A New Framework for Complex System Reliability Analysis: Modeling, Verification, and Evaluation

Liudong Xing[a*], Haiping Xu[a], Suprasad V. Amari[b], and Wendai Wang[c]

[a]*University of Massachusetts Dartmouth, 285 Old Westport Road, Dartmouth, MA 02747, USA*

[b]*Relex Software Corporation, 540 Pellis Road, Greensburg, PA 15601, USA*

[c]*Applied Materials,3320 Scott Blvd., M/S 1117, Santa Clara, CA 95052, USA*

Abstract - Due to recent advances in science and technology, computing and engineering systems are evolving toward enabling much larger collaboration and handling missions that are more complicated. The increasing complexity and scale imply that reliability problems will not only continue to be a challenge but also require more accurate models and efficient solutions. In this paper, a new reliability framework called dynamic reliability block diagrams will be presented to address the above challenge. The framework uses modeling, formal specification, formal verification and validation, and model evaluation for accurate reliability analysis of complex computer-based systems. The basics and application of the DRBD approach will be illustrated through the analysis of several examples.

Keywords: *dynamic reliability block diagram, Object-Z formalism, formal verification, colored Petri net, modular approach*

1. Introduction

With the rapid advances in science and technology, modern computing and engineering systems are designed to enable larger collaboration and handle more complicated missions, and thus they exhibit more complex dependent and dynamic behaviors. Failure to model these behaviors accurately results in over/understated system reliability, which renders reliability analysis less effective in system design and tuning activities. This paper presents a new modeling techniques based on reliability block diagrams (RBD) that aims to fully capture complex system behaviors, leading to more accurate analysis of complex system reliability than existing methods.

RBDs are currently one of the widely used techniques for system reliability studies due to their simplicity [1, 2]. An RBD is a success-oriented network describing the functions of a system. Specifically, each RBD model consists of an input point, an output point, and a set of blocks; each block representing a physical component that needs to function correctly. The blocks in the RBD are arranged in a way that illustrates the proper combinations of working components that keep the entire system operational [2].

[*]Corresponding author: Email: ldxing@ieee.org, Phone: +1 508 999 8883, Fax: +1 508 999 8489

Typically, if there is at least one path connecting between the input and output points, the system is operational. On the other hand, the failure of a component is indicated by the removal of the corresponding block in the RBD model, and the system failure occurs when enough blocks are removed to interrupt the connection between the input and output points.

The main virtues of the RBD model are that it is easy to read, and it is readily understood by customers, people who sell systems, engineers who design and test systems, and managers who make decisions on systems. With knowledge of the system, design engineers can easily construct and modify the corresponding RBD model, and communicate with people from different disciplines [3]. However, the traditional RBD model has a distinct disadvantage that it cannot capture the dependent and dynamic behaviors of large and complex systems; it can only model the system whose failure criteria are expressed in terms of combinations of component fault events. Recently, several commercial reliability modeling software packages [4, 17, 18, 23] incorporated standby junctions or containers for modeling the standby redundancy and some other features such as limited repair personnel, load-sharing blocks or containers, common-spare pools, and phased-mission systems. These efforts on extending the traditional RBD for modeling dependent and dynamic behaviors are piecemeal. In this paper, we present a new modeling framework called dynamic reliability block diagrams (DRBD) that will extend the traditional RBD by fully considering the various dependencies and system dynamics. The DRBD framework will consist of new model development, formal specification, formal verification and validation, and model evaluation for the reliability analysis of complex systems with dependent and dynamic behaviors. Note that reference [3] introduced a set of DRBD constructs as an extension to the traditional RBD model for modeling dynamic and dependent behaviors; however, these DRBD constructs are complicated and difficult to use. In addition, no formal specification and verification has been addressed in [3].

The remainder of the paper is organized as follows. Section 2 gives an example that will be used to illustrate the DRBD approach, in particular, the DRBD modeling and evaluation. Section 3 presents the new DRBD components. Section 4 presents the formal specification of DRBD models using Object-Z and describes the colored Petri nets based formal verification of DRBD models. Section 5 presents the solution for evaluating the DRBD model. In the last section, conclusions as well as directions for future work will be given.

2. An Illustrative Example

To illustrate the DRBD approach, we consider a hypothetical example computer system (HECS) adapted from [5, 6]. As shown in Figure 1, the system is composed of four subsystems: a processor subsystem, a bus subsystem, a memory subsystem, and an I/O subsystem. The processor subsystem is further composed of dual redundant processors (P_1 and P_2) that share a common cold spare (P_S), which can replace either of the primary processors upon failure. The memory subsystem is further composed of five memory units (M_1, M_2, M_3, M_4, and M_5) accessed through two memory interface units (MIU_1 and MIU_2). If the MIU fails, the memory units connected to it become unusable or inaccessible. M_1 and M_2 are connected to the bus via MIU_1. M_4 and M_5 are connected to the bus via MIU_2. M_3 is connected to the bus via both interfaces. Thus, M_3 is accessible as long as either of interface uints is operational. The bus subsystem is composed of two buses (B_1 and B_2). The entire computer system is operational if at least one processor, 3-out-of-5 memory units, at least one bus, and the I/O unit are functioning.

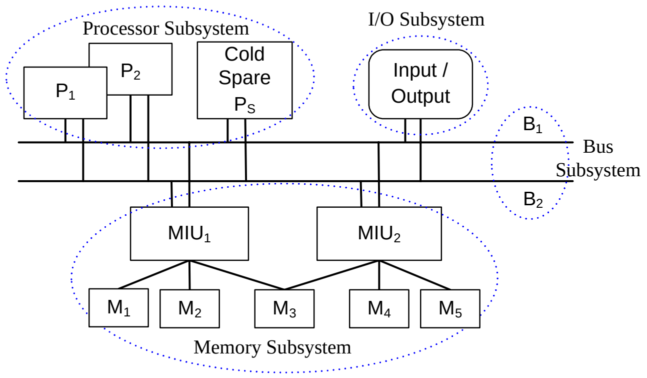


Fig. 1. An Example Computer System (Adapted From [5, 6])

For simplicity, we assume all components in the system fail exponentially. Table 1 gives the constant failure rates 'λ' for all components. Note that the exponential failure distribution is only required for dynamic components and our methodology is applicable to arbitrary component failure distributions for static components. The mission time of 1000 hours will be considered in our analysis.

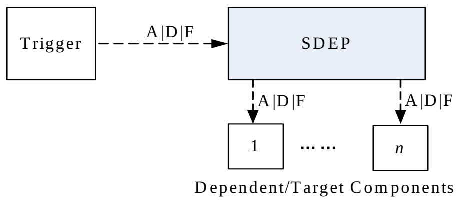
Table 1: Component Failure Rates for HECS

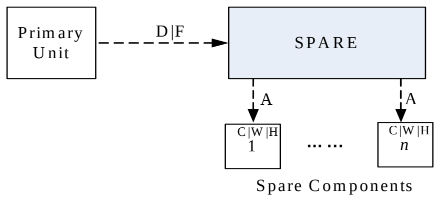
Subsystem	Processor			Memory		I/O	Bus	
Component	P_1	P_2	P_s	M_i, i=1…5	MIU_i, i=1,2	I/O	B_1	B_2
λ (10^{-6}/hr)	1.0	1.0	1.0	2.0	4.0	5.0	1.0	1.0

3. Reliability Modeling using Dynamic Reliability Block Diagrams

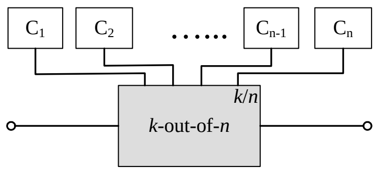
In this section, we identify typical dynamic behaviors of complex systems through the analysis of the HECS system and propose new DRBD components to model these behaviors.

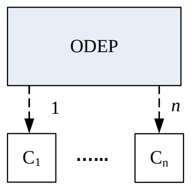
Consider the memory subsystem in the HECS (Figure 1). The state of a memory unit relies on the state of the MIU to which it is connected. Specifically, the failure of the MIU causes the connected memory units to become unusable. The state dependence abounds in many other real applications. For example, when communication is achieved through a network interface card, the failure of the card makes the connected components inaccessible in the computer network [7]. In the clustered wireless sensor network example given in [8], for conserving limited energy of sensor nodes, the activation (or wake-up) of one subset of sensor nodes in a cluster leads to the deactivation (or sleeping) of the other subset of sensor nodes in the same cluster. To model the various state dependencies, a new DRBD component called State Dependency (SDEP) block is proposed. Figure 2(a) illustrates the general structure of the SDEP block, where the annotation letters *A*, *D*, and *F* represent activation, deactivation, and failure, respectively. Occurrence of the trigger event will force all the dependent events to occur. Both the trigger event and the dependent events can be the activation, deactivation or failure of a system component. Therefore, the SDEP block can model nine different types of dependencies among the system components: (*A*, *A*), (*A*, *D*), (*A*, *F*), (*D*, *A*), (*D*, *D*), (*D*, *F*), (*F*, *A*), (*F*, *D*), and (*F*, *F*). Reconsider the three examples described above, (*F*, *D*) dependency exists between the MIU and its connected memory units; (*F*, *D*) dependency exists between the network interface card and its connected components; and (*D*, *A*) or (*A*, *D*) dependency exists between the two subsets of sensor nodes in the wireless sensor network.

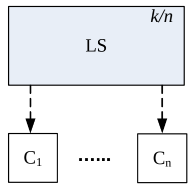




(a) State Dependence (b) Spare







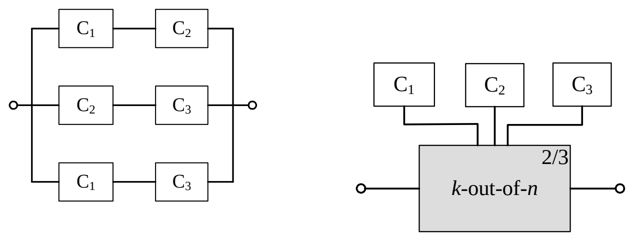
(c) *k*-out-of-*n* (d) Order Dependence (e) Load Sharing

Fig. 2. Proposed New DRBD Models

Note that the trigger event in the SDEP block may involve multiple components. When this happens, logic AND relationship will apply. For example, in the memory subsystem of the HECS, (the failure of MIU_1) AND (the failure of MIU_2) leads to the deactivation of memory unit M_3. This situation can be easily modeled using the SDEP block, as we will show in Figure 4 shortly. It is also important to point out that the existing dynamic fault trees (DFT) model [7] can only model the (*F*, *F*) or (*F*, *D*) state dependency using the functional dependency (FDEP) gate; it cannot capture the other state dependency behaviors. Hence, as compared to the DFT approach, the DRBD is more powerful and flexible in modeling dependent behaviors among system components.

For modeling the cold standby sparing processor subsystem in the HECS, we propose another new DRBD component called SPARE block. Figure 2(b) illustrates the general structure of the SPARE block, where *A*|*D*|*F* have the same meaning as in the SDEP block, and *C*|*W*|*H* means *cold*|*warm*|*hot* spare. Specifically, a cold spare is unpowered until needed to replace a faulty component; a hot spare operates in synchrony with a primary or online component, and is prepared to take over the work at any time [9]. A warm spare is a trade-off between the cold and hot spares in terms of reconfiguration time and power consumption [7]. In general, the SPARE block can model the behavior that the deactivation or failure of the primary component leads to the activation of a spare component, which could be in cold, warm, or hot standby state. In other words, using the SPARE block, a reconfiguration can be triggered by either the deactivation or the failure of a primary component. All the spare units will be used in the specified order, i.e., from left to right. Note that some commercial products on RBD also allow the sparing behavior to be modeled using standby junctions and/or blocks; however, they did not differentiate between the case where a reconfiguration is triggered by the deactivation of a primary component and the case where the reconfiguration is triggered by the failure of a primary component.

In the traditional RBD model, to represent a *k*-out-of-*n* structure, the *n* components will be firstly grouped in a series connection of *k* components and these series structures will then be connected in parallel. The total number of blocks contained in a *k*-out-of-*n* structure is the number of components contained in each series structure (i.e., *k*) multiplied by the number of series structures (i.e., C_n^k): $k \times C_n^k = k \times n!/[k!(n-k)!]$. For example, Figure 3 (a) illustrates the RBD model of a 2-out-of-3 structure that consists of 6 blocks. For the *3*-out-of-*5* structure in the memory subsystem of the HECS, 30 blocks must be drawn in the traditional RBD representation. To simplify the representation of a general *k*-out-of-*n* structure, similar to the *k*-out-of-*n* junctions used in some commercial RBD software packages [23], we propose a new DRBD component called *k*-out-of-*n* block, using which only *n* blocks are connected as shown in Figure 2(c). The simplified representation of 2-out-of-3 structure using the proposed block is shown in Figure 3(b).



(a) Traditional RBD (b) DRBD

Fig. 3. Representation of a 2-out-of-3 Structure

Using the proposed SDEP, SPARE, and *k*-out-of-*n* blocks as well as the traditional RBD notations, we now can construct the DRBD model for the entire HECS system as shown in Figure 4. Since the operation of the system requires only one processor to be functional, the three processors (P_1, P_2, P_S) are connected in a parallel structure. The cold sparing behavior is modeled through two SPARE blocks. The *3*-out-of-*5* operational criteria for the memory subsystem are modeled via a *k*-out-of-*n* block connected to the five memory units. The state dependencies of the five memory units upon the pair of memory interface units are modeled via three SDEP blocks, with the memory interface units (MIU_1 and MIU_2) as trigger inputs and the memory units (M_1, M_2, M_3, M_4, and M_5) as the dependent events. All the subsystem DRBD models are connected in series to form the entire system DRBD model since the system fails when any of the subsystems fails.

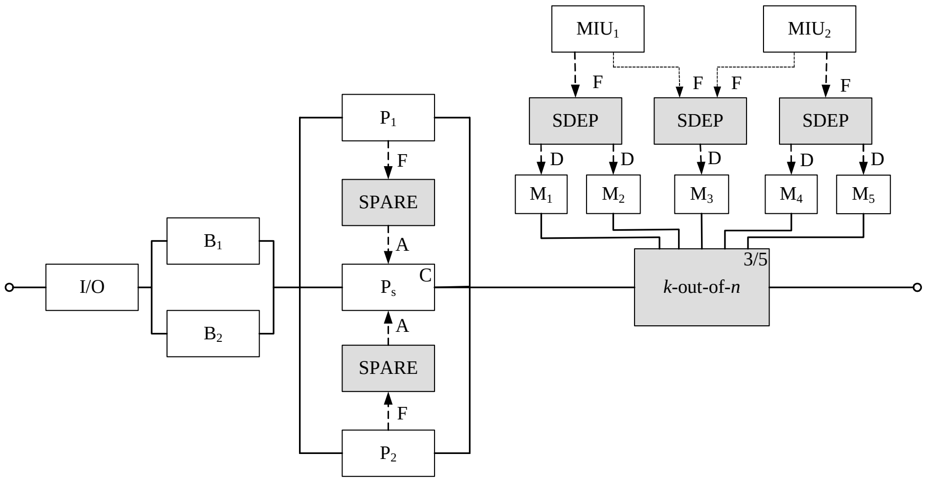


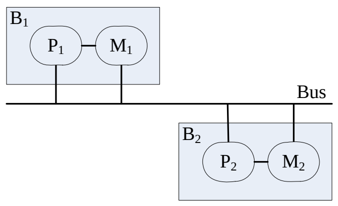
Fig. 4. The DRBD Model of the HECS

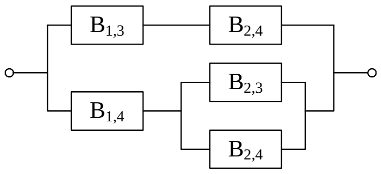
Besides the capability of modeling the various state dependencies and sparing behaviors (as illustrated above), the DRBD will also consist of components for modeling other dynamic behaviors such as sequence/order dependence and load sharing as well as multistate behavior. Due to the space limitation, we only brief the basics of these DRBD components as follows. More case studies involving these behaviors will be studied in our future work.

The order dependence occurs when the order that input events occur is important. For example, consider a fault-tolerant system with a primary component and a standby spare connected with a switch controller [10]. If the switch controller fails after the primary component fails and thus the standby component is switched into active operation, then the system can continue to operate. However, if the switch controller fails before the primary component fails, then the standby component cannot be activated, and the system fails when the primary component fails even though the spare part is still operational. In summary, the failure of the fault-tolerant system depends on the occurrence order of two failure events: the failure of the switch and the failure of the primary component. Figure 2(d) illustrates the general structure of the proposed Order Dependence (ODEP) block. Note that it is not necessary to connect the ODEP block in the main structure of the system DRBD model. The ODEP block is typically used to detect whether the input events, which are connected to the main structure, occur in the specified order (from 1 to *n*).

The load sharing represents a condition where multiple components share the same workload. These components usually perform the same task and exhibit different failure characteristics when one or more of them have failed or been deactivated. Specifically, as individual components fail, the failure behavior of the remaining components in the load sharing redundancy will change since they now have to carry a heavier load [24]. Some commercial software tools [4] use a load sharing container or block to model the load sharing behavior and assumes that the container fails only when all the components in the container have failed. As illustrated in Figure 2(e), our proposed Load Sharing (LS) block is more flexible than the container defined in the existing tools because the LS block allows defining a threshold for failure. Specifically, the annotation "*k/n*" in the LS block represents that there are *n* components sharing the same workload, corresponding to the *n* input events of the block; and at least *k* out of *n* components must be functioning for the load sharing system to work. When fewer than *k* components are functioning, each component will be overloaded and become failed. The entire block is thus regarded as being failed. When $k = 1$, the function of the LS block is the same as the load sharing container used in [4].

Lastly, each block in the DRBD model may represent a state of a multistate component, instead of representing a binary-state component in the traditional RBD model. Specifically, in the traditional RBD model, if there is a connection between the two end-points of a block, we say that the component represented by the block is functioning. In the multistate DRBD model, if there is a connection between the two end-points of a block, we say that the component is in a specific state represented by the block. For example, consider the multistate computer system in [11, 12]. As shown in Figure 5(a), the system consists of two boards B_1 and B_2. Each board contains a processor and a memory and can be considered as a single component with four states: $B_{i,4}$ (both P and M are functional), $B_{i,3}$ (M is functional, but P is down), $B_{i,2}$ (P is functional but M is down), and $B_{i,1}$ (both P and M are down). The entire system has three states: S_3 (at least one processor and both memories are functional), S_2 (at least one processor and exactly one memory are functional), and S_1 (no processor or no memory is functional). For illustration purpose, the DRBD model for the system being in state S_3 is shown in Figure 5(b). According to the DRBD model, it is easy to see that the system is in state S_3 if the board B_1 is in state 3 and B_2 is in state 4; or if B_1 is in state 4 and B_2 is in state 3 or 4.





(a) System structure (b) DRBD

Fig. 5. Example Multistate System and DRBD Model

4. Formal Specification and Verification of DRBD Models

To provide the denotational semantics for the development of DRBD models in a precise manner and help eliminate ambiguity in a constructed DRBD model, it is necessary to formally specify the DRBD modeling constructs. For example, Figure 6 shows the DRBD model of a system with state dependencies and sparing relationship among components *C*1, *C*2, *C*3 and *C*4. Here, *C*4 is a cold spare for *C*2. Suppose component *C*1 fails at some time. According to the state dependency (*F*, *F*) from *C*1 to *C*2, and (*F*, *D*) from *C*1 to *C*3, component *C*2 and *C*3 will become *Failed* and *Standby*, respectively. Since component *C*4 is a spare unit of component *C*2, the failure of component *C*2 will lead to the activation of component *C*4; however, since there is a (*D*, *D*) state dependency from component *C*3 to *C*4, the deactivation of component *C*3 will lead to the deactivation of component *C*4. To evaluate the system's reliability, the following question has to be answered: when component *C*1 fails, will *C*4 be in an *Active* state or a *Standby* state? Or will the result be nondeterministic? The above question can actually be reduced to the following question: when component *C*1 fails, will the two state dependencies (from *C*1 to *C*2, and from *C*1 to *C*3) happen immediately and thus simultaneously, or with some random time delay? To answer this type of questions, it is vital for us to formally define the denotational semantic of the DRBD constructs.

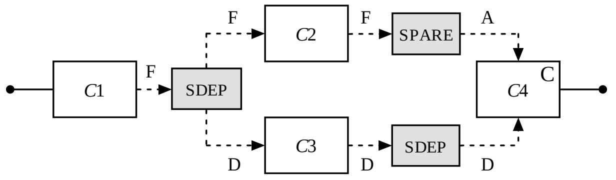


Fig. 6. A DRBD Model with Conflicts

There is only little work done on formal specification of reliability modeling constructs. Coppit *et al.* once used the Z formalism to provide formal and precise definitions for various dynamic fault trees (DFT) gates [7], which were proposed to extend the traditional fault tree analysis for modeling some dynamic behaviors [13, 14]. However, in their approach, only state schemas are defined, while the operation schemas for modeling dynamic behaviors of gates are missing. Furthermore, no solutions have been provided for the verification of DFT models to ensure a correct design. In this section, we propose to use the Object-Z formalism [15] to specify both the state space and operations of a DRBD construct as a class schema. Furthermore, in Section 5, we propose a colored Petri net based method to the formal verification and validation of DRBD models.

Object-Z is an extension to the Z formal specification language for modular design of complex systems [15, 16]. It has strong data and state modeling capabilities, and thus is suitable for specifying the formal semantics of DRBD modeling constructs. For illustration purpose, we show the formal specification of SDEP and SPARE blocks in Object-Z in Figures 7 and 8, respectively.

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

$SDEP$

$$\begin{array}{l}
trigger : Component \\
targets : \mathbb{P}\, Component \\
nTargets : \mathbb{N} \\
triggerEvent : Event \\
targetEvents : Component \rightarrow Event \\
sdep : \mathbb{T} \times Component \times Event \rightarrow \mathbb{P}(\mathbb{T} \times Component \times Event) \\
\hline
nTargets = \#targets \wedge nTargets > 0 \wedge targets = \mathrm{dom}\, targetEvents \\
\forall\, c \in targets \bullet c \neq trigger \\
\quad \wedge probability(c \mid triggerEvent) \neq probability(c) \\
\quad \wedge\ probability(triggerEvent \mid c) = probability(triggerEvent) \\
\{(t, trigger, triggerEvent) \mid t \in \mathbb{T}\} = \mathrm{dom}\, sdep
\end{array}$$

$ActivateTrigger$

$$\begin{array}{l}
\Delta(trigger, targets) \\
t? : \mathbb{T} \\
\hline
(triggerEvent = Active) \wedge (trigger.state' = Active) \\
\forall\, c \in targets \bullet \\
\quad (t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent) \\
\quad \wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active) \\
\quad \vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby) \\
\quad \vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))
\end{array}$$

$DeactivateTrigger$

$$\begin{array}{l}
\Delta(trigger, targets) \\
t? : \mathbb{T} \\
\hline
(triggerEvent = Deactivation) \wedge (trigger.state' = Standby) \\
\forall\, c \in targets \bullet \\
\quad (t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent) \\
\quad \wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active) \\
\quad \vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby) \\
\quad \vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))
\end{array}$$

$FailTrigger$

$$\begin{array}{l}
\Delta(trigger, targets) \\
t? : \mathbb{T} \\
\hline
(triggerEvent = Failure) \wedge (trigger.state' = Failed) \\
\forall\, c \in targets \bullet \\
\quad (t? + \delta_c, c, targetEvents(c)) \in sdep(t?, trigger, triggerEvent) \\
\quad \wedge\ ((targetEvents(c) = Activation \wedge c.state' = Active) \\
\quad \vee\ (targetEvents(c) = Deactivation \wedge c.state' = Standby) \\
\quad \vee\ (targetEvents(c) = Failure \wedge c.state' = Failed))
\end{array}$$

Fig. 7. Object-Z Specification of SDEP Block

For formally defining the state dependence (SDEP) block, we first define *Event* occurring on a component as an enumerated type with the following values: *Activation*, *Deactivation*, and *Failure*, which will lead a component to a state of *Active*, *Standby* and *Failed*, respectively (Figure 7). Then we define a state dependency as a block that consists of a trigger component and a set of target/dependent components. The relationship between the trigger component and the target components are defined by a function *sdep*, which maps the trigger event happening at time *t*? (a standard notation in Object-Z for input variable *t*) to a set of target events happening at *t*?+δ_c, where *c* represents a target component. This definition precisely states that the target events do not need to happen simultaneously as long as δ_c is different for different target components. The three operations defined on a state dependency are *ActivateTrigger*, *DeactivateTrigger*, and *FailTrigger*. According to the definition, the activation of the trigger component may lead to one of three different states for each target component, i.e., *Active*, *Standby* and *Failed*, which correspond to three different relationships between the trigger and one of the targets, namely (*A*, *A*), (*A*, *D*), and (*A*, *F*) state dependency. Similarly, the formal definition of *DeactivateTrigger* operation specifies how state dependencies (*D*, *A*), (*D*, *D*), and (*D*, *F*) are enforced; while the formal definition of *FailTrigger* specifies how state dependencies (*F*, *A*), (*F*, *D*), and (*F*, *F*) are enforced.

For formally defining the SPARE block in Object-Z, we define a spare part redundancy as a block that consists of a primary unit component and a set of alternative (spare) unit components (Figure 8). The relationships among them are defined by the function *switch*, which maps a state change event happened on a primary unit or an alternative unit at time *t*? to a state change event happened on an alternative unit at *t*?+δ_c, where *c* represents an alternative unit component. The *PrimarySwitch* operation defines when the primary unit is deactivated or failed, the first alternative unit will be activated; while the *AlternativeSwitch* operation defines when alternative unit *i*? is deactivated or failed, alternative unit *i*?+1 will be activated. Note that the previous state of an activated alternative unit must be *Standby*, which can be in one of the three cases: *Hot*, *Cold* and *Warm*.

Formal verification is used to make sure the model constructed is correct, and more specifically, the model is an accurate representation of an actual system in terms of its reliability properties. Traditional simulation approaches to model testing is not suitable for verifying the DRBD models because it is hard (actually almost impossible) to cover all execution paths. A feasible way is to use formal methods to verify whether the DRBD model satisfies the specified behavioral properties of the system under investigation. In our work, the formal model of Petri nets is adopted because it is supported by powerful verification tools and it has a distinct advantage of being easy to understand and use [19]. Petri nets provide an intuitive and graphical way to express conditions, events, and their relationships, as well as essential characteristics like nondeterminism and concurrency. Refer to [8] for an example showing how we converted a DRBD model into a colored Petri net and then used an existing Petri net tool called CPN Tools [20] to analyze the resulting Petri net model for detecting possible errors in the DRBD model. In general, some error exists in the DRBD model if the Petri net analysis result shows that some specified behavioral property is not satisfied. For example, a dead marking in the reachability graph of the Petri net model implies some design error existing in the model. Using the CPN tool, the dead marking state can be traced to find the firing sequence that leads to the dead marking. The trace file thus helps designers to detect the errors and improve the design. If the Petri net analysis result indicates that all specified properties are satisfied, then the corresponding DRBD model is guaranteed to be correct. The evaluation of the DRBD model may then start for finding the reliability of the designed system.

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

$SPARE$

$$\begin{array}{l}
primaryUnit : Component \\
alternativeUnits : \mathbb{P}\, Component \\
nAlternatives : \mathbb{N} \\
alternative : \mathbb{N} \rightarrow Component \\
switch : \mathbb{T} \times Component \times Event \rightarrow \mathbb{T} \times Component \times Event \\
\hline
\forall\, c \in alternativeUnits \bullet c \neq primaryUnit \\
nAlternatives = \# alternativeUnits \wedge nAlternatives > 0 \\
\{1, 2, ..., nAlternatives\} = \operatorname{dom} alternative \\
\operatorname{dom} switch = \{(t, c, e) \mid t \in \mathbb{T}, c \in \{primaryUnit\} \cup \\
\quad \{(alternative(i)) \mid 1 \leq i \leq nAlternatives - 1\}, \\
\quad e \in \{Deactivation, Failure\}\}
\end{array}$$

$\textsc{Init}$

$$\begin{array}{l}
trigger.state = Active \\
\forall\, i, where\ 1 \leq i \leq nAlternatives \bullet alternative(i).state = Standby
\end{array}$$

$PrimarySwitch$

$$\begin{array}{l}
\Delta(primaryUnit, alternativeUnits) \\
t? : \mathbb{T} \\
\hline
\forall\, e \in \{Deactivation, Failure\} \bullet \\
\quad switch(t?, primaryUnit, e) = (t? + \delta_c, alternative(1), Activation) \\
(primaryUnit.state = Active) \wedge \\
\quad (primaryUnit.state' \in \{Standby, Failed\}) \\
\quad \wedge\ (alternative(1).state' = Active)
\end{array}$$

$AlternativeSwitch$

$$\begin{array}{l}
\Delta(alternativeUnits) \\
t? : \mathbb{T}, i? : \mathbb{N} \\
\hline
\forall\, e \in \{Deactivation, Failure\}, 1 \leq i? \leq nAlternatives - 1 \bullet \\
\quad switch(t?, alternative(i), e) = \\
\quad (t? + \delta_c, alternative(i + 1), Activation) \\
\quad \wedge\ (alternative(i?).state = Active) \\
\quad \wedge\ (alternative(i?).state' \in \{Standby, Failed\}) \\
\quad \wedge\ (alternative(i? + 1).state = Standby) \\
\quad \wedge\ (alternative(i? + 1).state' = Active)
\end{array}$$

Fig. 8. Object-Z Specification of SPARE Block

5. DRBD Model Evaluation

To obtain a prediction of the system reliability with the consideration of various dependent and dynamic behaviors described in Section 3, we propose an integrated approach that adapts the modular approach to dynamic fault trees analysis [5, 21] for the DRBD model evaluation. The approach first modularizes the entire system DRBD model into multiple independent modules. Each module is a sub-DRBD whose blocks do not occur elsewhere in the entire DRBD model. A module involving dependencies and dynamics (i.e., proposed new DRBD blocks) will be identified as a dynamic module; otherwise (i.e., containing only traditional RBD blocks), it is identified as a static module. The modularization allows the use of Markov models for dynamic parts of the system that require them and the use of combinatorial methods for static parts of the system to retain the efficiency of combinatorial solutions wherever possible.

There are two main concepts in the Markov chain solution to dynamic DRBD modules: module states and state transitions. The state of a dynamic module generally represents a distinct combination of states of components. The state transitions govern the changes of a state that occur within the module. As time passes and failures occur, the module goes from one state to another in a recursive fashion, until absorbing states, i.e. the module failure states are reached. Solving a Markov model consists of solving a set of differential equations: $\mathbf{P}'(t)=\mathbf{A}\mathbf{P}(t)$, where $\mathbf{P}$ is the state probability vector: $\mathbf{P}(t)=[P_1(t), P_2(t), \ldots, P_n(t)]^T$. $P_i(t)$ is the probability of the module being in state i at time t, and $\mathbf{A}$ is an $n \times n$ transition rate matrix with n being the number of states that are presented in the Markov model. Solving the above set of differential equations involves the use of the condition of $\sum_{i=1}^{n} P_i(t) = 1$ [2]. The solution includes the probability of the module being in each state. The unreliability of the module can be obtained by adding the occurrence probability of each failure state. The Markov model is a powerful solution method since it can deal with dynamic and dependent behaviors easily. However, the Markov-based approach has a major disadvantage that its size grows exponentially as the size of the system/module increases, which may lead to intractable models. This weakness can be alleviated by integrating the Markov model with combinatorial solutions that are used wherever possible in the evaluation of DRBD models.

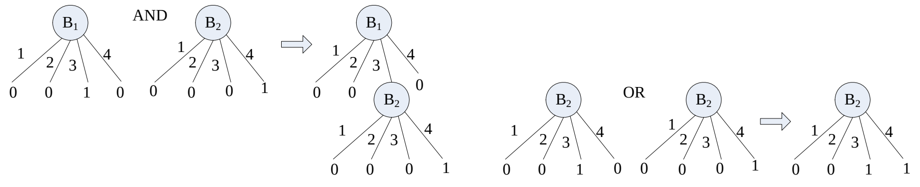
It has been shown that the combinatorial methods based on binary decision diagrams (BDD) [7] can provide a faster and more efficient analysis of large static binary-state systems than other traditional methods, such as cut sets and path sets. Therefore, the modular approach to the DFT analysis adopts the BDD as the solution to the analysis of static modules. However, the major problem with the BDD based methods is that they can only efficiently model and analyze systems with binary state components. For analyzing systems with multistate components, a large number of Boolean variables (one corresponding to each state of the component) as well as state dependencies among these variables must be dealt with, which greatly reduces the efficiency of the solution [11, 12]. Recently, an algorithm based on multi-state multi-valued decision diagrams (MMDD) has been proposed to the multistate system analysis [12, 22]. An MMDD is a logical extension of BDD. Each MMDD is a direct acyclic graph with two and only two sink nodes, representing the system being/not being in a specific state, respectively. Each non-sink node, representing a multistate component, is associated with a multi-valued state variable and has multiple outgoing edges; one corresponding to each state of the component. Our empirical comparison results in [22] have shown that the MMDD based method is more computationally efficient than the BDD-based method. Therefore, we adopt the MMDD method in our modular approach for DRBD evaluation. The BDD method will be used when only binary-state components are involved.

The MMDD-based DRBD analysis can be viewed as a two-step process: conversion from DRBD to MMDD model followed by the evaluation of the model. The conversion process relies on the following equation for manipulating two logic expressions in the MMDD generation.

$$
\begin{aligned}
G \lozenge H &= case(x, G_1, \ldots G_n) \lozenge case(y, H_1, \ldots, H_n) \\
&= \begin{cases} case(x, G_1 \lozenge H_1, \ldots G_n \lozenge H_n) & index(x) = index(y) \\ case(x, G_1 \lozenge H, \ldots G_n \lozenge H) & index(x) < index(y) \\ case(y, G \lozenge H_1, \ldots G \lozenge H_n) & index(x) > index(y) \end{cases}
\end{aligned}
\tag{1}
$$

where $case(x,G_1, \ldots, G_n)$ and $case(y,H_1, \ldots, H_n)$ are the *case* format for logic expressions G and H, representing the two sub-MMDDs, respectively [12]. The *index* represents the order of the variable. The symbol $\lozenge$ represents any logic operation (AND or OR). Specifically, a logic AND operation will be applied when a series structure is encountered in the multistate DRBD module; a logic OR operation will be applied for a parallel structure. Note that this is different from the binary systems where the components have only two states (operation and failure), and a logic OR operation is applied for a series structure and a logic AND operation is applied for a parallel structure.

To illustrate the conversion from DRBD to MMDD, we consider the example multistate DRBD in Figure 5(b). Since each board in the computer system has four states, it is represented by a node in MMDD with four outgoing edges, one corresponding to each state of the component. Assume the order of $B_1 < B_2$ is used for the MMDD generation. Because $B_{1,3}$ and $B_{2,4}$ are in a series structure in the DRBD module, the case 2 in Equation (1) and an AND operation are applied, as shown in Figure 9(a). $B_{2,3}$ and $B_{2,4}$ are in a parallel structure, hence the case 1 in Equation (1) and an OR operation are applied, as shown in Figure 9(b). This parallel structure and $B_{1,4}$ forms another series structure, thus the case 3 in Equation (1) and an AND operation are applied between the MMDD generated in Figure 9(b) and the MMDD for the basic event $B_{1,4}$, as illustrated in Figure 9(c). Finally, the MMDDs in Figure 9(a) and (c) are ORed together to generate the MMDD for the entire DRBD model shown in Figure 9(d).


(a) For series structure of $B_{1,3}$ and $B_{2,4}$ (b) For parallel structure of $B_{2,3}$ and $B_{2,4}$

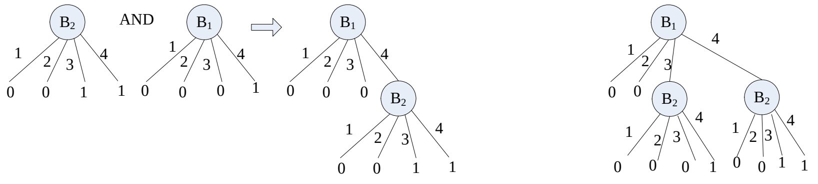

(c) For series structure in the bottom (d) For entire multistate DRBD

Fig. 9. MMDD Generation from DRBD

After generating the MMDD from a DRBD model, the probability of system being in the specific state can be given by the sum of the probabilities for all the paths from the root to a sink node labeled "1" [12].

For illustrating the modular approach described above, we consider the evaluation of the DRBD model in Figure 4 for computing the reliability of the HECS system. First, the entire system DRBD model is modularized into four independent modules: two static modules (the bus module and the I/O module) and two dynamic modules (the processor module and the memory module). The two dynamic modules are solved using Markov chains. For illustration purpose, the Markov chain model of the processor module in Figure 4 is shown in Figure 10(a). Each system state is represented by a three-tuple (S_1, S_2, S_S), where S_1, S_2, and S_S represent the state of processors P_1, P_2, and P_S, respectively. And each processor can assume three mutually exclusive states: active (A), standby (S), and failed (F). The state (F,F,F) in Figure 10(a) represents the failure of the processor module and is the absorbing state of the Markov model.

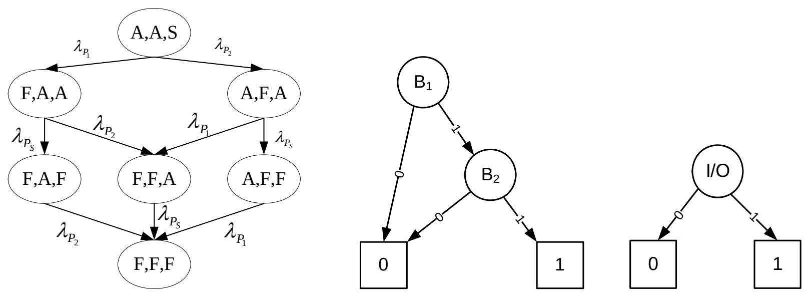

(a) Markov Chain of Processor (b) BDD of Bus (c) BDD of I/O

Fig. 10. Evaluation of DRBD Modules for the HECS

The Markov model of the memory module in Figure 4, which has a total of 87 states and 190 transitions, is not shown due to the space limitation. The two static modules are solved using the combinatorial MDD method. Since each component in the static modules exhibits only binary states, the BDD is used to solve the static modules. Figures 10(b) and (c) show the BDD model of the bus module and the I/O module, respectively. After solving these four modules, their results are integrated to obtain the entire system unreliability of 5.05178e-3 for the mission time of 1000 hours, which corresponds to the reliability of 0.99494822 for the HECS system.

6. Conclusions and Future Work

In this paper, we presented a new framework based on the extension of traditional RBD model for the reliability analysis of complex computer based systems. The framework includes (1) new DRBD models for representing various dependent and dynamic behaviors in an intuitive way; (2) formal specification of these new models using Object-Z to provide precise semantics and help eliminate ambiguity in development of DRBD models; (3) formal verification using colored Petri nets to ensure the correctness of DRBD models; and (4) a modular approach that integrates the efficient MMDD method and Markov solution for DRBD model evaluation. The DRBD analysis will provide a new and powerful tool for modeling, verification, and evaluation of large and complex system reliability.

In addition to the previously mentioned investigation into more case studies with dynamic features not detailed/addressed in this paper, our next research tasks include designing conversion algorithms to support automatic translation of DRBD models to colored Petri nets, designing modularization algorithms for automatically detecting DRBD modules, and development of an easy-to-use computer software tool to implement the entire framework.

References

[1] W. Wang, J. M. Loman, R. G. Arno, P. Vassiliou, E. R. Furlong, and D. Ogden, Reliability block diagram simulation techniques applied to the IEEE std. 493 standard network, IEEE Transactions on Industry Applications (**2004**) Vol. 40, No. 3, pp. 887-895.

[2] M. Rausand and A. Høyland, System reliability theory: models, statistical methods, and applications, New York, USA, Wiley-Interscience (**2003**).

[3] S. Distefano and L. Xing, A new approach to modeling the system reliability: dynamic reliability block diagrams, Proceedings of the 52nd Annual Reliability and Maintainability Symposium, (**2006**) January; Newport Beach, CA, pp. 189-195.

[4] "Reliasoft Corporation," URL: http://www.reliasoft.com.

[5] J. B. Dugan, B. Venkataraman, and R. Gulati, DIFtree: A software package for the analysis of dynamic fault tree models, Proceedings of the Annual Reliability and Maintainability Symposium, (**1997**) January; pp. 64-70.

[6] R. Manian, J. B. Dugan, D. Coppit, and K. Sullivan, Combining various solution techniques for dynamic fault tree analysis of computer systems, Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium, (**1998**) November; Washington, D.C., pp. 21-28.

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

[8] H. Xu and L. Xing, Formal Semantics and Verification of Dynamic Reliability Block Diagrams for System Reliability Modeling, Proceedings of the 11th International Conference on Software Engineering and Applications, (**2007**) November; Cambridge, Massachusetts, USA (to appear).

[9] B. W. Johnson, Design and analysis of fault tolerant digital systems, Boston, USA, Addison-Wesley Longman Publishing Co. Inc. (**1989**).

[10] E. J. Henley and H. Kumamoto, Probabilistic Risk Assessment: Reliability Engineering, Design, and Analysis, IEEE Press (**1992**).

[11] X. Zang, D. Wang, H. Sun, and K. S. Trivedi, A BDD-Based Algorithm for Analysis of Multistate Systems with Multistate Components, IEEE Transactions on Computers, (**2003**) Vol. 52, No. 12, pp. 1608-1618.

[12] 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.

[13] D. Coppit and K. J. Sullivan, Formal specification in collaborative design of critical software tools, Proceedings of the Third IEEE International High-Assurance Systems Engineering Symposium, (**1998**) November; Washington, D.C., pp. 13-20.

[14] D. Coppit, K. J. Sullivan, and J. B. Dugan, Formal semantics of models for computational engineering: a case study on dynamic fault trees, Proceedings of the International Symposium on Software Reliability Engineering, (**2000**); San Jose, California, pp. 270-282.

[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] “Clockwork Solutions Inc.” URL: http://www.clockwork-solutions.com.

[18] “Isograph Inc.” URL: http://www.isograph-software.com.

[19] T. Murata, Petri nets: properties, analysis and applications, Proceedings of the IEEE, (**1989**) Vol. 77, No. 4, pp. 541-580.

[20] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen and K. Jensen, CPN Tools for editing, simulating, and analyzing colored Petri nets, Proceedings of the 24th International Conference on the Application and Theory of Petri Nets, (**2003**) June; Eindhoven, The Netherlands.

[21] R. Gulati and J. B. Dugan, A Modular Approach for Analyzing Static and Dynamic Fault Trees, Proceedings of the Annual Reliability and Maintainability Symposium, (**1997**) January; Philadelphia, PA, pp. 568-573.

[22] L. Xing and Y. Dai, A new decision diagram based method for efficient analysis on multi-state systems, IEEE Transactions on Dependable and Secure Computing, 2007 (under the third revision)

[23] "Relex Software Corporation," URL: http://www.relex.com.

[24] S. V. Amari, K. B. Misra, H. Pham, "Tampered Failure Rate Load-Sharing Systems: Status and Perspectives," Chapter 20 in Handbook on Performability Engineering (Editor: K. B. Misra), Springer (**2007**).