LARVA: Safer Monitoring of Real-Time Java Programs

Christian Colombo  Gordon Pace  Gerardo Schneider
Dept. of Computer Science  Dept. of Computer Science  Dept. of Applied IT
University of Malta  University of Malta  Chalmers | Univ. of Gothenburg
Malta  Malta  Sweden
(Univ. of Oslo, Norway)

7th IEEE International Conference on Software Engineering and Formal Methods (SEFM’09)
Runtime Verification

MONITORED SYSTEM

FEEDBACK

EVENTS

MONITORING SYSTEM

Specification
How to monitor a program

- Static phase:
  - Code instrumentation

- Dynamic phase:
  - Observing events from the program

- The monitor may be automatically extracted from the specification
  - Statically
  - Dynamically
A 'send' should only be followed by an 'ack'.

Runtime Verification

error

send

ack

else

else

M

!send

?ack

!ack

!send

?send

?ack

!ack

2009
A Scenario

Event

Condition

Action

Cond & actions:
- Local variables
- Can access context (eg(user)

One monitor for each user

Dynamic creation of monitors when new users

Monitors can communicate (channels)

Event:
- interact\(t\).reset();
- goodlogin\(t\).reset();
- logout\(c=0;\)
- badlogin\(c>2\)

Condition:
- Action:
- badlogin\(c++;\)
- logged in
- logged out
- inactive
- bad logins

Timer:
- \(t@30\times60\)
What Is the Message?

There is a need of a runtime verification tool that allows:

- The representation of **contextual** properties
- The **dynamic** creation of monitors
- To monitor **real-time** properties
- The **communication** between different monitors
Adding a monitor at runtime slows down the system and may invalidate certain properties which would be valid otherwise.

Eliminating a monitor at runtime speeds up the system and may invalidate certain properties which would be valid otherwise.
Logical Automata for Runtime Verification and Analysis (http://www.cs.um.edu.mt/svrg/Tools/LARVA/)
**DATE: Dynamic Automata with Timers & Events**

Communicating symbolic automata enriched with **events** and **timers**

- **Events**: e.g. method calls and exception handling
- **Timers** can reset, pause and resume

Automata are **dynamic**: automatically replicated according to **context**

- **Context** is taken into account

It supports: **Conditions** and **actions** on transitions

**Communication** between automata

- **Actions** can access code and data from the monitored program
- **Communication** is through **channel synchronization**
GLOBAL {
  VARIABLES {
    int c = 0;
    Clock t;
  }

  EVENTS {
    badlogin() = {*.badlogin()}
    clk() = {t@30*60}
    ...
  }

  PROPERTY users {
    STATES {
      BAD { badlogins inactive }
      NORMAL {loggedin }
      STARTING { loggedout }
    }
    TRANSITIONS {
      loggedout -> badlogins [badlogin\c>=2\]
      loggedout -> loggedin [goodlogin\t.reset();]
      ...
      loggedout -> loggedout [badlogin\c++;]
      loggedin -> inactive [clk\] 
    }
  }
}
LARVA – Other input languages

- DC
- QDDC
- Lustre
- DATE

LARVA

- Weaved Code
- Monitoring System
- Property Analysis
## Benchmark - Expressivity

Table 1. Expressivity features of various tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>LARVA</th>
<th>ConSpec</th>
<th>Java-MOP</th>
<th>Java-MaC</th>
<th>Hawk</th>
<th>Lola</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exceptions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Temporal Logics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Real-Time</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile Application Policies</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Invariants</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Feedback</td>
<td>✓</td>
<td>Stop.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Conditions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Numerical Queries</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*a* in specification it supports all the mentioned scopes but currently only *session* is supported

*b* restricted (cannot trigger clock events)

*c* can be extended to support real-time

*d* restricted to implementing conditions in violation/validation handling method
What Is Behind the Stage?

Definition of suitable automata for RV with real-time (DATE)

A sound translation from Phase Automata into DATEs
- There exists a translation from DC into Phase Automata (characterize "implementable" DC) ([Bouajjani et al.95], [Hoenicke06])

- Characterization of slow-down and speed-up real-time formulae
- Formal definition and theoretical results on time transformation (instantiated into the Duration Calculus)

Translation of Lustre and QDDC into DATEs
Real Case Studies

- Applications at Ixaris Systems Ltd.
  - Operates EntroPay, an online prepaid payment service
  - Services: E.g., virtual credit cards
- Runtime verification of
  - Contextual properties of objects (dynamically created)
  - Real-time: Expiration dates
  - Security: Rights, authorization, data integrity

Recent Developments:

- An extension to Larva called **LarvaStat** which is able to handle statistical properties
- An adapter connecting Larva to an SQL database, extracting events and allowing Larva to monitor offline
Demo

- A simple bank system
  - General bank system, users, accounts, transactions
- Property
  - No more than 5 users logged in at the same time
  - No more than 5 accounts per user
  - No more than 5 transactions at the same time
Conclusions

RV of real-time properties should be taken carefully

- Monitors affect behavior

We have a theory of slow-down and speed-up truth preserving properties (characterization in DC)

- We know which properties to monitor at runtime and which during testing

A tool for RV of Java – LARVA

- Mathematical framework – DATE
- Highly expressive (contextual and real-time properties)

Application to real case studies
Thank you very much for your attention!

Questions?
LARVA – Compilation into Java

- AOP to capture events
- A hierarchy of classes
  - One for each context
- Each class has a reference to its parent context (E.g. The account context, have access to the user context)
- A hashmap to keep track of the distinct objects which we are checking
How to monitor a program

- **Static** phase:
  - Code instrumentation

- **Dynamic** phase:
  - Observing events from the program

- The monitor may be *automatically extracted from the specification*
  - Statically
  - Dynamically

2009
Runtime Verification of Real-Time Properties

1. A 'send' should only be followed by an 'ack'.
2. Any 'send' must be followed by an 'ack' within 30 sec.

B "knows" that there is at most 3 sec delay between sending his 'ack' and receiving it.
The monitor "invalidates" a valid property, because it slows down the system!
Duration Calculus

DC is an interval logic: It expresses properties which hold over time intervals

- It formalizes the concept of duration

A duration calculus formula is said to hold with respect to a particular interpretation

- An interpretation is the evaluation of a set of variables with respect to time

The calculus is based on two main concepts

- **Integration**: it measures the duration of a Boolean variable over an interval
- **The chop operator**: which splits an interval into two with different formulae which must hold on each subinterval
A slide on LUSTRE Here!!
A slide on QDDC Here!!
A Scenario - Events

No 3 successive bad logins
A Scenario – Conditions & Actions

No 3 successive bad logins

- Logged out
- Goodlogin
- Logged in
- Badlogin
- Logged out
- Goodlogin
- Logged out
- Badlogin

Badlogin \ ValidAccount() \ BlockAccount();
Badlogin

2009
A Scenario - Clocks

Logged in

Logged out

Goodlogin

Logged out

Goodlogin

Logged out

Goodlogin

Logged out

Goodlogin

Logged out

Badlogin

Badlogin

Badlogin

Badlogin

Inactive

Interact \ t.reset();

t@30
A Scenario – Channels

Load Site

Prompt for PW

PressOK \ checkUserName()

PressOK \ checkPassword()

\ ChGoodlogin!

Good Login

Logged in

ChGoodlogin?

Logged out

Badlogin

Logged out

Logged out

ChGoodlogin?

Logged out

Logged out

Badlogin

Badlogin

Badlogins

Logged out

Badlogin

ChGoodlogin?
A Scenario – Dynamic Triggers

- Imagine we need to check login/logout for each user.
- We have to trigger an automaton for every user, to keep track whether each user is logged in or not.
- Use method parameters to get context.
Specifying Context

- Actions and conditions on transitions can access the context (User).
- A context can be nested to have a more specific context within it:
  - Eg: Check login for each site of each individual user.
A Scenario – Context

Trigger new automaton
FOREACH user

Load Site

Prompt for PW

PressOK \ checkUserName()

PressOK \ checkPassword()

\ Goodlogin !

Logged in

Logged out

ChGoodlogin?

ChGoodlogin?

ChGoodlogin?

ChGoodlogin?

Logged in

Logged out

Logged out

Logged out

Logged out

ChGoodlogin?

Badlogin

Badlogin

Badlogin

Badlogin

Badlogins

Trigger new automaton
FOREACH user

PressOK 

Good Login

PressOK 

checkUserName()
Benchmark – Performance

- Dummy transaction processing system (4 properties – 2 real-time)
- Memory and time required is considerable but linear to the number of objects being monitored (replication of automata).
- Compares well with Java-MOP which is the most similar work available for usage.