Erlang - A monitoring-oriented programming language

Christian Colombo

joint work with
Adrian Francalanza, Rudolph Gatt, Kevin Falzon,
Andrew Gauci, Ruth Mizzi

University of Malta
Runtime Monitoring
Runtime Monitoring

- State space explosion
- Checks at runtime
- Links monitor to system
- Executes system and monitor concurrently
- Separates logic from exception handling
Non-defensive programming

Client -> Garbage to server

Non-defensive server

Client request is handled by the monitor

Response to client

Monitor

(the server does not try to handle error)
Runtime Monitoring

- State space explosion
- Checks at runtime
- Links monitor to system
- Executes system and monitor concurrently
- Separates logic from exception handling
Erlang Systems
Supervision tree
The Elevator Example
### Trace of the elevator execution

<table>
<thead>
<tr>
<th>Level</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Come pressed</td>
</tr>
<tr>
<td>1</td>
<td>Door opened</td>
</tr>
<tr>
<td>1</td>
<td>Go 2 pressed</td>
</tr>
<tr>
<td>1</td>
<td>Door closed</td>
</tr>
<tr>
<td>3</td>
<td>Come pressed</td>
</tr>
<tr>
<td>2</td>
<td>Door opened</td>
</tr>
<tr>
<td>2</td>
<td>Go 3 pressed</td>
</tr>
<tr>
<td>2</td>
<td>Door closed</td>
</tr>
<tr>
<td>3</td>
<td>Door opened</td>
</tr>
<tr>
<td>⋮</td>
<td></td>
</tr>
</tbody>
</table>
The door opens ONLY if requested!

Trace of the elevator execution

<table>
<thead>
<tr>
<th>Level</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Come pressed</td>
</tr>
<tr>
<td>1</td>
<td>Door opened</td>
</tr>
<tr>
<td>1</td>
<td>Go 2 pressed</td>
</tr>
<tr>
<td>1</td>
<td>Door closed</td>
</tr>
<tr>
<td>3</td>
<td>Come pressed</td>
</tr>
<tr>
<td>2</td>
<td>Door opened</td>
</tr>
<tr>
<td>2</td>
<td>Go 3 pressed</td>
</tr>
<tr>
<td>2</td>
<td>Door closed</td>
</tr>
<tr>
<td>3</td>
<td>Door opened</td>
</tr>
</tbody>
</table>

...
Current supervisor configuration
Property-aware supervisors
Possible actions to take
trace

kill & restart using another version of the code
What other versions of the code may be

- "Safe mode" version of the code
- The previous version of the code (before last patch)
- The previous (stable) implementation of the system
Summary of our idea

Existing Erlang linking mechanism is just a special case...

**Erlang**
- the supervisor is only notified of abnormal termination

**Our Proposal**
- the supervisor is notified of all relevant events, checking behaviour against properties

Not all errors lead to abnormal termination...
so force offending processes to fail-fast!
What tools are available for monitoring?

There is a tool for Java systems called Larva...

...can it be adapted for Erlang? ... ELarva!
Larva

Java system

Interesting events

Specification (Larva automata)

AspectJ Code

Monitor
Erlang system

Interesting events

Specification (Larva automata)

Elarva

Monitor

Tracing Code
Elarva

Erlang system

Interesting events

Specification (Larva automata)

Monitor

Tracing Code

Erlang system

Tracing Code

Monitor

Don’t throw away your tests!
"I have to write formal properties!"

True!... but you would probably already have them!
Don't throw away your tests!

QuickCheck
state machines

QuickCheck automata

Larva
Larva automata
Centralisation

Tracing is centralised
... we have a bottleneck of incoming events

We wish to have decentralised tracing...
Centralised tracing

Central Tracer

worker

supervisor
Decentralised tracing
A Cleaner Approach

Wrapping the worker inside a verifier

Supervisor mechanism unchanged!
Feedback

Questions, comments, suggestions...

...Thanks!