A Model for the Measurement of the Runtime Testability of Component-based Systems

Alberto González Éric Piel Hans-Gerhard Gross

Delft University of Technology
Delft, The Netherlands

5^{th} Workshop in Advances in Model-based Testing
April 1st, 2009
Denver, USA

Problem Statement

Runtime Testability

Concrete Measurement

Examples

Conclusions

Problem Statement

Motivation

- New types of systems...
  - Service Oriented Architectures
  - Systems of Systems
  - Dynamic Component-based in general

Motivation

- New types of systems...
  - Service Oriented Architectures
  - Systems of Systems
  - Dynamic Component-based in general
- ...introduce new challenges...
  - Components are not available
  - Components are not known
  - Components are autonomous
  - Components are out of our control

Motivation

- New types of systems...
  - Service Oriented Architectures
  - Systems of Systems
  - Dynamic Component-based in general
- ...introduce new challenges...
  - Components are not available
  - Components are not known
  - Components are autonomous
  - Components are out of our control
- ...and require new approaches
  - Runtime Testing

Runtime Testing

Definition

Any testing method that is carried out on the final execution environment of a system is considered *Runtime Testing*.

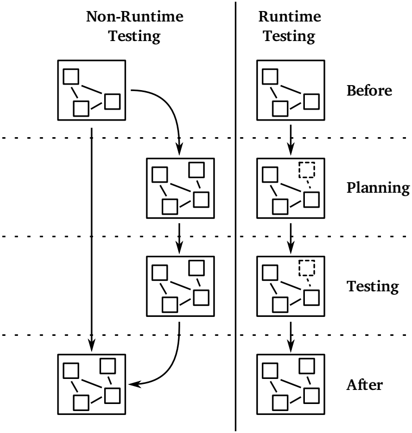
- Deployment RT: when the system is first installed
- In-service RT: while the system is in use

Runtime Testing

Definition

Any testing method that is carried out on the final execution environment of a system is considered *Runtime Testing*.

- Deployment RT: when the system is first installed
- In-service RT: while the system is in use



Runtime Testability

Runtime Testability

If we are going to runtime test we'd like to know...

- ...how resilient our system is to interferences of runtime testing.
- ...which parts of the system have to be left untested.
- ...which tests are safe to run.
- ...how to isolate the effects of the runtime tests.

Runtime Testability

1. The degree to which a *system* or a component facilitates runtime testing without being extensively affected
2. The specification of which *tests* are allowed to be performed during runtime without extensively affecting the running system

Runtime Testability

If we are going to runtime test we'd like to know...

- ...how resilient our system is to interferences of runtime testing.
- ...which parts of the system have to be left untested.
- ...which tests are safe to run.
- ...how to isolate the effects of the runtime tests.

Runtime Testability

1. The degree to which a *system* or a component facilitates runtime testing without being extensively affected
2. The specification of which *tests* are allowed to be performed during runtime without extensively affecting the running system

Affecting Factors

Test Sensitivity...

...characterises all the features of the system that, when involved in a test, will interfere with the running system or its environment in an unacceptable way.

Test Sensitivity

- Components state:
  - Tests could alter a component's state
    - Empty a bank account
  - Normal operations can influence tests (controllability)
- Component interactions:
  - Direct impact on the environment of the system
    - Launch a missile
  - Indirect influence on the state of other components/environment
    - Interacting with other test-sensitive components
- Resource constraints:
  - Tests will compete with normal operations for resources
- Availability requirements:
  - Is the component blocked during testing?

Test Sensitivity

- Components state:
  - Tests could alter a component's state
    - Empty a bank account
  - Normal operations can influence tests (controllability)
- Component interactions:
  - Direct impact on the environment of the system
    - Launch a missile
  - Indirect influence on the state of other components/environment
    - Interacting with other test-sensitive components
- Resource constraints:
  - Tests will compete with normal operations for resources
- Availability requirements:
  - Is the component blocked during testing?

Affecting Factors

Test Sensitivity...

...characterises all the features of the system that, when involved in a test, will interfere with the running system or its environment in an unacceptable way.

Test Isolation...

...is the mean test engineers have of countering the interference between tests, and the normal operation of the system and its environment. i.e., of neutralising test sensitivity.

Test Isolation

- State separation:
  - Save&Rollback, Cloning, Test sessions [SPB+06]
- Interaction separation:
  - Interception/Omission, Simulation
- Resource monitoring
  - Postpone tests, Resource Negotiation [BAM+07]
- Scheduling
  - Test preemption

Measurement

$$RTM = \frac{\text{CAN}}{\text{WANT}}$$

- Generic definition:
  - Can be tailored to any measurement of features of the system

Measurement

$$\begin{tikzcd}
RTM = \dfrac{\text{CAN}}{\text{WANT}} \arrow[d, "\text{Coverage criterion}"'] \\
RTM = \dfrac{|C_r|}{|C|}
\end{tikzcd}$$

- Generic definition:
  - Can be tailored to any measurement of features of the system
- Coverage-based:
  - Applicable to any representation with a coverage criterion
  - High-level: function points
  - Low-level: state machines

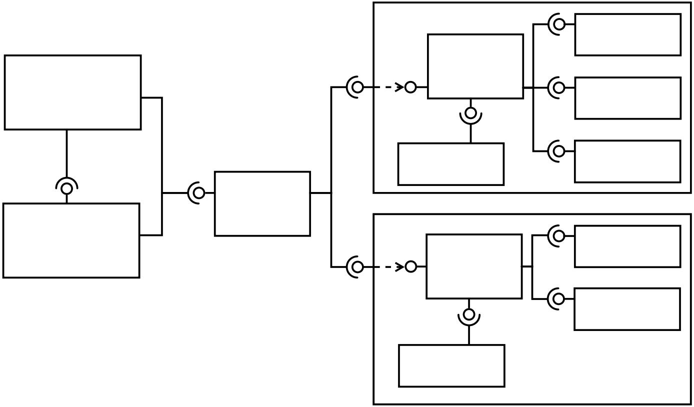
Concrete Measurement

Component-based Model

Component Interaction Graph: $CIG = (V, E)$

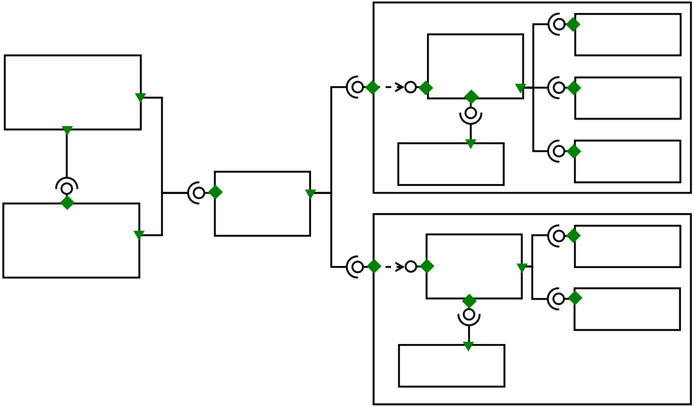
Component-based Model

Component Interaction Graph: $CIG = (V, E)$



Component-based Model

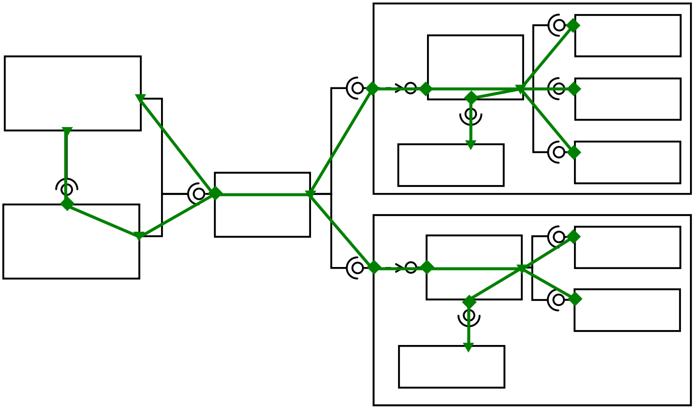
Component Interaction Graph: $CIG = (V, E)$



- (!!) Level of granularity: interface methods

Component-based Model

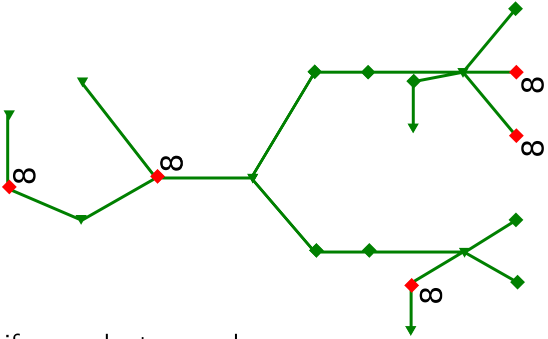
Component Interaction Graph: $CIG = (V, E)$



- (!!) Level of granularity: interface methods

Component-based Model

Component Interaction Graph: $CIG = (V, E)$



$$\tau_i = \begin{cases} 0 & \text{if } v_i \text{ can be traversed} \\ \infty & \text{otherwise} \end{cases}$$

- (!!) Level of granularity: interface methods

Coverage Criteria

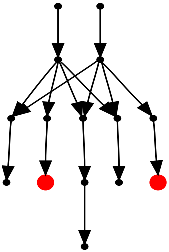
Vertex Coverage

Every provided and required method of each interface has to be tested at least once. Therefore, every vertex $v_i \in V$ must be covered.

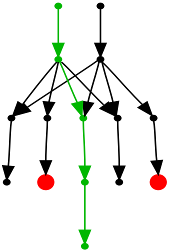
Context Dependence Coverage

A vertex v_j is context dependent on v_i if there's an invocation sequence from v_i that reaches v_j. For each of this dependences, all the possible paths $(v_i, v_{i+1}, \ldots, v_j)$ are considered viable, and need to be tested.

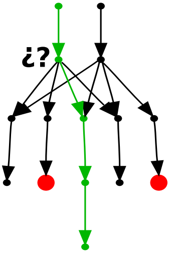
Value for the Measurement



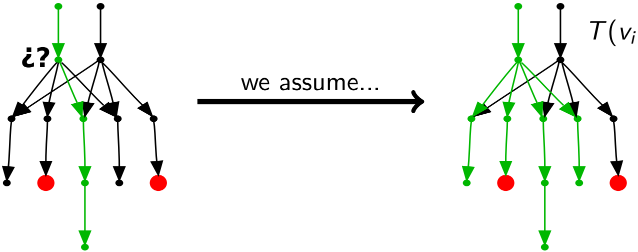
Value for the Measurement



Value for the Measurement

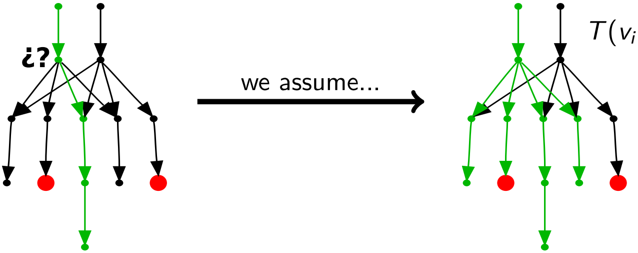


Value for the Measurement



$$T(v_i) = \sum_{v_j \in P_{v_i}} \tau_j$$

Value for the Measurement



we assume...

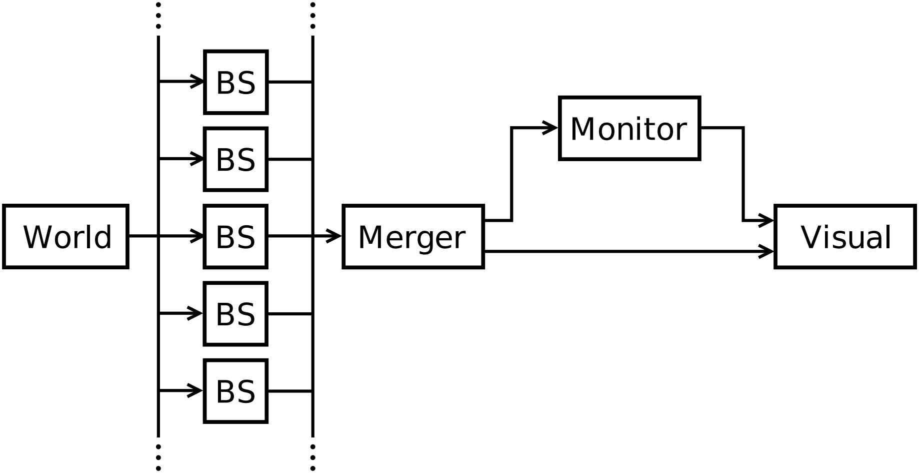
$$T(v_i) = \sum_{v_j \in P_{v_i}} \tau_j$$

$$RTM_v = \frac{|\{v_i \in V \mid T(v_i) \neq \infty\}|}{|V|}$$

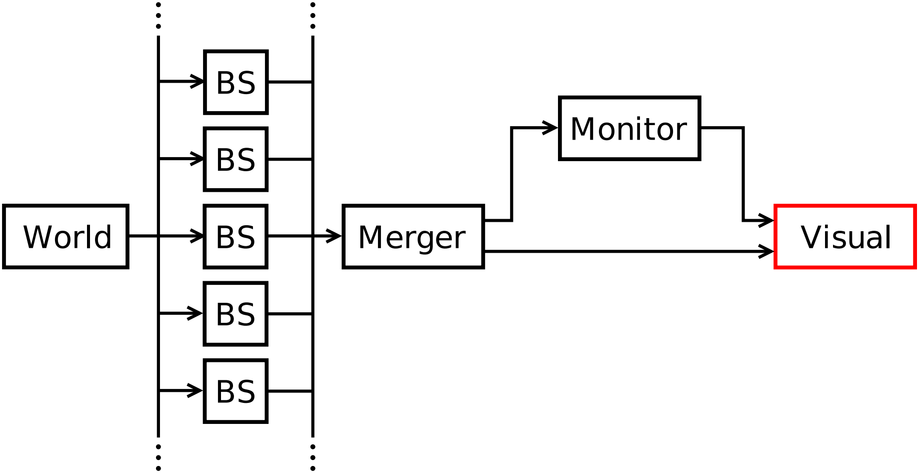
$$RTM_{c\text{-}dep} = \frac{|\{(v_i, v_j, v_k, \dots) \in CIG \mid T(v_i) \neq \infty\}|}{|\{(v_i, v_j, v_k, \dots) \in CIG\}|}$$

Examples

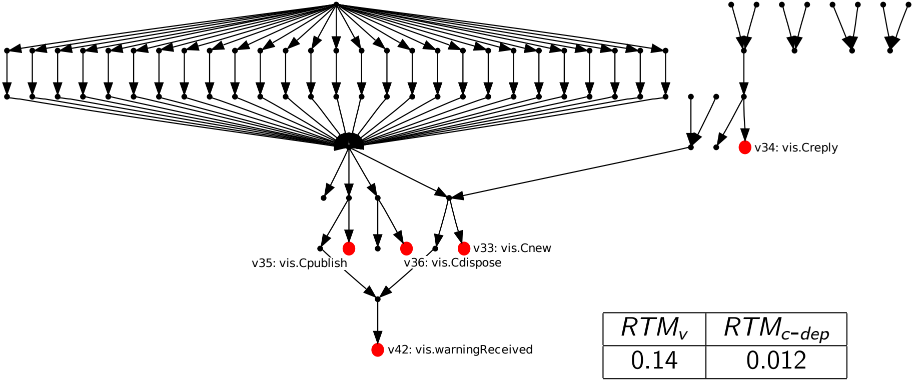
AISPlot: Component Architecture



AISPlot: Component Architecture

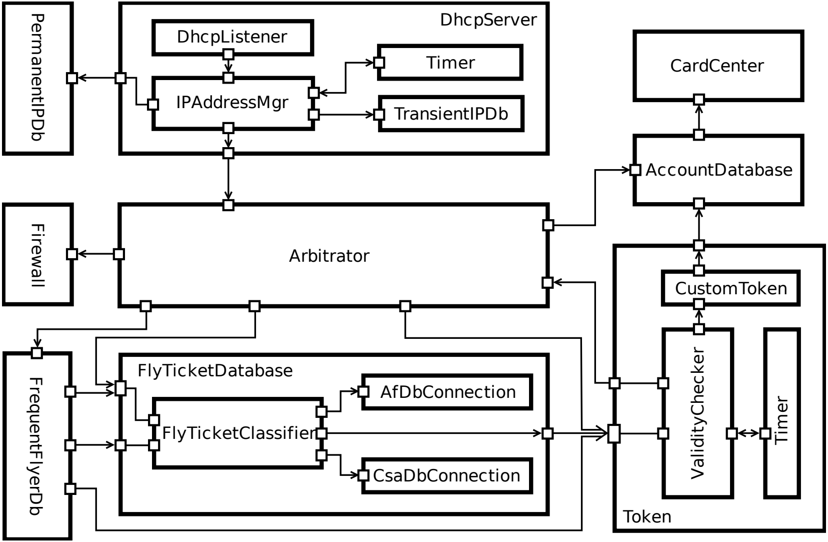


AISPlot: Interaction Graph & RTM

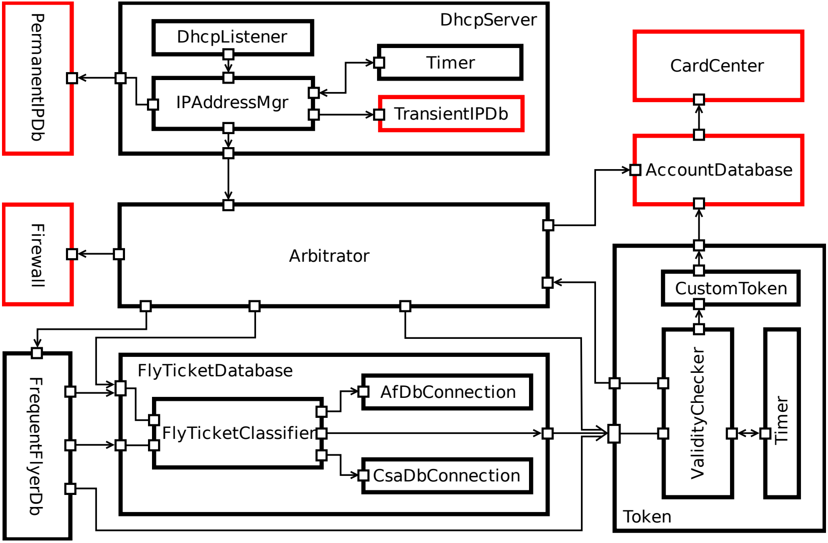


RTM_v	RTM_{c-dep}
0.14	0.012

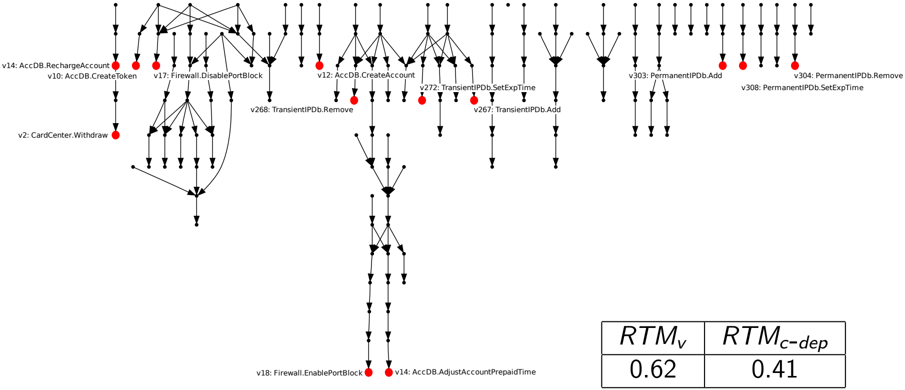
WifiLounge


PermanentIPDb
DhcpServer
DhcpListener
Timer
IPAddressMgr
TransientIPDb
CardCenter
AccountDatabase
Firewall
Arbitrator
CustomToken
FlyTicketDatabase
AfDbConnection
FrequentFlyerDb
FlyTicketClassifier
ValidityChecker
Timer
CsaDbConnection
Token

WifiLounge


PermanentIPDb
DhcpServer
DhcpListener
Timer
IPAddressMgr
TransientIPDb
CardCenter
AccountDatabase
Firewall
Arbitrator
CustomToken
FlyTicketDatabase
AfDbConnection
FlyTicketClassifier
CsaDbConnection
FrequentFlyerDb
ValidityChecker
Timer
Token

WIFILounge: Interaction Graph & RTM


v14: AccDB.RechargeAccount
v10: AccDB.CreateToken
v17: Firewall.DisablePortBlock
v2: CardCenter.Withdraw
v268: TransientIPDb.Remove
v12: AccDB.CreateAccount
v272: TransientIPDb.SetExpTime
v267: TransientIPDb.Add
v303: PermanentIPDb.Add
v304: PermanentIPDb.Remove
v308: PermanentIPDb.SetExpTime
v18: Firewall.EnablePortBlock
v14: AccDB.AdjustAccountPrepaidTime

RTM_v	$RTM_{c\text{-}dep}$
0.62	0.41

Conclusions

Potential Uses

Prediction of maximum coverage

Obtain a maximum coverage of the system based on runtime testing limitations, without looking at the specific test cases. It can be extended to accomodate other generic limitations (e.g., missing infrastructure, etc)

Evaluation of isolation techniques

Evaluate the improvement of RTM when different isolation techniques are applied. Compare different techniques.

Fix optimisation

Search algorithm to find the optimal fix, given some fix cost.

Conclusions & Future Work

- Runtime Testing is limited by the characteristics of the system
- Runtime Testability is a measurement for the impact of those limitations
- Generic coverage-based framework to measure Runtime Testability
  - We provide an instantiation for CBS on a static dependency graph
- Estimate untestable features independently of test cases
- Further work:
  - Cost-based optimisation: choose the optimal set of vertices to fix
  - Safe test-case generation: integrate test sensitivity into generation algorithms
  - Accuracy of the estimation?
  - Concrete link to reliability?

Thank you!

Bibliography

Daniel Brenner, Colin Atkinson, Rainer Malaka, Matthias Merdes, Barbara Paech, and Dima Suliman.
Reducing verification effort in component-based software engineering through built-in testing.
Information Systems Frontiers, 9(2-3):151–162, 2007.

Dima Suliman, Barbara Paech, Lars Borner, Colin Atkinson, Daniel Brenner, Matthias Merdes, and Rainer Malaka.
The MORABIT approach to runtime component testing.
In *30th Annual International Computer Software and Applications Conference*, volume 2, pages 171–176, September 2006.