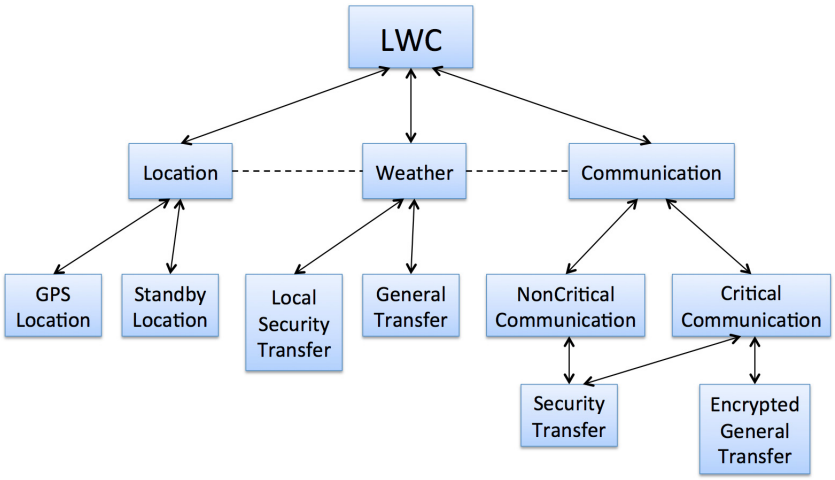


Fig. 2 The structure of warship command-and-control system (LWC).

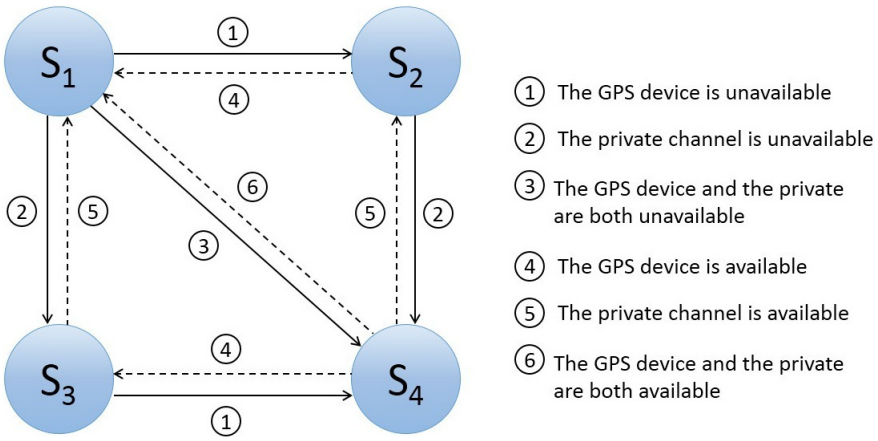


Fig. 3 The switch of four services in the LWC system.

Communication need to get their information from other services respectively. So these two functionality will use the two kinds of channels to communicate with other services.

We give the details of the three functionalities of the LWC system as follows. Figure 2 shows the hierarchy of the LWC system.

- **Location.** LWC provide two kinds of implementations to get the longitude and latitude of the warship. The information can be acquired from a local GPS device first, denoted by GPSLocation. When GPSLocation is unavailable, the information can be acquired from the local backup device, which is called StandbyLocation. Once the GPSLocation becomes available, the LWC system will get the location information via it again.
- **Weather Query.** Weather information is quite necessary for the LWC system, which is got from channels. There are two kinds of implementations: LocalSecurityTransfer and General Transfer. In the first implementation, the weather information is acquired from a Navy Meteorological device and transferred via the private channel. When the private channel is unavailable, the weather information is acquired from the internet via the public channel, which is the second implementation.
- **Communication.** Actually it can be divided into two sub functionalities since the data can be critical or noncritical. For the first functionality CriticalCommunication, the data is usually transferred via the private channel. The implementation of this functionality is called SecurityTransfer. Once the private channel becomes unavailable, the data will be encrypted first and then transferred via the public channel until the private channel becomes available again. This kind of implementation is called EncryptedGeneralTransfer. For the second functionality NonCriticalCommunication, the noncritical data is transferred via the private channel. Once the private channel becomes unavailable, the data will be dropped. The two kinds of functionalities share one implementation: SecurityTransfer.

If LWC system is a survivable system, it should always provide one of the following services in different environ-

ments:

- Service S_1: It uses function GPSLocation to implement its location functionality, LocalSecurityTransfer to implement its weather query functionality, and SecurityTransfer to implement its CriticalCommunication and NonCriticalCommunication functionalities.
- Service S_2: It has the same implementations as service S_1 except using StandbyLocation to implement its location functionality instead of GPSLocation.
- Service S_3: It uses GPSLocation to implement its location functionality as service S_1. It adopts GeneralTransfer to implement the weather query functionality, and EncryptedGeneralTransfer to implement the CriticalCommunication functionality. Both of them only use the public channel.
- Service S_4: It uses StandbyLocation to implement its location functionality, GeneralTransfer to implement its weather query functionality, and EncryptedGeneralTransfer to implement its CriticalCommunication functionality.

The environment makes the system employ different services, which is decided by the status of the GPS device and the private channel. When the system starts, we assume all the devices and channels are available and the system provides service S_1. If the GPS device is unavailable, the system turns to service S_2. Or if the private channel is unavailable, the system turns to service S_3. When both the GPS device and private cannot be available, the system can still provide service S_4. The switch of the four services is shown in Fig. 3.

Let f_1 be the location functionality, f_2 be the weather query functionality, f_3 be the CriticalCommunication functionality and f_4 be the NonCriticalCommunication functionality. The LWC system can be formalized by our model as follows:

The system $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$, which means that the system LWC can provide four kinds of services in certain environments. The four kinds of services are defined as:

$$S_1 = \{f_1, f_2, f_3, f_4\}$$
$$S_2 = \{f_1', f_2, f_3, f_4\}$$
$$S_3 = \{f_1, f_2', f_3'\}$$
$$S_4 = \{f_1', f_2', f_3'\}$$

Here f_i and f_i' $(1 \le i \le 4)$ represent the different implementations of the same functionality which implies $f_i \sqsupseteq f_i'$. For example, f_1 and f_1' represent the two kinds of implementations of the functionality location and f_1 represents GPSLocation, f_1' represents StandbyLocation, and it implies that $f_1 \sqsupseteq f_1'$. Obviously the essential functionalities are location, weather query and critical communication, so the set $F = \{f_1, f_2, f_3\}$. As LWC does not concern the environment element outside the system, we merely give two kinds of environment states and the set E_p can be set as $E_p = \{(e_1, 0.7), (e_2, 0.3)\}$, where e_1, e_2 represent the possible environment states respectively. The V table lists the total order under different environment states in Table 3. It is no doubt that the value of S_1 is the highest and the value of S_4 is the lowest, but the ordering of the values of S_2 and S_3 cannot be determined. Since our specification cannot indicate which of them are higher, both of them shown in Table 3. can comply with the specification according to the different environment states. Obviously, the V table is consistent. At last, we give a possible service probabilities in the example:

$$P = \{(S_1, 0.95), (S_2, 0.04), (S_3, 0.005), (S_4, 0.005)\}$$

Thus the five-tuple $(\mathbb{S}, F, E_p, V, P)$ defines the survivable specification of the Warship Command-and-Control system in our framework.

Table 3 The satisfied V tables.

V	S_1	S_2	S_3	S_4
e_1	4	2	3	1
e_2	4	3	2	1

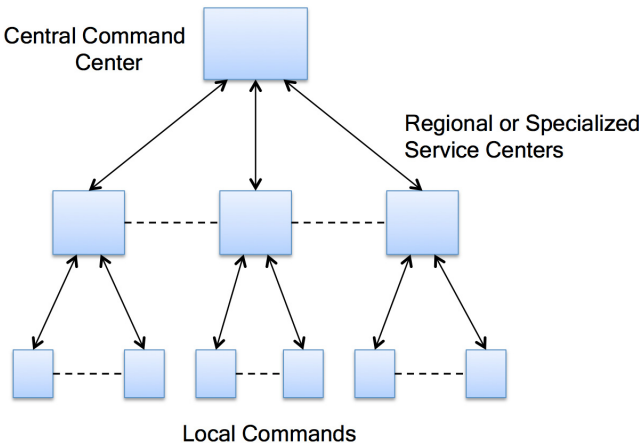


Fig. 4 The structure of hypothetical command-and-control system (C2).

5.3 Command-and-Control System

The last example is Command-and-Control (C2) system in [2]. The C2 system as a hypothetical military system has a central command center and some intermediate nodes which provide regional or specialized services, as well as a large number of leaf network nodes used by local commanders. The general network topology is shown in Fig. 4.

We will propose the consistent specification of C2 system and construct the survivable system satisfying the specification $(\mathbb{S}, F, E_p, V, P)$ in our framework. The details about implementations and specifications of functions are ignored and the services are denoted as the set of the functions since we merely want to display the satisfaction of the system and the specification here. According to the requirement documents, the C2 system provides five different basic functions, including both central information servers f_1, regional information servers f_2, transmission of command information f_3, transmission of security information f_4 and transmission of normal information f_5. Thus the set FN of function identifiers is defined as $\{f_i \mid 1 \le i \le 5\}$. Here we don't distinguish

the name and the specification of the function and the function f_i is the refinement of f_i' with respect to i. The transmission of command information f_3 is regarded as the core function in C2 system, i.e., $F = \{f_3\}$.

In the Command-and-Control system, there are five acceptable forms of services. *Full Command, Control, and Analysis* S_1 provides complete and normal functionality and *Low Performance* S_2 also has full functionality, but with higher latencies. *Regional* S_3 is limited to regional information servers and transmission of security information is unavailable. *Maximum Alert* S_4 requires the system operate with no network traffic, but the local regional information processing is available. *Command only* S_5 limits transmission operate to basic command only. The above service would be formalized as the set of functions in the following. Thus we conclude $S_1 \sqsupseteq_S S_i$ for $2 \leq i \leq 5$ and no more services meet service refinement.

$$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$$
$$S_2 = \{f_1', f_2', f_3', f_4', f_5'\}$$
$$S_3 = \{f_2, f_3, f_5\}$$
$$S_4 = \{f_2, f_3, f_4\}$$
$$S_5 = \{f_1, f_2, f_3\}$$

Thus the set $\mathbb{S} = \{S_i \mid 1 \leq i \leq 5\}$ defines all the services which the C2 system could provide to the user. Next we consider the operating environment *Env* which can affect the acceptable forms of service. In Knight's example we can get four different environment states: *No Threat, Regional Conflict, Dispersed Conflict, Distributed Security Threat*, denoted as e_1, e_2, e_3, e_4 respectively. Correspondingly, a reasonable probability distribution over *Env* is given, i.e., $E_p = \{(e_1, 0.9), (e_2, 0.05), (e_3, 0.03), (e_4, 0.02)\}$.

The relative service values, which are displayed in Table 5, are determined by the user subjectively and the ordering indicates which service the user would prefer in the environment conditions. V should meet the consistent requirement defined above, i.e., finer service has higher relative service value. The service S_1 provide complete functions with higher performance and it has the highest relative service value. The completely reasonable V is listed in Table 5. Moreover we give a possible service probabilities in Table 4 according to the dependability requirement, which is given by the user subjectively. Thus we finish the definition of the survivability specification $(\mathbb{S}, F, E_p, V, P)$ of the C2 system and obviously the specification is consistent.

Table 4 Services probabilities.

Service Probability	The Value
$P(S_1)$	0.9975
$P(S_2)$	$1 - 10^{-4}$
$P(S_3)$	$1 - 10^{-4}$
$P(S_4)$	$1 - 10^{-6}$
$P(S_5)$	$1 - 10^{-6}$

A survivable system would be described as $Sys = \sum\{p_i \& S_i \mid 1 \leq i \leq 5\}$. The value of p_i $(1 \leq i \leq 5)$ can not be fixed since the concrete specification of function is known. A system is survivable if it complies with its consistent specification. Thus we conclude $Sys \sqsupseteq_{P(S_i)} S_i$ for $1 \leq i \leq 5$. Therefore we could utilize the probabilistic refinement to determine whether the system is deemed to satisfy the survivability specification.

6. Related Work

In recent years, many researchers have paid more attentions on the precise definition of survivability. Thus various comprehension and interpretations [16], [29] from different perspectives are given and the corresponding survivability evaluation models and methods are also proposed [33]. The notion of survivability initially is a common concept in weapons systems engineering [2], [24] and the definition indicates the degraded or different services should be included implicitly. The concepts of damage and probability are also included. Furthermore, a networked survivable system [30] was specified which was used widely in real world. Vickie investigated some previous definitions [33] about survivability and then proposed his own definition, in which some features should be involved in the survivable system, such as threat, adaptability and continuity of service. Deutsch presented a general and intuitive notion of the concept of survivability in [11] and Ellison *et al*. presented that the survivability is the ability of a network computing system to provide essential services in the presence of attacks and failures, and recovers full services in a timely manner in [29]. Knight believed all the above definitions are not adequately precise to support an engineering approach to the specification. And all of them lack of decidable criteria to determine whether a given system can be deemed survivable. Based on the previous work and research results, Knight made a comprehensive discussion about system survivability in [16] and analyzed four critical infrastructure applications involving financial payment system, electric power system, rail transportation system and air traffic control system [15]. He presented a survivable specification and gave a rigorous definition of what a survivable system is. Besides he also expounded the differences between survivability and other related concepts, e.g., reliability and availability.

Koziolek et al. propose an analytical model and metrics for survivability assessment and uses this modeling to

Table 5 Relative service values.

	S_1	S_2	S_3	S_4	S_5
e_1	5	4	3	1	2
e_2	5	2	4	1	3
e_3	5	3	2	1	4
e_4	5	1	3	4	2

design power distribution system in smart grid [36]–[38]. Three design principles are presented to reduce the complexity of computing the metrics of interest. In their model, the system survivability can be reflected by the efficiency of the failure recovery process. In our model, the assessment of system survivability is considered as the satisfactory between the system behavior under hostile environment and the promised core function requirement. Our model has inherent compositionality which can significant reduce the computation complexity.

Heegaard et al. demonstrate modeling approaches including stochastic reward nets and continuous time Markov chain to quantify network survivability and clarify the tradeoffs regarding the cost of changing, extending and solving models [39], [40]. Our modeling quantifies the system behavior with the probability distribution on post states. We uses uniformed formal semantics to specify the system behavior and the requirement. The survivability can be assessed by checking a refinement relation between system and requirement with probability.

7. Conclusion

This paper presents a denotational semantics in terms of a probabilistic model for survivable systems. It also proposes a new survivability specification based on the probability. The model is designed into two layers, i.e., function and service. A system is regarded as a probabilistic choice of guarded services with respect to the operating environment. The system is called survivable if it complies to its corresponding consistent survivability specification. The meanings of essential service and degraded service have been formalized. Furthermore, the probabilistic refinement is introduced for designing, analyzing and reasoning about the survivable system. In the future, we will continue to explore further related theories for the survivability.

Acknowledgement

This work was partly supported by the Danish National Research Foundation and the National Natural Science Foundation of China (Grant No. 61361136002) for the Danish-Chinese Center for Cyber Physical Systems. And, it is also supported by National Basic Research Program of China (Grant No. 2011CB302904), National High Technology Research and Development Program of China (Grant Nos. 2011AA010101 and 2012AA011205), National Natural Science Foundation of China (Grant Nos. 61321064 and 91118008), Shanghai STCSM Project (No. 12511504205), and Shanghai Knowledge Service Platform Project (No. ZF1213).

References

[1] A. Avizienis, J. Laprie, and B. Randell, Fundamental Concepts of Computer System Dependability, IARP/IEEE-RAS Workshop on Robot Dependability, Technological Challenge of Dependable Robots in Human Environment, Seoul, Korea, May 2001.

[2] R.E. Ball, The Fundamentals of Aircraft Combat Survivability Analysis and Design, American Insitute of Aeronautics and Astronautics (AIAA), 1985.

[3] C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall International Series in Computer Science, 1985.

[4] C.A.R. Hoare and J. He, Unifying Theories of Programming, Prentice Hall International Series in Computer Science, 1998.

[5] C.C. Morgan, Programming from Specifications, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1994.

[6] C.C. Morgan, A. Mclver, K. Seidel, and J.W. Sanders, Refinement oriented probability for CSP, Technical Report PRG-TR-12-94, Programming Research Group, Aug. 1994.

[7] C.C. Morgan, A. Mclver, K. Seidel, and J.W. Sanders, "Probabilistic predicate transformers," Technical Report PRG-TR-5-95, Programming Research Group, Jan. 1995.

[8] C. Jones and G. Plotkin, "A probabilistic power domain of evaluations," Proc. Fourth IEEE Symposium on Logic in Computer Science, Cambridge, MA, 186-195, 1989.

[9] C. Jones, Probabilistic Non-determinism, Doctoral Thesis, Edinburgh University, also available as Technical Report ECS-LFCS-90-105, 1990.

[10] C. Wang, J. Davidson, J. Hill, and J. Knight, "Protection of software-based survivability mechanisms," 2001 International Conference on Dependable Systems and Networks (DSN 2001) (formerly: FTCS), July 2001.

[11] M.S. Deutsch and R. Willis, Software Quality Engineering: A Total Technical and Management Approach, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[12] E.W. Dijkstra, "Guarded commands, non-determinancy, and formal derivation of programs," Commun. ACM, vol.18, pp.453–457, 1975.

[13] E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, Eenglewood Cliffs, NJ, 1976.

[14] F.W. Lawvere, "The category of probabilistic mappings," preprint, 1962.

[15] J.C. Knight, M. Elder, J. Flinn, and P. Marx, "Summaries of four critical infrastructure systems," Technical Report CMU /SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, Nov. 1997.

[16] J.C. Knight, E.A. Struck, and K.J. Sullivan, "Towards a rigorous definition of information system survivability," DARPA Information Survivability Conference and Exposition, pp.78–89, 2003.

[17] J.C. Knight and E.A. Strunk, Achieving Critical System Survivability through Software Architectures, Architecting Dependable Systems II, LNCS 3069, pp.51–78, Springer-Verlag, 2003.

[18] J. He, "Simulation and process refinement," Form Aspect Comp, vol.229-241, 1989.

[19] J. He, K. Seidel, and A. Mclver, "Probabilistic models for the guarded command language," Science of Computer Programming, vol.28, no.2-3, pp.171–192, 1997.

[20] J. He and C.A.R. Hoare, "Linking theories in probabilistic programming," Inf. Sci., vol.119, no.3-4, Oct. 205-218, 1999.

[21] J. He and J.W. Sanders, "Unifying probability. Unifying theories of programming," First International Symposium, UTP 2006, Walworth Castle, County Durham, UK, Feb. 2006.

[22] J. He, "Refinement and test case generation in Unifying Theory of Programming," 24th IEEE International Conference on Software Maintenance (ICSM 2008), 2008.

[23] J. He, "Service refinement," Science in China Series F: Information Sciences, vol.51, no.6, pp.661–682, 2008.

[24] J. He, "Probabilistic programming with coordination and compensation," 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement, p.1, 2009.

[25] J.M. Morris, "A theoretical basis for stepwise refinement and the programming calculus," Science of Computer Programming, vol.9, no.3, pp.287–306, 1987.

[26] J.R. Rao, "Reasoning about probabilistic parallel programs," ATM Trans. Programming Languages Systems, vol.16, no.3, pp.798–842,

1994.
[27] K.G. Larsen and A. Skou, "Bisimulation through probabilistic testing," Information and Computation, vol.94, no.1, pp.344–352, 1991.
[28] K.J. Sullivan, J.C. Knight, X. Du, and S. Geist, "Information Survivability Control Systems," ICSE, pp.184–192, 1999.
[29] P. Ellison, B.D. Fisher, R. Linger, H. Lipson, T. Longstaff, and N. Mead, "Survivable network systems: An emerging discipline," Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, Nov. 1997.
[30] R.C. Linger, N.R. Mead, and H.F. Lipson, "Requirements definition for survivable network systems," ICRE, 0014, Third International Conference on Requirements Engineering (ICRE'98), 1998.
[31] R. Fagin, J.Y. Halpern, and N. Meggido, "A logic for reasoning about probabilities," Inform and Comput., vol.87, pp.78–128, 1990.
[32] R. Gupta, S.A. Smolka, and S. Bhashar, "On randomization in sequential and distributed algorithms," ACM Comput. Surveys, vol.26, no.1, pp.7–86, 1994.
[33] V.R. Westmark, "A definition for information system survivability," HICSS, vol.9, 90303a, Proc. 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9, 2004.
[34] National Defense Industrial Association Symposium, Proc. Aircraft Survivability 2000, Monterey, CA, Nov. 2000.
[35] J. He, H. Zhu, and G. Pu, "A model for BPEL-like languages," Frontiers of Computer Science in China, vol.1, no.1, pp.9–19, 2007.
[36] A. Koziolek, A. Avritzer, S. Suresh, D.S. Menasche, K.S. Trivedi, and L. Happe, "Design of distribution automation networks using survivability modeling and power flow equations," ISSRE 2013, pp.41–50, 2013.
[37] A. Avritzer, S. Suresh, D.S. Menasche, R.M.M. Leao, E. de Souza e Silva, M.C. Diniz, K.S. Trivedi, L. Happe, and A. Koziolek, "Survivability models for the assessment of smart grid distribution automation network designs," ICPE 2013, pp.241–252, 2013.
[38] D.S. Menasche, R.M.M. Leao, E. de Souza e Silva, A. Avritzer, S. Suresh, K.S. Trivedi, R.A. Marie, L. Happe, and A. Koziolek, Survivability analysis of power distribution in smart grids with active and reactive power modeling, SIGMETRICS Performance Evaluation Review, vol.40, no.3, pp.53–57, 2012.
[39] P.E. Heegaard and K.S. Trivedi, "Network survivability modeling," Computer Networks, vol.53, no.8, pp.1215–1234, 2009.
[40] P.E. Heegaard and K.S. Trivedi, Survivability Modeling with Stochastic Reward Nets, Winter Simulation Conference 2009, pp.807–818, 2009.

Appendix

Below are the algebraic laws for probabilistic programs and the proof for the inference rules of probabilistic refinement introduced in Sect. 3. Here we will confine ourselves to those laws involving the probabilistic choice operator.

Probabilistic choice is idempotent, skew-symmetric and quasi-associative.

Law-1 $P \,{}_r\!\oplus\, P = P$

Proof. By the definition of probabilistic choice, we know that $P \,{}_r\!\oplus\, P = \exists\, prob_1, prob_2 \in Prob_{St} \cdot P[prob_1/prob'] \wedge P[prob_2/prob'] \wedge prob' = r * prob_1 + (1-r) * prob_2$. In particular, if we let $prob_2 = prob_1$, we claim that $P \sqsupseteq P \,{}_r\!\oplus\, P$ in terms of the definition of refinement in the probability program. Obviously, $P \,{}_r\!\oplus\, P \sqsupseteq P \sqcap P$ if we consider the definition of ${}_r\!\oplus$ and $\sqcap$. According to the healthiness condition **H1**: $P = P \sqcap P$, we conclude that $P \,{}_r\!\oplus\, P = P$. □

Law-2 $P \,{}_r\!\oplus\, Q = Q \,{}_{1-r}\!\oplus\, P$

Proof.

$$
\begin{aligned}
& LHS \\
\equiv\ & \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge \\
& prob' = r * prob_1 + (1-r) * prob_2 \\
\equiv\ & \exists prob_2, prob_1 \in Prob_{St}, Q[prob_2/prob'] \wedge \\
& P[prob_1/prob'] \wedge \\
& prob' = (1-r) * prob_2 + (1-(1-r)) * prob_1 \\
\equiv\ & RHS
\end{aligned}
$$

□

Law-3 $(P \,{}_{r1}\!\oplus\, Q) \,{}_{r2}\!\oplus\, R = P \,{}_{s1}\!\oplus\, (Q \,{}_{s2}\!\oplus\, R)$

Where $s1 = r1 * r2$ and $(1-r2) = (1-s1) * (1-s2)$

Proof.

$$
\begin{aligned}
& LHS \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = r2 * (r1 * prob_1 + (1-r1) * prob_2) + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = r1 * r2 * prob_1 + (1-r1) * r2 * prob_2 + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + (r2 - s1) * prob_2 + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + (1-r2) * prob_3 + \\
& ((1-s1) - (1-r2)) * prob_2 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + (r2 - s1) * prob_2 + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + \\
& ((1-s1) - (1-s1)(1-s2)) * prob_2 + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + (r2 - s1) * prob_2 + \\
& (1-r2) * prob_3 \\
\equiv\ & \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge \\
& Q[prob_2/prob'] \wedge R[prob_3/prob'] \\
& \wedge prob' = s1 * prob_1 + ((1-s1) * s2) * prob_2 +
\end{aligned}
$$

$(1-s1)*(1-s2)*prob_3$

$\equiv \exists prob_1, prob_2, prob_3 \in Prob_{St}, P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge R[prob_3/prob'] \wedge prob' = s1*prob_1 + (1-s1)*(s2*prob_2 + (1-s2)*prob_3)$

$\equiv RHS$

□

Law-4 $P_1 \oplus Q = P$

Proof.

LHS

$\equiv \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge prob' = 1*prob_1 + 0*prob_2$

$\equiv \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge prob' = prob_1$

$\equiv RHS$

□

Law-5 $P_r \oplus (Q \lhd b \rhd R) = (P_r \oplus Q) \lhd b \rhd (P_r \oplus R)$

Proof.

LHS

$\equiv \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge (Q \lhd b \rhd R)[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2$

$\equiv \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge ((b \wedge Q) \vee (\neg b \wedge R))[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2$

$\equiv (\exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge (b \wedge Q)[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2) \vee (\exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge (\neg b \wedge R)[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2)$

$\equiv (b \wedge (\exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2)) \vee (\neg b \wedge (\exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge Q[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2))$

$\equiv (b \wedge (P_r \oplus Q)) \vee (\neg b \wedge (P_r \oplus R))$

$\equiv RHS$

□

Law-6 If $P \sqsupseteq Q$, then $P_r \oplus R \sqsupseteq Q_r \oplus R$

Proof.

Obviously, we have $\forall s \in St, prob' \in Prob_{St}, (s, prob') \in P_r \oplus R$, $\exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge R[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2$. We know that $\forall (s, prob') \in P, (s, prob') \in Q$ since $P \sqsupseteq Q$. Thus $(s, prob_1) \in Q$ and $(s, prob') \in Q_r \oplus R$. That is $P_r \oplus R \sqsupseteq Q_r \oplus R$. □

Law-7 $(P \sqcap Q)_r \oplus R = (P_r \oplus R) \sqcap (Q_r \oplus R)$

Proof. On the one hand, according to Law 6, $P_r \oplus R \sqsupseteq (P \sqcap Q)_r \oplus R$ and $P_r \oplus R \sqsupseteq (P \sqcap Q)_r \oplus R$, thus $(P_r \oplus R) \sqcap (Q_r \oplus R) \sqsupseteq (P \sqcap Q)_r \oplus R$. On the other hand, $RHS = \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge R[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2 \vee \exists prob_3, prob_4 \in Prob_{St}, P[prob_3/prob'] \wedge R[prob_4/prob'] \wedge prob' = r*prob_3 + (1-r)*prob_4$ and $LHS = \exists prob_1, prob_2 \in Prob_{St}, (P[prob_1/prob'] \vee Q[prob_1/prob']) \wedge R[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2 = \exists prob_1, prob_2 \in Prob_{St}, P[prob_1/prob'] \wedge R[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2 \vee Q[prob_1/prob'] \wedge R[prob_2/prob'] \wedge prob' = r*prob_2 + (1-r)*prob_2$. Obviously, $\forall (s, prob') \in (P \sqcap Q)_r \oplus R, (s, prob') \in (P_r \oplus R) \sqcap (Q_r \oplus R)$. That is $(P \sqcap Q)_r \oplus R \sqsupseteq (P_r \oplus R) \sqcap (Q_r \oplus R)$. Thus $(P \sqcap Q)_r \oplus R = (P_r \oplus R) \sqcap (Q_r \oplus R)$. □

Law-8 $(P_r \oplus Q); R = (P; R)_r \oplus (Q; R)$

Proof.

RHS

$\equiv (\exists prob_1, prob_2 \in Prob_{St}, (P;R)[prob_1/prob'] \wedge (Q;R)[prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2)$

$\equiv \exists prob_1, prob_2 \in Prob_{St}, \exists prob_3, prob_4 \in Prob_{St}, P[prob_3/prob'] \wedge \uparrow R[prob_3/prob, prob_1/prob'] \wedge Q[prob_4/prob'] \wedge \uparrow R[prob_4/prob, prob_2/prob'] \wedge prob' = r*prob_1 + (1-r)*prob_2$

$\equiv \exists prob_3, prob_4 \in Prob_{St}, P[prob_3/prob'] \wedge Q[prob_4/prob'] \wedge R[prob_0/prob] \wedge prob_0 = r*prob_3 + (1-r)*prob_4$

$\equiv LHS$

□

Inference Rule-1 $P \sqsupseteq_r P$

Proof. $P \sqsupseteq_r P$ since $P \sqsupseteq_S P[r]\perp_S$. □

Inference Rule-2 $$\frac{P \sqsupseteq_r R \quad Q \sqsupseteq_r S}{P \lhd b \rhd Q \sqsupseteq_r R \lhd b \rhd S}$$

Proof. We know that $P \sqsupseteq_S R[r]\perp_S$ and $Q \sqsupseteq_S S[r]\perp_S$, thus $P \lhd b \rhd Q \sqsupseteq_S R[r]\perp_S \lhd b \rhd S[r]\perp_S = (R \lhd b \rhd S)[r]\perp_S$. That is $P \lhd b \rhd Q \sqsupseteq_r R \lhd b \rhd S$. □

Inference Rule-3 $$\frac{P \sqsupseteq_r Q \quad Q \sqsupseteq_s T}{P \sqsupseteq_{r*s} T}$$

Proof. According to the premise, we know that $P \sqsupseteq_S Q[r]\perp_S$ and $Q \sqsupseteq_S T[s]\perp_S$. Thus $P \sqsupseteq_S (T[s]\perp_S)[r]\perp_S = T[r*s]\perp_S$, that is $P \sqsupseteq_{r*s} T$. □

Inference Rule-4 $$\frac{P \sqsupseteq_r Q \quad r \geq s}{P \sqsupseteq_s Q}$$

Proof. $P \sqsupseteq_S Q[r]\perp_S \sqsupseteq_S Q[s]\perp_S$, that is $P \sqsupseteq_s Q$. □

Inference Rule-5 $$\frac{P \sqsupseteq_r Q}{P; R \sqsupseteq_r Q; R}$$

Proof.
$P; R \sqsupseteq_S (Q[r]\perp_S); R$ since $P \sqsupseteq_S Q[r]\perp_S$. Then $(Q[r]\perp_S); R = (Q; R)[r](\perp_S; R) = (Q; R)[r]\perp_S$, thus $P; R \sqsupseteq_r Q; R$. □

Inference Rule-6 $\dfrac{P \sqsupseteq_r Q}{R; P \sqsupseteq_r R; Q}$

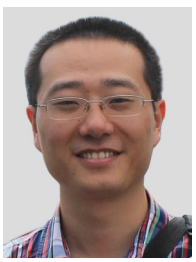
Proof.
$R; P \sqsupseteq_S R; (Q[r]\perp_S)$ since $P \sqsupseteq_S Q[r]\perp_S$. Then $R; (Q[r]\perp_S) = (R; Q)[r](R; \perp_S) = (R; Q)[r]\perp_S$, thus $R; P \sqsupseteq_r R; Q$. □

Inference Rule-7 $\dfrac{P \sqsupseteq_r Q \quad R \sqsupseteq_r T}{P[s]R \sqsupseteq_r Q[s]T}$

Proof. We claim that $P[s]R \sqsupseteq_S (Q[r]\perp_S)[s](T[r]\perp_S)$ according to the monotonicity of the probabilistic choice operator since $P \sqsupseteq_S Q[r]\perp_S$ and $R \sqsupseteq_S T[r]\perp_S$. Furthermore, $(Q[r]\perp_S)[s](T[r]\perp_S) = (Q[s]T)[r]\perp_S$. Thus $P[s]R \sqsupseteq_S (Q[s]T)[r]\perp_S$, i.e., $P[s]R \sqsupseteq_r Q[s]T$. □

Yongxin Zhao is currently an associate Professor at Software Engineering Institute, East China Normal University. He was a postdoc at School of Computing of National University of Singapore, Singapore from 2012 to 2014. His research interests include program analysis and verification, semantics theory, web services and formal methods and he owns more than 28 referred publications.

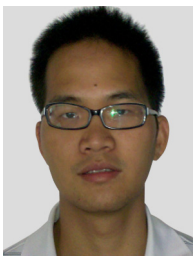
Yanhong Huang received her PhD from East China Normal University, China, in 2014. She was visiting Teesside University from Feb. 2012 to July 2012. Her research topics are embedded and real-time systems, the interrupt mechanism, analysis and verification of operating systems, and semantics theory.



Qin Li received his PhD from East China Normal University, China, in 2011. He is now working as a research staff at The University of Queensland, Australia. His research interests include formal specification and verification, unifying theories of programming, distributed systems and concurrent systems.

Huibiao Zhu is Professor at Software Engineering Institute, East China Normal University, also the Executive Deputy Director of Shanghai Key Laboratory of Trustworthy Computing. He earned his PhD in Formal Methods from London South Bank University in 2005. During these years he has studied various semantics and their linking theories for Verilog, SystemC, web services and probability system. Currently he is the Chinese PI of the Sino-Danish Basic Research Center IDEA4CPS.

Jifeng He is currently a professor and the Dean of Software Engineering Institute, East China Normal University. He is an Academician of Chinese Academy of sciences. He was appointed as the Chief Scientist for several projects of NSFC and 973 program. And he was also appointed as the leader of the creative research group of the National Natural Science Foundation of China. In recent years, he has also been working on the mathematical model about the co-design of software and hardware, design of real-time embedded systems and Cyber Physical system.



Jianwen Li is a PhD graduate student supervised by Jifeng HE and Geguang PU in East China Normal University. He was in University Rice as a visiting student. His research topics are LTL model checking, automata theory, and data flow analysis.

Xi Wu received her BSc in Software Engineering from Software Engineering Institute, East China Normal University in 2011. She is currently a PhD student in Formal Methods with the same institute. Her research interests include process algebra and its applications, program analysis and verification, and web services.