Extending Reliability Block Diagrams to Software Architectures

Ahmed A. Abd-Allah
aabdalla@sunset.usc.edu

March 11, 1997

USC Center for Software Engineering

Annual Research Review
Finding an Appropriate Starting Point

- Many ways to correlate Software Architecture and Software Reliability

<table>
<thead>
<tr>
<th>Software Architecture</th>
<th>Software Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views of Components &amp; Connectors</td>
<td>Reliability Block Diagrams</td>
</tr>
<tr>
<td>Architectural Styles</td>
<td>Fault Tolerance</td>
</tr>
<tr>
<td>Heterogeneous Composition</td>
<td>Reliability Estimation Models</td>
</tr>
<tr>
<td>Architectural Description Languages</td>
<td>Reliability via Complexity Measures</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Question: Are Reliability Block Diagrams suited for Software Architectural reliability analyses?
  - provide sensitivity analysis
  - useful for tradeoff analysis against other nonfunctional properties (e.g. performance, cost)
  - stepping stone to fault tolerance techniques (N-version programming, recovery blocks, ...)

Extending Reliability Block Diagrams to Software Architectures
Tradeoff example: Software Cost/Reliability (Boehm)

- Use COCOMO required reliability (RELY) ratings to estimate both:
  - module’s effort multiplier
  - module’s operational reliability

<table>
<thead>
<tr>
<th>RELY rating</th>
<th>Effort Multiplier</th>
<th>Reliability (per hour)</th>
<th>MTBF (hours)</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very High</td>
<td>1.40</td>
<td>0.99999999</td>
<td>$10^7$</td>
<td>FAA standard for life-critical systems</td>
</tr>
<tr>
<td>High</td>
<td>1.15</td>
<td>0.99999</td>
<td>$10^5$</td>
<td></td>
</tr>
<tr>
<td>Nominal</td>
<td>1.0</td>
<td>0.999</td>
<td>$10^3$</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>0.88</td>
<td>0.9</td>
<td>10</td>
<td>Typical performance of market-driven COTS</td>
</tr>
<tr>
<td>Very Low</td>
<td>0.75</td>
<td>0.5</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Reliability Block Diagrams (RBDs): Boxes & Lines

- Boxes are computational blocks with reliabilities:
  - probability of failure-free operation over a given amount of time (e.g. 97% over an 24 hour period)

- Connectors are... connectors
  - data or control
  - assumed to be perfectly reliable

- Common configurations and resulting reliabilities:

\[
R = \prod_{i=1}^{n} R_i
\]

\[
R = 1 - \prod_{i=1}^{n} (1 - R_i)
\]
Failure Rate Calculation

- Reliability $R_i$ often derived from effective failure rate $\lambda_i$ (failures per unit time)

\[ R = e^{-\lambda t} \]

- Four important parameters for $\lambda$:

  \[
  f = \text{measured Failure rate (e.g. failures/hour)}
  \\
  t = \text{relative Time spent in block}
  \\
  u = \text{Utilization of underlying platform}
  \\
  s = \text{relative Speed of underlying platform}
  \\
  \lambda = f \cdot t \cdot u \cdot s
  \]

- These parameters can be affected by software architectural extensions to RBDs
Software Architecture: Styles

- MAIN/SUBROUTINE
- PIPE & FILTER
- LAYERED
- SOFTWARE BUS
- RULE-BASED
- EVENT-BASED
- MULTI-THREADED
- FEEDBACK SYSTEMS
- BLACKBOARD
- LOGIC COMPUTING
- DISTRIBUTED PROCESSES
- DATABASE
- REAL-TIME SYSTEMS
- OBJECT-ORIENTED

<table>
<thead>
<tr>
<th>Vendor</th>
<th>COTS* Package</th>
<th>Styles found in resulting COTS-based products</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRW</td>
<td>UNAS</td>
<td>event-based, distributed processes</td>
</tr>
<tr>
<td>Quintus</td>
<td>Prolog</td>
<td>rule-based, logic computing</td>
</tr>
<tr>
<td>HP</td>
<td>SoftBench</td>
<td>event-based</td>
</tr>
<tr>
<td>FI</td>
<td>OBST</td>
<td>object-oriented, database</td>
</tr>
<tr>
<td>AT&amp;T</td>
<td>BaseWorX</td>
<td>software bus, event-based</td>
</tr>
<tr>
<td>Borland</td>
<td>Delphi</td>
<td>object-oriented, database, distributed processes</td>
</tr>
</tbody>
</table>
Common Foundation of Styles

- **Base entities** underly style-specific entities: port, data component, control component, object, data connector, control connector, trigger, system

- **Recurring conceptual features** form a space of styles

<table>
<thead>
<tr>
<th>Conceptual Feature</th>
<th>Choices</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamicism of computations (supported control transfers)</td>
<td>static, dynamic (calls, spawns)</td>
</tr>
<tr>
<td>supported data transfers</td>
<td>implicit global data distributor, explicit data channels, shared variables</td>
</tr>
<tr>
<td>triggering capability</td>
<td>yes/no</td>
</tr>
<tr>
<td>concurrency of computations</td>
<td>single-threaded, multi-threaded</td>
</tr>
<tr>
<td>distribution</td>
<td>single node, multiple nodes</td>
</tr>
<tr>
<td>layering</td>
<td>layered (connector-specific), none</td>
</tr>
<tr>
<td>encapsulation</td>
<td>yes/no</td>
</tr>
</tbody>
</table>
Effects of Concurrency & Distribution on RBDs

- **Concurrency:**
  - assuming no distribution, concurrency causes effective failure rates to decrease

\[ \lambda_1 = f_1 t_1 u_1 s \]
\[ \lambda_2 = f_2 t_2 u_2 s \quad u_1 + u_2 \leq 1 \]

\[ \lambda_{system} = \lambda_1 + \lambda_2 \]

- **Distribution:**
  - multiple concurrent, distributed blocks promotes maximum effective failure rates

\[ \lambda_1 = f_1 t_1 u_1 s_1 \]
\[ \lambda_2 = f_2 t_2 u_2 s_2 \quad u_1 + u_2 \leq 2 \]

\[ \lambda_{system} = \lambda_1 + \lambda_2 \]

- Tradeoff with performance
Effects of Dynamicism & Implicit Data Transfers on RBDs

- **Dynamicism**
  - a spawn/fork control connector produces a dynamic effective failure rate

\[
\begin{align*}
\lambda_1 &= f_1 t_1 u_1 s_1 \\
\lambda_2 &= f_2 t_2 u_2 s_2 \\
\lambda_{\text{system}} &= \lambda_1 + n\lambda_2, \quad n \geq 0
\end{align*}
\]

- countered by no distribution

- **Implicit Data Transfers**
  - hidden components and connectors may contribute to a higher effective failure rate

\[
\begin{align*}
\lambda_1 &= f_1 t_1 u_1 s_1 \\
\lambda_2 &= f_2 t_2 u_2 s_2 \\
\lambda_3 &= f_3 t_3 u_3 s_3 \\
\lambda_{\text{system}} &= \lambda_1 + \lambda_2 + \lambda_3
\end{align*}
\]
RBD’s which are Architectural Style-Aware

- Conceptual feature choices impact

<table>
<thead>
<tr>
<th>Conceptual Feature</th>
<th>Pipe/Filter</th>
<th>Main/Sub.</th>
<th>Dist. Proc.</th>
<th>Event-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamicism</td>
<td>static</td>
<td>static</td>
<td>dynamic</td>
<td>static</td>
</tr>
<tr>
<td>Supported data transfers</td>
<td>explicit data connectors</td>
<td>shared data variables</td>
<td>explicit data connectors</td>
<td>impulsive work, shared data variables</td>
</tr>
<tr>
<td>Triggering capability</td>
<td>no</td>
<td>N/A</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td>multi-threaded</td>
<td>single-threaded</td>
<td>multi-threaded</td>
<td>single-threaded</td>
</tr>
<tr>
<td>Distribution</td>
<td>unconstrained</td>
<td>single node</td>
<td>multiple nodes</td>
<td>unconstrained</td>
</tr>
<tr>
<td>Layering</td>
<td>unconstrained</td>
<td>unconstrained</td>
<td>unconstrained</td>
<td>unconstrained</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

- Traditional RBD’s are very appropriate for main/subroutine style, but modeling other styles requires accounting for the impact of their conceptual features
Example

- RBD analysis of a simple system where all components are vital to success:

\[ \lambda_{\text{system}} = \sum_{i=1}^{5} f_i \]
\[ = \frac{18}{1000\text{hrs}} \]

\[ R_{\text{system}}(10) = e^{-\lambda_{\text{system}} \cdot 10} \]
\[ = 83.5\% \]

(Failure rates per 1000 hours)
Example

- RBD analysis which reflects architectural issues

\[
\lambda_{Server} = \frac{2 + 3 + 1 + 2}{4} = 2 \\
\lambda_{Client} = 10 \\
\lambda_{system} = \lambda_{Server} + n\lambda_{Client} \quad (n \geq 0)
\]

(Failure rates per 1000 hours)
Conclusions

- Reliability block diagrams can be extended to reflect
  - architectural conceptual features
  - different connector types
- Major benefits
  - helps avoid consequence of serious oversimplification for sensitivity analyses
  - provides a tradeoff parameter (vs. performance, cost, etc.) at the architectural level
- Next steps
  - integrate into A4 (Architecture Attribute Analysis Aid)
  - continue to look for additional correlations between software architecture & software reliability