Review of Verification Methodologies for Dynamic Reliability Block Diagram

Jiacong Zhao[1, a *], Dongzhao Zhou[1, b], Shuang Gao and Yao Li[1, c]

[1]Dalian Neusoft University of Information, Dalian, China

[a]zhaojiacong@neusoft.edu.cn, [b]zhoudongzhao@neusoft.edu.cn, [c]gaoshuang@neusoft.edu.cn, liyao@neusoft.edu.cn

Keywords: Dynamic reliability block diagrams; Markov chain; Object-Z; Colored petri nets; Verification methodology

Abstract. Reliability modeling tool Dynamic Reliability Block Diagrams (DRBD) is widely used in modeling reliability of large and complex system. It gains highly achievement of dynamic behaviors like dynamics, dependencies, redundancy and load sharing. In order to guarantee the modeling results of DRBD are correct, widely used verification methodologies like Markov chain, Binary Decision Diagrams (BDD), Object-Z and Colored Petri Nets (CPN) are introduced to verify the DRBD model. The achievement of behavioral properties of DRBD, states and events modeling, whether the method is user-friendly to all kinds of system, whether the modeling process is productive, formal and automatic, whether there are formal tools to check the verification result of DRBD are picked as criteria to evaluate and compare these verification methods. The evaluation shows CPN is the best method.

Introduction

As the widely use of computer system in diverse industries, the requirement for achieving the reliability of large and complex system is increasingly important. It is stated that a system is composed by components and these components are designed to achieve desirable performance [1]. It indicates that the reliability of a system is directly affected by the arrangement, quantities and qualities of these components [1]. In this situation, many tools which are focused on components modeling are proposed to model the system reliability, such as Reliability Block Diagram (RBD) [2], Fault Tree (FT) [3] and Dynamic RBD (DRBD) [1,4,5]. Comparing with alternative tools, the evaluation results in [4,5] have shown that DRBD is the most productive and accurate method. The whole process for DRBD to model complex and large systems is summarized as following steps: system specification, sub-systems identification, structural linking, dynamic linking and reiteration [6]. Although this multiple steps process can models all components and behaviors of the system, flaws and faults are easily introduced in this complex process [5]. Thus, methodologies like Markov Chain (MC) [9], Petri Net (PT) [5] and Colored PN (CPN) [5,6,7] are used to verify DRBD to guarantee the accurate reliability modeling result. Based on the evaluation and comparison of DRBD's behaviors properties summarized in [1,4,5,8], this report takes the position that comparing with alternative tools, CPN is the most productive and accurate method to verify DRBD. The following parts will firstly give a explanation of DRBD model, then describes three widely used DRBD verification methodologies. The next part will give the comparison and evaluation of these methodologies. And finally turn to the conclusion.

DRBD

DRBD shares the same theory with RBD in terms of static behaviors modeling [1,4,5,6,8], but distinguishes itself by introducing different kinds of controller blocks [4,5] which are based on the state-events working mechanism [4,6] to analyze dynamic behaviors. RBD graphically figures out arrangement of system components and connections [10,11]. It decides the overall system reliability by analyzing the given reliability of each component [10,11] and components' connections. Fig.1

shows a basic example of RBD. It shows the serial connections [1,11] between components A and B, and the parallel connections [1,11] between C and D within B. Hence, one component of DRBD is named as State-based reliability block (SRBD) [1,4,5] which does the same work as RBD. Controller blocks deal with dynamic behaviors and they are based on the state-event mechanism [5]. Briefly, DRBD characterizes each component's condition with states and states' evolution with events [1,5,7,8]. In time-variant systems, every component's condition is changing as time passes [8]. Correspondingly, the component's state is changing among Active, Standby and failed (Fig.2) [8]. Component works without problem is Active, while failure of component is represented by Failed. Standby means the component is not work and has no effect on the system [4,5,6]. Additionally, Standby is divided into hot-standby, warm-standby and cold-standby [4]. Then, event is classified into four categories: Failure, Weak-Up, Sleep and Repair [4,5,6,8]. State that exchanges from Active/Standby to Failed is Failure [4]. State evolves from Active/Standby to Active is Weak-Up [4]. Sleep is the transformation from Active/Standby to Standby [4]. Repair describes state from Failed to Standby/Active [4]. Fig.3 [4] shows a more comprehensive DRBD state-events working mechanism, but without Repair (sometimes the failure cannot be back-up). It describes all the states (represented by rounded rectangles) and the events (described by directed arcs) among states. In conclusion, based o n this state-events mechanism, the process for DRBD to model system reliability is mainly achieved by three aspects:

Characterizing components by states: Active, Standby and Failed [4, 8].

Specifying components' static connections (structure relationships): Serial, Parallel and Hybrid structure [12].

Specifying components' dynamic connections (reliability relationships) by events: Failure, Weak-Up, Sleep and Repair [4,8,12].

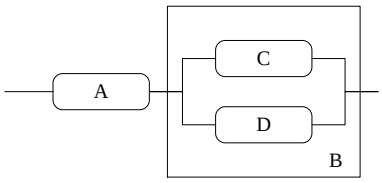

Figure 1. Basic RBD structure

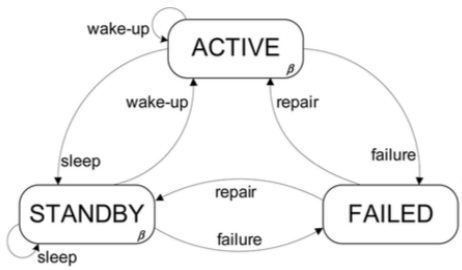

Figure 2. Basic DRBD State-Event Mechanism

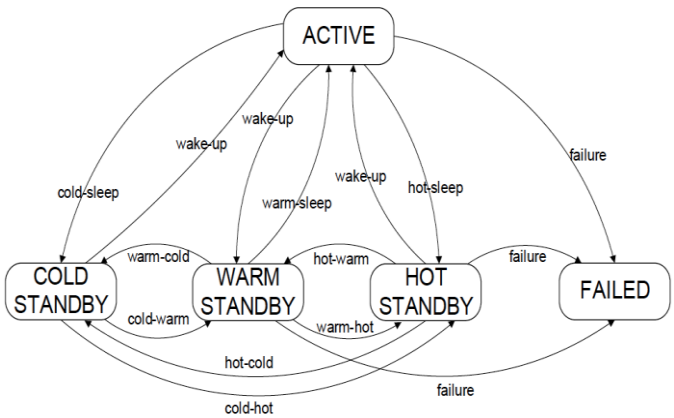

Figure 3. Complete DRBD State-Event Mechanism

Verification Methodologies for DRBD

In order to guarantee the target system's reliability properties are represented without problems in DRBD model. In this situation, formal methodologies are proposed to verify DRBD [13]. Based on

current review, these methodologies experienced a long development process. From partly modeling DRBD's features to comprehensively model all possible behavioral properties. From manually dealing with the complex verification process to automatically verify DRBD. Based on current review, the following parts will discuss the three most widely used methodologies for verifying DRBD: Markov-based combinational method, Object-Z and CPN.

Markov-based Combinational Method. This Markov-based methodology is a combinational method which verifies static and dynamic parts separately [9]. Its two steps modeling process is summarized in [9]. Firstly, modeling the whole DRBD model into multi-independent sub-DRBD modules. These independent modules own their specific blocks which do not appear in other modules of the system [9]. Additionally, modules who involve dependencies and dynamics are identified as dynamic modules (controller blocks), others are defined as static modules [9]. Secondly, using Markov chain to verify controller blocks and binary decision diagrams (BDD) [9] to check static parts, respectively.

Markov chain verifies states and events in a recursive fashion [9] until tracking all states of DRBD system. Specifically, the Markov model based on the following equations: (t)=AP(t) [9], P(t)=[P1(t), P2(t), ..., Pn(t)] [9]. P' represents a set of differential equations. P is the state probability vector and Pi(t) represents module's probability in state i at time t [9], n is the number of presented states and A is an transaction rate matrix of n × n [9]. The mentioned set of differential equations is solved by: ∑(n ,i=1) P (t) = 1 [2]. BDD is a powerful tool for modeling large static systems that with binary-state: Active and Failed [14]. The reason for using BDD to model the static parts is as the size of DRBD model increasing the size of Markov is growing exponentially [9]. Hence, BDD is introduced to fill the gap of Markov chain. Therefore this methodology can deal with DRBD behaviors properties. However, Markov is not suitable to model large and complex system for the restriction of recursive mechanism. Furthermore, BDD is only suitable to binary-state systems and cannot model multi-state systems.

Object-Z. Object-Z is an formal specification language for large and complex systems modular design [15,16], which enjoys great capabilities in modeling data and state [9]. Both [9] and [17] give the examples of using Object-Z to specify two kinds of DRBD controller blocks: state dependency block and spare part blocks. For illustration purpose, Fig.4 [9] and Fig.5 [9] separately shows the specification of SDEP and SPARE with Event. However, only state schemas are defined, the operation schemas for modeling dynamic behaviors of gates are missing. Based on the examples, the process for Object-Z to specify a DRBD model is summarized mainly as following steps. Firstly, define events of a component with: Activation, Deactivation and Failure [9,17], which corresponding to a state Active, Standby and Failed. Secondly, defining a state dependency as ActivateTrigger, DeactivateTrigger, and [9,17]. Thirdly, specifying the whole defined DRBD into Object-Z formal language. Based on the examples of [9,17]. It can be seen that what Object-Z does is to specify DRBD components and constructs to formal semantics [9,17]. Although it precisely defines all possible DRBD behaviors, there is lacking of analyzing and verifying tools to verify Object-Z's result feasibly and straightforwardly [17].

$Standby ::= Hot \mid Cold \mid Warm$

SPARE

$primaryUnit : Component$
$alternativeUnits : \mathbb{P}\ Component$
$nAlternatives : \mathbb{N}$
$alternative : \mathbb{N} \rightarrow Component$
$switch : \mathbb{T} \times Component \times Event \rightarrow \mathbb{T} \times Component \times Event$

$\forall c \in alternativeUnits \bullet c \neq primaryUnit$
$nAlternatives = \#alternativeUnits \wedge nAlternatives > 0$
$\{1, 2, ..., nAlternatives\} = \mathrm{dom}\ alternative$
$\mathrm{dom}\ switch = \{(t, c, e) \mid t \in \mathbb{T}, c \in \{primaryUnit\} \cup$
$\{(alternative(i)) \mid 1 \leq i \leq nAlternatives - 1\},$
$e \in \{Deactivation, Failure\}\}$

INIT

$trigger.state = Active$
$\forall i, where\ 1 \leq i \leq nAlternatives \bullet alternative(i).state = Standby$

PrimarySwitch

$\Delta(primaryUnit, alternativeUnits)$
$t? : \mathbb{T}$

$\forall e \in \{Deactivation, Failure\} \bullet$
$switch(t?, primaryUnit, e) = (t? + \delta_c, alternative(1), Activation)$
$(primaryUnit.state = Active) \wedge$
$(primaryUnit.state' \in \{Standby, Failed\})$
$\wedge\ (alternative(1).state' = Active)$

AlternativeSwitch

$\Delta(alternativeUnits)$
$t? : \mathbb{T}, i? : \mathbb{N}$

$\forall e \in \{Deactivation, Failure\}, 1 \leq i? \leq nAlternatives - 1 \bullet$
$switch(t?, alternative(i), e) =$
$(t? + \delta_c, alternative(i + 1), Activation)$
$\wedge\ (alternative(i?).state = Active)$
$\wedge\ (alternative(i?).state' \in \{Standby, Failed\})$
$\wedge\ (alternative(i? + 1).state = Standby)$
$\wedge\ (alternative(i? + 1).state' = Active)$

Figure 4. Object-Z Specification of SDEP Block Block

$Event ::= Activation \mid Deactivation \mid Failure$

SDEP

$trigger : Component$
$targets : \mathbb{P}\ Component$
$nTargets : \mathbb{N}$
$triggerEvent : Event$
$targetEvents : Component \rightarrow Event$
$sdep : \mathbb{T} \times Component \times Event \rightarrow \mathbb{P}(\mathbb{T} \times Component \times Event)$

$nTargets = \#targets \wedge nTargets > 0 \wedge targets = \mathrm{dom}\ targetEvents$
$\forall c \in targets \bullet c \neq trigger$
$\wedge probability(c \mid triggerEvent) \neq probability(c)$
$\wedge\ probability(triggerEvent \mid c) = probability(triggerEvent)$
$\{(t, trigger, triggerEvent) \mid t \in \mathbb{T}\} = \mathrm{dom}\ sdep$

ActivateTrigger

$\Delta(trigger, targets)$
$t? : \mathbb{T}$

$(triggerEvent = Active) \wedge (trigger.state' = Active)$
$\forall c \in targets \bullet$
$(t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent)$
$\wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active)$
$\vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby)$
$\vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))$

DeactivateTrigger

$\Delta(trigger, targets)$
$t? : \mathbb{T}$

$(triggerEvent = Deactivation) \wedge (trigger.state' = Standby)$
$\forall c \in targets \bullet$
$(t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent)$
$\wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active)$
$\vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby)$
$\vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))$

FailTrigger

$\Delta(trigger, targets)$
$t? : \mathbb{T}$

$(triggerEvent = Failure) \wedge (trigger.state' = Failed)$
$\forall c \in targets \bullet$
$(t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent)$
$\wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active)$
$\vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby)$
$\vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))$

Figure 6. Object-Z Specification of SPARE

CPN. CPN is an extension of PN by adding the Standard Modeling Language (SML) [18]. PN is an easy to use and graphical representation tool [18, 19], which can graphically define DRBD's states, events and its behaviors properties [19]. SML deals with data and creates models for all parameters of the system. It can be seen that CPN can not only graphically model states and events of DRBD, but represent DRBD model into formal modeling language. The method for verifying DRBD automatically is proposed in [5]. Hence, based on [5], steps for CPN to verify DRBD are specified as follows. Firstly, representing all components of DRBD by Backus-Naur form (BNF). Secondly, proposing Reliability Markup Language (RML) [5] as an interface to link DRBD and CPN. The RML is extended from the Extensible Markup Language (XML)[5,8,20]. Additionally, the design of RML is based on BNF definition of DRBD model [5]. Hence, RML can formally represent all the BNF format based DRBD blocks with formal markup language formats [5,2]. Thirdly, converting the RM-represented DRBD into CPN. SML is one component of CPN, it is extended from the XML and shares the same theory with RML. Hence, this step is mainly focus on linking these two XML-based modeling languages together. This enables the conversion process easily and precisely. Finally, using the CPN tools [5] to verify the converted CPN. CPN tools can trace CPN model's deadlock states [5], so as to identify the design flaws and the error states in the CPN model. It is therefore can be seen that this CPN verification process not only precisely achieve all behaviors of DRBD, but automate the modeling process which extremely reduces faults, errors and flaws that introduced by manual work.

Comparison and Evaluation

Based on the behavioral properties of DRBD in [1,4,5,8], its state-events working mechanism and the whole process of the mentioned three method to verify DRBD. This report picks the following criteria to evaluate the mentioned three works. Firstly, whether the method can model all DRBD behavioral properties like dynamic, dependency, redundancy and load-sharing. These three methodologies can successfully deal with all the properties. Secondly, whether these methods can

successfully model all states and events. Both Object-Z and CPN can precisely cover all states and components of DRBD model. However, the BDD of Markov-based method can only model the Active and Failed states, but cannot deal with the Standby. This limits it capability to model multi-state systems. Thirdly, whether the methodology is user-friendly to all systems. The Markov-based approach is apparently not suitable to large size DRBD system. Object-Z and CPN can be used to all kinds of DRBD model. Finally, although both Object-Z and CPN describe the DRBD into formal modeling language to alleviate the subtle flaws. The conversion result of CPN can be modeled by formal CPN tools automatically. However, Object-Z only specifies the whole DRBD model but without tools to check the generated Object-Z result.

Based on the working mechanism of DRBD and characteristics of system reliability, the picked criteria for evaluating and comparing is accuracy, automation, achievement of static and dynamic features and dynamic behaviors. Table.1 shows the comparison and evaluation of these methodologies. It shows that BDD, MMDD and MC can only model partly of the whole system of DRBD. Although the combination of them can model the whole DRBD features, it doses not overcome the shortcoming of each technique. Object-Z performs well in verifying DRBD, however, Object-Z only models the state dependency and spare part missing the dynamics and load sharing. CPN can model DRBD precisely and comprehensively and the covered CPN can verify automatically. Therefore, currently CPN is the best verification method for modeling DRBD among the mentioned techniques.

Table 1 Comparison of Verification Methodologies

Criteria / Methodologies	Accur-acy	Autom-ation	Static Modeling	Dynamic Modeling	Dynamic Behaviors Modeling			
					dynamics	dependencies	redundancy	load sharing
BDD	Yes	No	Yes	No	No	No	No	No
MMDD	Yes	No	Yes	No	No	No	No	No
MC	No	Yes	No	Yes	Yes	Yes	Yes	Yes
MC&BDD	No	No	Yes	Yes	Yes	Yes	Yes	Yes
MC&MMDD	No	No	Yes	Yes	Yes	Yes	Yes	Yes
Object-Z	Yes	Yes	Yes	Yes	No	Yes	Yes	No
CPN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Conclusion

System reliability is achieving increasing attention as the widely use of large and complex systems, especially computer system. High reliability should be achieved by the accurate and productive reliability modeling tools and the formal verification methodologies to guarantee the precise results of these tools. In this situation, DRBD is stated as the most productive reliability modeling tool, however it introduces flaws easily during the complex modeling process. In order to guarantee the correct result of DRBD, verification methodologies like MC, Object-Z and CPN are proposed to deal with the weakness of DRBD. Based on current review, comparing with alternative methodologies for DRBD verification, CPN not only fully achieve all behavioral properties, states and events of DRBD, but can identify DRBD's flaws and faults automatically. Thus, this paper summarized that, based on current review and comparison, CPN is the best formal methodology to verify DRBD.

References

[1] Distefano, Salvatore, and Antonio Puliafito. "Dynamic reliability block diagrams: Overview of a methodology." ESREL. Vol. 7. 2007.

[2] Rausand, M., and A. Hoyland. "System Reliability Theory: Models and Statistical Methods, 2003."

[3] Vesely, William E., et al. Fault tree handbook. No. NUREG-0492. Nuclear Regulatory Commission Washington DC, 1981.

[4] S. Distefano and L. Xing. A new approach to modeling the system reliability: dynamic reliability block diagrams. In RAMS'06 proceedings, 2006, pp. 189-195.

[5] R. Robidoux, H. P. Xu, L. D. Xing, and M. C. Zhou, "Automated modelling of dynamic reliability block diagrams using coloured Petri nets," IEEE Trans. Syst., Man, Cybern. A, Syst., Humans, vol. 40(2010)No. 2, pp. 337–351.

[6] R. Manian, J. Dugan, D. Coppit, and K. Sullivan, "Combining various solution techniques for dynamic fault tree analysis of computer systems," in Proc. 3rd Int. Symp. High-Assurance Systems Engineering (HASE'98), Washington, D.C., USA, 1998, pp. 21–28.

[7] Distefano S, Puliafito A. Dynamic reliability block dagrams: overview of a methodology. In: Proceedings of the safety and reliability conference (ESREL07); 2007.

[8] S. Distefano and A. Puliafito, "Dependability Evaluation with Dynamic Reliability Block Diagrams and Dynamic Fault Trees," IEEE Trans. Dependable Secur. Comput., vol. 6(2009)No. 1, pp. 4–17.

[9] Xing, L., Xu, H., Amari, S. V., & Wang, W. A New Framework for Complex System Reliability Analysis: Modeling, Verification, and Evaluation.

[10] A. Abd-Allah, "Extending Reliability Block Diagrams to Software Architectures," Technical Report USC-CSE-97-501, Dept. of Computer Science, Univ. Southern California, 1997.

[11] R. Duke, G. Rose, and G. Smith, Object-Z: a specification language advocated for the description of standards, Computer Standards and Interfaces (1995) Vol. 17, North-Holland, pp. 511-533.

[12] Robidoux, Ryan Mark. Automated verification of a computer system reliability model. Diss. University of Massachusetts Dartmouth, 2007.

[13] L.Xing, Efficient Analysis of Systems with Multiple States, Proceedings of The IEEE 21st International Conference on Advanced Information Networking and Applications, (2007) May 21-23; Niagara Falls, Canada, pp. 666-672.

[14] J. B. Dugan and S. A. Doyle, New results in fault-tree analysis, Tutorial Notes of the Annual Reliability and Maintainability Symposium, (1997) .

[15] R. Duke, G. Rose, and G. Smith, Object-Z: a specification language advocated for the description of standards, Computer Standards and Interfaces (1995) Vol. 17, North-Holland, pp. 511-533.

[16] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking, MIT Press (2001).

[17] Xu, Haiping, and Liudong Xing. "Formal semantics and verification of dynamic reliability block diagrams for system reliability modeling." In Proc. 11th International Conference on Software Engineering and Applications (SEA 2007), 2007, pp. 155-162.

[18] A. V. Ratzer, L. Wells, H.M. Lasen, M. Laursen, J.F. Qvortrup, et al., "CPN Tools for editing, simulating and analysing coloured Petri nets," in Proc. 24th Int. Conf. Application and Theory of Petri Nets, Eindhoven, Netherlands, Jun. 2003, pp. 450-462.

[19] Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured Petri nets. International Journal on Software Tools for Technology Transfer 2 (1998) ,98–132

[20] C. Goldfarb and P. Prescod, The XML Handbook, Upper Saddle River, NJ, Prentice Hall, 2000.