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

Embedded Systems INSTITUTE

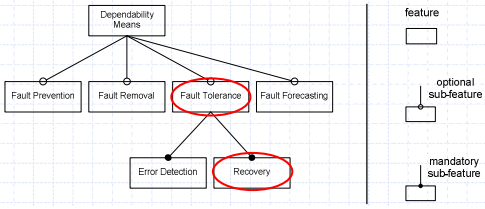
University of Twente

Outline

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

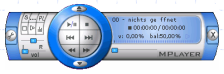
Dependability

Fault → (activation) → Error → (propagation) → Failure



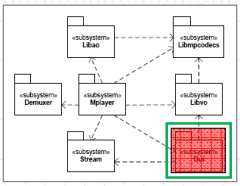
Case Study: MPlayer

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



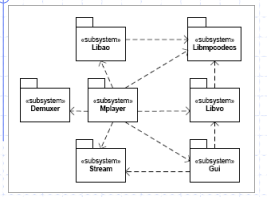
Local Recovery

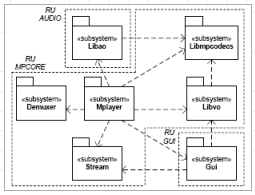
- Local recovery only affects the erroneous parts of the system
- Example
  - Error:
    deadlock in the GUI component
  - Recovery strategy:
    restart only the GUI component
    - ☺ Availability is high
      (the other components are available)
    - ☺ Recovery is fast
      (only the GUI component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



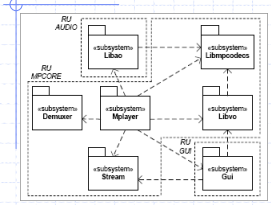
Local Recovery

Separate the system into a set of *recoverable units (RUs)*

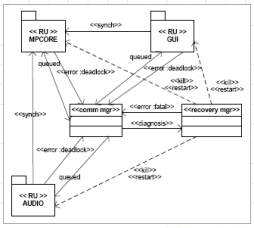




Local Recovery



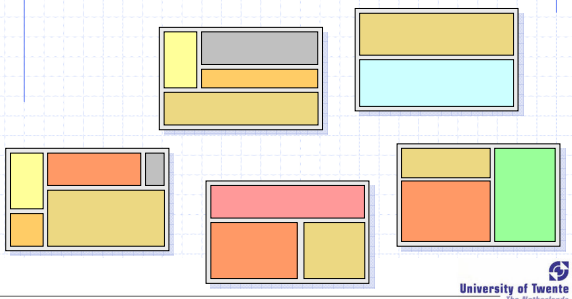
Isolate the recoverable units from each other



Employ and integrate necessary communication and recovery protocols

Software Decomposition for Local Recovery

- How to choose the set of RUs?



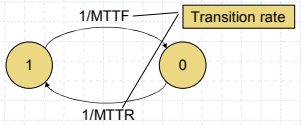
Software Decomposition for Local Recovery

- 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..

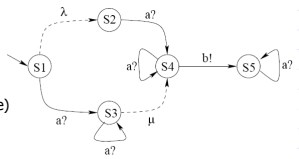
A simple availability model based on Markov Chains:



- 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 (?)
  - Output signals (!)
  - Markovian transitions (rate)



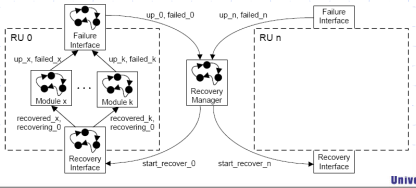
- Multiple I/O-IMC models can be composed together based on the common inputs they consume and outputs they generate

I/O-IMC Formalism

- Modularity in model building and analysis
- Management of large state spaces
  - Compositional aggregation: composing elementary models in successive iterations and reducing (i.e. aggregating) the state-space after each iteration

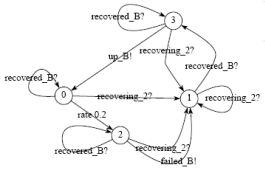
Modeling Approach

- 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

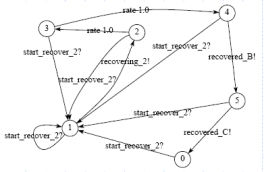


Example I/O-IMC Models

- A Module I/O-IMC

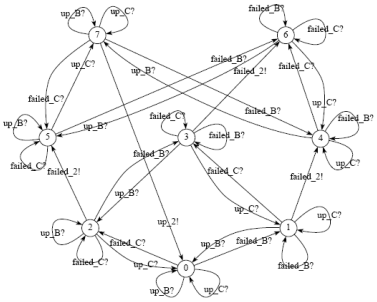


- A recovery interface I/O-IMC for a RU comprising the modules *B* and *C*



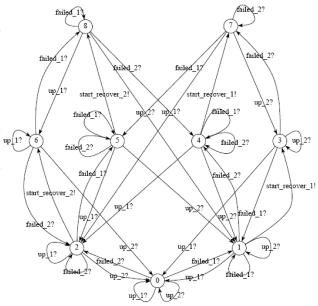
Example I/O-IMC Models

- 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

```
IOIMC: failure interface
  signature:
    input: failed(n:Int)? up(n:Int)?
    output: failed_RU! up_RU!
  states:
    set: Set[n:Int] := {}
    rufailed: Bool := false
  transitions:
    input: failed(i)?
    effect:
      if i ∉ set
        add(i, set)
    input: up(i)?
    effect:
      if i ∈ set
        remove(i, set)
    output: failed_RU!
    precondition: set.size() > 0 ∧ rufailed = false
    effect:
      rufailed := true
    output: up_RU!
    precondition: set.size() = 0 ∧ rufailed = true
    effect:
      rufailed := false
```

MIOA specification for the RU failure interface

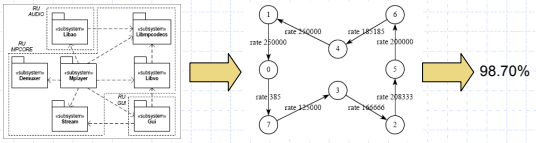
Composition & Analysis Script

- 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

```
"X.bcg" = branching stochastic reduction of(
(hide start_recover_1 in
  (hide failed_B,up_B,failed_C,up_C in
"fint_1.aut"
    ||[failed_1,failed_B,up_B,failed_C,up_C]||
    (hide recovered_B in "module_B.aut"
    ||[recovered_B]||
    (hide recovered_C in "module_C.aut"
    ||[recovered_C]||
    "rint_1.aut"))
  )
  ||[start_recover_1,failed_1,up_1]||
(hide start_recover_2 in
  (hide failed_A,up_A in "fint_2.aut"
    ||[failed_2,failed_A,up_A]||
    (hide recovered_A in "module_A.aut"
    ||[recovered_A]||
    "rint_2.aut")
  )
  ||[start_recover_2,failed_2,up_2]||
"recmgr.aut")));
```

Availability Analysis

- Input:
  - The set of modules, their MTTF & MTTR
  - The set of RUs
- Output: Availability Estimation



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 2 Periodically Activated Thread for Error Injection

```
1:  time_init ← currentTime()
2:  while TRUE do
3:      time_elapsed ← currentTime() − time_init
4:      p ← 1 − 1/e^(time_elapsed/MTTF)
5:      r ← random()
6:      if p ≥ r then
7:          injectError()
8:          break
9:      end if
10: end while
```


Evaluation of the Analysis Approach

Module	*Libao*	*Libmpcodecs*	*Demuxer*	*Mplayer*	*Libvo*	*Stream*	*Gui*
MTTR (ms)	480	500	540	800	400	400	600
MTTF (s)	60	1800	1800	1800	1800	1800	30

manually assigned for experimentation

measured from the actual implementation

Decomposition Alternative	Measured Availability	Estimated Availability
all modules in 1 RU	83.27	83.60
Gui, the rest	92.31	93.25
Gui, Libao, the rest	97.75	98.70

Limitations

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

# of modules within the RU

# of RUs

- 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

*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