DRBD: Dynamic Reliability Block Diagram for System Reliability Modeling

Prof. Haiping Xu
Concurrent Software Systems Laboratory
Computer and Information Science Department
University of Massachusetts Dartmouth

Acknowledgement

- **Dr. Liudong Xing**, Assistant Professor
  Electrical and Computer Engineering Department
  University of Massachusetts Dartmouth

- **Ryan Robidoux**, Graduate Student
  Concurrent Software Systems Laboratory
  Computer and Information Science Department
  University of Massachusetts Dartmouth
Outline

- DRBD controller component blocks
- Development of DRBD models (example)
- Formal specifications of DRBD constructs
- Formal verification of DRBD models
- Conversion of DRBD models into colored Petri nets (CPN)
- Case study: modeling, verification
- Conclusions and future work

A Motivating Example

Initially, sensor nodes in S1 are operational; sensor nodes in S2 are in a sleeping mode.

- When the primary cluster head fails, the secondary cluster head will be automatically activated.
- Sensor nodes in S1 can be put into a sleeping mode, and sensor nodes in S2 will be activated.
- How to model the state dependency between S1 and S2: Deactivation -> Activation dependency?
The State of the Art

- Most of the existing reliability modeling tools (e.g., RBD) cannot capture the state dependency between components.
- Other tools, such as Dynamic Fault Tree (DFT), may support modeling a functional dependency
  - The failure of a component causes some other dependent components to become inaccessible or unusable
  - However, it still cannot capture the Deactivation -> Activation state dependency between components.
- We propose a set of new Dynamic Reliability Block Diagram (DRBD) constructs as an extension to the existing RBD modeling tool.

DRBD Controller Component Blocks

- A stands for an activation event occurred on a component that leads to an Active state of that component,
- D stands for a deactivation event occurred on a component that leads to a Standby state of that component, and
- F stands for a failure event occurred on a component that leads to a Failed state of that component.
The failure of the primary cluster head will automatically activate the secondary cluster head.
The components labeled S1 and S2 represent the two sets of sensor nodes that may work alternatively.
The deactivation of S1 (S2) will automatically activate S2 (S1).

To support formal verification and validation of our proposed DRBD model, it is necessary to formally define the DRBD modeling constructs.

Provide the denotational semantics for the development of DRBD models in a precise manner.
Help to eliminate ambiguity in a constructed DRBD model.

Question 1: When component C1 fails, will C4 be in a state of Active or Standby, or will the result be nondeterministic?
Object-Z Specification

**Event** :: Activation | Deactivation | Failure

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Composition</td>
</tr>
<tr>
<td>Target</td>
<td>Component</td>
</tr>
<tr>
<td>$c$</td>
<td>Targets</td>
</tr>
<tr>
<td>$\text{targetEvent}$</td>
<td>Event</td>
</tr>
<tr>
<td>$\text{triggerEvent}$</td>
<td>Event</td>
</tr>
<tr>
<td>$\text{targetEvents}$</td>
<td>Component</td>
</tr>
<tr>
<td>$\text{set}sdep$</td>
<td>Target x Component x Event</td>
</tr>
</tbody>
</table>

$nTargets = \# targets \land nTargets > 0 \land targets = \text{dom targetEvents}$

\(\forall c \in \text{targets} \cdot c \neq \text{trigger} \land \text{probability}(c | \text{triggerEvent}) \neq \text{probability}(c) \lor \text{probability}(\text{triggerEvent} | c) \neq \text{probability}(\text{triggerEvent})

\[(t, \text{trigger}, \text{triggerEvent}) \in \text{dom sdep}]

$\text{activateTrigger}$

\(\Delta(\text{trigger}, \text{targets})

\begin{align*}
\forall c \in \text{targets} \cdot & (t + \delta_c, c, \text{targetEvents}(c)) \in \text{sdep} \land (t, \text{trigger}, \text{triggerEvent}) \in \text{dom sdep} \land \\
& \text{targetEvents}(c) = \text{Activation} \land c.\text{state}' = \text{Active} \\
& \text{or} \text{targetEvents}(c) = \text{Deactivation} \land c.\text{state}' = \text{Standby} \\
& \text{or} \text{targetEvents}(c) = \text{Failure} \land c.\text{state}' = \text{Failed}
\end{align*}

$\text{deactivateTrigger}$

$\text{failTrigger}$

Formal Verification Approach

- Testing or simulations are not suitable for verifying DRBD models because it is almost impossible to cover all cases.
- Use formal methods (e.g., model checking techniques) to verify the behavioral properties of a DRBD model before the evaluation process starts.
- Use temporal logic to specify system properties:
  - **Property P:** "If component A fails, component B and C will also fail, which leads to the failure of the whole system S." The temporal formula in LTL (Linear Temporal Logic) can be written as:
    \[ \neg(-A \Rightarrow (-B \lor -C)) \land \neg S \Rightarrow S \]
- When a DRBD model is proved to be incorrect:
  - Any quantitative evaluation results might be unusable.
  - The DRBD model needs to be fixed.
Formal Verification Models

- DRBD models are not formally defined & executable.
- Object-Z specifications of DRBD constructs are formal specifications, however
  - Are not feasible for verification of behavioral properties.
  - Have no effective analysis and verification tool support.
- Convert a DRBD model into a formal executable model such as a state machine or a Petri net model.
- We adopt Colored Petri Net (CPN) model because
  - Is user friendly based on its graphical notations.
  - Has powerful, but intuitive rules for defining structure and dynamic behaviors.
  - Has many existing analysis and verification tools.

Introduction to Petri Net

- “Three-in-one” capability of Petri net models [Murata 1989]
  - Graphical representation
  - Mathematical description
  - Simulation tool

Definition:

A Petri net is a 4-tuple, PN = (P, T, F, M₀) where

\[ P = \{P_1, P_2, \ldots, P_m\} \] is a finite set of places;

\[ T = \{t_1, t_2, \ldots, t_n\} \] is a finite set of transitions;

\[ F \subseteq (P \times T) \cup (T \times P) \] is a set of arcs (flow relation);

\[ M_0: P \rightarrow \{0, 1, 2, 3, \ldots\} \] is the initial marking.
In an ordinary Petri net, tokens are all of color **black**.

In a Colored Petri net (CPN or CP-net),
- Colors of tokens can represent values.
- A transition may have a guard and executable code.

---

**Convert DBBD into CPN Models**

- Define three different colors/states: **Active**, **Standby** and **Failed**.
- A transition is associated with a guard and executable code
  - Can fire **only if** the guard \([x=\text{Failed}, y=\text{Active}]\) evaluates to true.
  - Code `output(z); action(\text{Standby})` deposits a **Standby** token in \(C2\).
A Case Study

- Router R1 is connected to two server computers C1 and C2.
- Server computers C1 and C2 are load sharing servers.
- When router R1 fails, the computers C1 and C2 will be deactivated.
- To make the system more reliable, we introduce a cold spare (CSP) for router R1, which is represented by component R2.

Colored Petri Net Model

- Colset UNIT = unit with e;
- Colset BOOL = bool;
- Colset STATE = with Active | Standby | Failed;
- Var x, y, z : STATE;
- Var u : UNIT;
- Var b : BOOL;

11/19/2007 CIS Dept., UMass Dartmouth
### Analysis Results

<table>
<thead>
<tr>
<th>Result-1</th>
<th>Result-2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statistics</strong></td>
<td>DeadMarking(32)</td>
</tr>
<tr>
<td>State Space</td>
<td>val it = true : bool</td>
</tr>
<tr>
<td>Nodes: 33</td>
<td>print(NodeDescriptor 32)</td>
</tr>
<tr>
<td>Arcs: 69</td>
<td></td>
</tr>
<tr>
<td>Secs: 0</td>
<td></td>
</tr>
<tr>
<td>Scc Graph</td>
<td></td>
</tr>
<tr>
<td>Nodes: 33</td>
<td></td>
</tr>
<tr>
<td>Arcs: 62</td>
<td></td>
</tr>
<tr>
<td>Secs: 0</td>
<td></td>
</tr>
<tr>
<td><strong>Live Markings</strong></td>
<td></td>
</tr>
<tr>
<td>State Space</td>
<td>val it = () : unit</td>
</tr>
<tr>
<td>Nodes: 33</td>
<td>Reachable'(1, 32)</td>
</tr>
<tr>
<td>Arcs: 69</td>
<td></td>
</tr>
<tr>
<td>Secs: 0</td>
<td></td>
</tr>
<tr>
<td><strong>States</strong></td>
<td></td>
</tr>
<tr>
<td>State Space</td>
<td></td>
</tr>
<tr>
<td>Nodes: 33</td>
<td></td>
</tr>
<tr>
<td>Arcs: 62</td>
<td></td>
</tr>
<tr>
<td>Secs: 0</td>
<td></td>
</tr>
</tbody>
</table>

### Deadlock in CPN

```
colset UNIT      = unit with e;
colset BOOL     = bool;
colset STATE    = with Active | Standby | Failed;
var x, y, z         : STATE;
var u                : UNIT;
var b                : BOOL;
T7 [x=Failed orelse y=Failed]
output (b);
action (true);
T6 [x=Active, y=Active, z=Active]
output (b);
action (true);
T5 [x=Active, y=Active, z=Active]
output (b);
action (true);
T1 [x=Active]
T2 [x=Active]
LSH_2 [x=Failed, y=Active]
output (z);
action (Standby);
LSH_1 [x=Failed, y=Active]
output (z);
action (Standby);
SDEP_R1_C2 [x=Failed, y=Active]
output (z);
action (Standby);
SDEP_R2_C1 [x=Failed, y=Active]
output (z);
action (Standby);
SDEP_R2_C2 [x=Failed, y=Active]
output (z);
action (Standby);
Fail_C2 [x=Active]
output (y);
action (Failed);
Stop [b=true]
Run [b=true]
SDEP_R1_C1 [x=Failed, y=Standby]
output (z);
action (Active);
Spare [x=Failed, y=Standby]
output (z);
action (Active);
Syn_2 [UNIT]|
Syn_1 [UNIT]|
R1_or_R2 [STATE]
C2 1`Active
R1 1`Active
R2 1`Standby
C1 1`Standby
R1 1`empty
R2 1`empty
R1_or_R2 1`Active
Syn_1 1`empty
Syn_2 1`empty
System_down 1`empty
System_up 1`empty
val it = true : bool
```
Analysis Results (after revision)

**Result-3**

<table>
<thead>
<tr>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Space</td>
</tr>
<tr>
<td>Nodes: 67</td>
</tr>
<tr>
<td>Arcs: 162</td>
</tr>
<tr>
<td>Secs: 0</td>
</tr>
<tr>
<td>Status: Full</td>
</tr>
<tr>
<td>Scc Graph</td>
</tr>
<tr>
<td>Nodes: 67</td>
</tr>
<tr>
<td>Arcs: 141</td>
</tr>
<tr>
<td>Secs: 0</td>
</tr>
<tr>
<td>Liveness Properties</td>
</tr>
<tr>
<td>Dead Markings: None</td>
</tr>
<tr>
<td>Dead Transition Instances: None</td>
</tr>
<tr>
<td>Live Transition Instances: None</td>
</tr>
</tbody>
</table>

- Fix the colored Petri net model by adding:
  - New transition \( SDEP_{R2}C12 \)
  - New synchronization place \( Syn_3 \)
  - And arcs and guards
- The analysis results show no deadlock markings.
- **Question 3**: How to verify additional properties?
Model Checking Results

<table>
<thead>
<tr>
<th>Formulas</th>
<th>ASK-CTL in ML</th>
<th>After Rev</th>
<th>Before Rev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula_1</td>
<td>val myASKCTLformula = EXIST_UNTIL(TT, NOT(MODAL(TT))); eval_node myASKCTLformula InitNode;</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Functions</td>
<td>fun R1_Failed n = (Mark.R1 1 n = 1'Failed); fun R2_Failed n = (Mark.R2 1 n = 1'Failed); fun SystemFailed n = (Mark.System_down 1 n = 1'true);</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Formula_2</td>
<td>val isFailed = FORALL_UNTIL(TT, NF(&quot;,SystemFailed)); val system = OR(NOT(NF(&quot;, R1_Failed)), isFailed); val myASKCTLformula = INV(system); eval_node myASKCTLformula InitNode</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Formula_3</td>
<td>val isFailed = FORALL_UNTIL(TT, NF(&quot;,SystemFailed)); val system = OR(NOT(NF(&quot;, R2_Failed)), isFailed); val myASKCTLformula = INV(system); eval_node myASKCTLformula InitNode</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

Conclusions and Future Work

- Proposed a new modeling approach called Dynamic Reliability Block Diagrams (DRBD)
  - Resolves the shortcomings of the existing work.
  - Provides a powerful but easy-to-use reliability modeling tool for complex and large computer-based systems.
  - Supports automated verification of DRBD models.
- Develop a software tool that can automatically translate DRBD models into colored Petri nets.
- Study efficient evaluation methods for DRBD models.
- Develop a comprehensive system reliability modeling tool that supports editing, formal verification, and evaluation of DRBD models.
Related Publications


Contact Information

Haiping Xu, Assistant Professor
Computer and Information Science (CIS) Department, College of Engineering
University of Massachusetts Dartmouth
Phone: (508) 910-6427
Email: hxu@umassd.edu

Liudong Xing, Assistant Professor
Electrical and Computer Engineering (ECE) Department, College of Engineering
University of Massachusetts Dartmouth
Phone: (508) 999-8883
Email: lxing@umassd.edu

Questions?

The slides for this talk can be downloaded from
http://www.cis.umassd.edu/~hxu