Availability Analysis of Software Decomposition for Local Recovery

TTT 05-03-2009 Enschede
Hasan Sozer

In collaboration with Hichem Boudali and Marielle Stoelinga
@ Formal Methods and Tools (FMT) Group

Outline
- Introduction
- Local Recovery
- Software Decomposition for Local Recovery
- Availability Modeling
- I/O-IMC formalism
- Availability Estimation
- Evaluation
- Conclusion

Dependability
- Fault activation
- Error propagation
- Failure recovery

Case Study: MPlayer
- MPlayer version v1.0rc1
  - Media player
  - GNU General Public License
  - ~700K lines of code
  - Linux Platform
    - Ubuntu version 7.04

Local Recovery
- Local recovery only affects the erroneous parts of the system
- Example
  - Error: deadlock in the GUI component
  - Recovery strategy: restart only the GUI component
    - Availability is high
    - (the other components are available)
    - Recovery is fast
    - (only the GUI component needs to be initialized)
Local Recovery

- Isolate the recoverable units from each other
- Employ and integrate necessary communication and recovery protocols

Software Decomposition for Local Recovery

- How to choose the set of RUs?

Scoping the design space based on domain constraints
- the number of RUs
- requires-mutex relations

Availability vs. Performance Trade-off
- Increased availability by partitioning the system into recoverable units
- Performance overhead due to module interdependencies

Availability Modeling

- MTTF: Mean Time To Failure
- MTTR: Mean Time To Recover

Several formalisms: Markov models, Petri Nets...

There exist tools that can analyze analytical models and provide availability estimations automatically

I/O-IMC Formalism

- Input/Output Interactive Markov Chains

Type of transitions
- Input signals (I)
- Output signals (O)
- Markovian transitions (rate)

Multiple I/O-IMC models can be composed together based on the common inputs they consume and outputs they generate
Modeling Approach

- 4 type of I/O-IMC models communicate with each other
  - Module I/O-IMC: failure/recovery behavior of a module
  - Recovery Manager (RM) I/O-IMC: recovery coordination
  - RU failure interface I/O-IMC: keeping track of failed modules within a RU and notifying the RM
  - RU recovery interface I/O-IMC: initiating the recovery of modules within a RU

Example I/O-IMC Models

- A Module I/O-IMC
  - A recovery interface I/O-IMC for a RU comprising the modules B and C

Example I/O-IMC Models

- A failure interface I/O-IMC for a RU comprising the modules B and C

Example I/O-IMC Models

- A RM I/O-IMC for 2 RUs

I/O-IMC Model Generation

- An algorithm explores the state space for I/O-IMC model generation

Example I/O-IMC Models

- A failure interface I/O-IMC for a RU comprising the modules B and C

Composition & Analysis Script

- A composition script is automatically generated, which merges the generated I/O-IMC models in several iterations and prunes the state space in each iteration

The result of the composition and aggregation is a regular Continuous Time Markov Chain (CTMC), which is provided to a model checker (CADP) to compute system availability

Example I/O-IMC Models

- A recovery interface I/O-IMC for a RU comprising the modules B and C

Example I/O-IMC Models

- A RM I/O-IMC for 2 RUs
Availability Analysis

- **Input:**
  - The set of modules, their MTTF & MTTR
  - The set of RUs

- **Output:** Availability Estimation

![Diagram]

Evaluation of the Analysis Approach

- **Availability Analysis**
  - Running the system (~5 hours)
  - Injecting errors
    - Every module initiates an error injection thread
    - Errors are injected according to a probability distribution
  - Calculating the up-time

```
Algorithm 1 Periodically Activated Thread for Error Injection
1. module(i) = currentTime()
2. while (true) do
3.     time.sleep = currentTime() - time.start
4.     p = 1 - e^(-time.sleep/MTTF)
5.     if random() < p then
6.         injectError()
7.     end if
8. end while
```

Evaluation of the Analysis Approach

<table>
<thead>
<tr>
<th>Modules</th>
<th>Library</th>
<th>Libraries</th>
<th>Devices</th>
<th>Modules</th>
<th>Libraries</th>
<th>Streams</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTTF (s)</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
<td>1800</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Limitations**
  - State Space Size
    - The module I/O-IMC: constant (i.e. 4 states)
    - The failure interface I/O-IMC: O(m)
    - The recovery interface I/O-IMC: O(2^m)
    - The RM I/O-IMC*: O(n!)

- **Assumptions**
  - The failure and recovery of a modules is governed by an exponential distribution
  - The RM does not fail and it always correctly detects a failing RU

<table>
<thead>
<tr>
<th>Decomposition</th>
<th>Alternative</th>
<th>Measured Availability</th>
<th>Estimated Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>all modules in I BU</td>
<td>98.70%</td>
<td>98.70%</td>
<td></td>
</tr>
<tr>
<td>GUI, GUI, the rest</td>
<td>92.31%</td>
<td>95.70%</td>
<td></td>
</tr>
</tbody>
</table>

* A RM I/O-IMC coordinating 7 RUs has 27,399 states and 397,285 transitions

Conclusion

- Local recovery is an effective technique with respect to availability and mean time to recover
- Decomposing software architecture for local recovery leads to a trade-off between performance and availability
- Availability can be estimated based on analytical models if the assumptions hold and the size of the models remain tractable

Questions

[Diagram]