Online Testing of Real-time Systems

Kim G. Larsen, Marius Mikučionis, Brian Nielsen

{ kgl, marius, bnielsen } @cs.aau.dk.

Center for Embedded Software Systems  Basic Research in Computer Science  Aalborg University

http://www.cs.aau.dk/~marius/tuppaal
Overview

- Classical model-based testing.
- Motivation and related work for testing.
- Test setup: from specification to testing.
- Relativized timed input/output conformance relation.
- Online testing algorithm.
- Testing online in action.
- Symbolic techniques from UPPAAL.
- T-UPPAAL implementation details.
- Experiments with train-gate: mutants and benchmarks.
- Conclusions.
- Future work.
Classical Model-based Testing Framework

- Model-based, black-box, conformance testing offline.

Does the behavior of the implementation comply to that of specification?

Implementation

Under

Test
Classical Model-based Testing Framework

- Model-based, black-box, conformance testing offline.

- Timed, online (on-the-fly generation and execution in real-time).
Motivation and Related Work for Testing

- Verification vs. testing: abstract models vs. real world.
- “Testing is routine and boring work”: let’s automate!
- Testing requires most of development skills.
- Testing is the main validation technique used by industry.
- About 1/3 of project resources is spent on testing.
- Testing still remains ad-hoc, based on heuristics and error prone.

This work is based on the following ideas:

- Jan Tretmans’ testing theory (un-timed, with quiescence).
- TORX testing tool framework (un-timed, without environment).
- UPPAAL model-checking algorithms for timed systems.
- Timed systems scheduling (?).
Test Setup: Systems ⇒ Models ⇒ Online Testing

- Imp is (weakly) input enabled.
- Clear and explicit Env assumptions.
- Imp||Env forms a closed system.
- Observable input/output actions.

- Testing with general Env is expensive and often unnecessary.
- Flexible: only relevant behavior (Env change, guiding, debug).

- Generation and execution in RT allow long and exhaustive tests.
**Test Specification: Timed Automata Networks**

Timed automaton over $A$ is $\langle L, l_0, X, D, E, I \rangle$:
- $L$ – set of locations,
- $l_0 \in L$ – the initial location,
- $X$ – set of real-valued clocks,
- $D$ – bounded integer variables,
- $I : l \mapsto G(X)$ – location invariant mapping,
- $E \subseteq L \times G(X) \times A \times 2^{R(X)} \times L$ is a superset of directed edges: $l \xrightarrow{g,a,r} l'$ iff $\langle l, g, a, r, l' \rangle \in E$.

- Has Labeled Transition System (LTS) semantics.
- I/O, internal and timing non-determinism allow modelling parallelism, abstraction and possible time slacks.
- Test Spec: $\langle (E_1 || E_2 || \ldots || E_n) || (I_1 || I_2 || \ldots || I_n), A_{in}, A_{out}, T \rangle$
Relativized Timed Input/Output Conformance

- Idea: extend ioco (J.Tretmans) from TORX with time and env.
- Timed trace e.g.: \( \sigma = \text{coin}? \cdot 5 \cdot \text{req}? \cdot 2 \cdot \text{weakCoffee}! \cdot 9 \cdot \text{coin}? \)
- \( \text{TTr}(s) \) – set of timed traces from state \( s \): \( \{ \sigma \in (A \cup \mathbb{R}_{\geq 0})^* \mid s \xrightarrow{\sigma} \} \)
- Timed trace inclusion as conf. relation: \( \text{TTr}(i) \subseteq \text{TTr}(s) \)
- No illegal output and legal output is observed at right time.

\[
\text{Out}(P) \overset{\text{def}}{=} \bigcup \{ \alpha \in (A_{out} \cup \mathbb{R}_{\geq 0}) \mid p \in P. p \xrightarrow{\alpha} \}
\]

- Relativized Timed Input/Output Conformance:
  \( s \ xrioco_e t \overset{\text{def}}{=} \forall \sigma \in \text{TTr}(e). \text{Out}( (e, s) \text{ After } \sigma ) \subseteq \text{Out}( (e, t) \text{ After } \sigma ) \)

\[
s \ xrioco_e t \iff \text{TTr}(s) \cap \text{TTr}(e) \subseteq \text{TTr}(t) \cap \text{TTr}(e)
\]

- Environment ordering. \( f \) is more discriminating than \( e \):

\[
e \subseteq f \overset{\text{def}}{=} r\text{rioco}_f \subseteq r\text{rioco}_e
\]
Timed I/O Conformance Relation Example

Specification $s$

Implementation $i_1$

Implementation $i_2$

<table>
<thead>
<tr>
<th>Trace, $\sigma$</th>
<th>Out($s$ After $\sigma$)</th>
<th>Out($i_1$ After $\sigma$)</th>
<th>Out($i_2$ After $\sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c \cdot 2$</td>
<td>$\mathbb{R}_{\geq 0}$</td>
<td>$\mathbb{R}_{\geq 0}$</td>
<td>$\mathbb{R}_{\geq 0}$</td>
</tr>
<tr>
<td>$c \cdot 4 \cdot r \cdot 1$</td>
<td>${wCoffee, sCoffee} \cup [0, 4]$</td>
<td>$[0, 1]$</td>
<td>$[0, 2]$</td>
</tr>
<tr>
<td>$c \cdot 4 \cdot r \cdot 2$</td>
<td>${wCoffee, sCoffee} \cup [0, 3]$</td>
<td>${wCoffee, 0}$</td>
<td>${wCoffee} \cup [0, 1]$</td>
</tr>
<tr>
<td>$c \cdot 5 \cdot r \cdot 3$</td>
<td>${sCoffee} \cup [0, 2]$</td>
<td>${sCoffee, 0}$</td>
<td>$[0, 4]$</td>
</tr>
<tr>
<td>$c \cdot 5 \cdot r \cdot 5$</td>
<td>${sCoffee, 0}$</td>
<td>$\emptyset$</td>
<td>$[0, 2]$</td>
</tr>
</tbody>
</table>
Randomized Test Generation and Execution Online

while $\mathcal{Z} \neq \emptyset \land \#\text{iterations} \leq T$ do choose randomly:

1. if $\text{EnvOutput}(\mathcal{Z}) \neq \emptyset$ // offer an input
   randomly choose $a \in \text{EnvOutput}(\mathcal{Z})$
   send $a$ to IUT
   $\mathcal{Z} := \mathcal{Z} \text{ After } a$

2. randomly choose $\delta \in \text{Delays}(\mathcal{Z})$ // wait for an output
   sleep for $\delta$ time units and wake up on output $o$
   if $o$ occurs at $\delta' \leq \delta$ then
     $\mathcal{Z} := \mathcal{Z} \text{ After } \delta'$
     if $o \notin \text{ImpOutput}(\mathcal{Z})$ then return fail
   else $\mathcal{Z} := \mathcal{Z} \text{ After } o$
   else $\mathcal{Z} := \mathcal{Z} \text{ After } \delta$ // no output within $\delta$ delay

3. $\mathcal{Z} := \{(s_0, e_0)\}$, reset IUT //reset and restart
   if $\mathcal{Z} = \emptyset$ then return fail else return pass
Randomized Test Generation and Execution Online

while $\mathcal{Z} \neq \emptyset \land \#\text{iterations} \leq T$ do choose randomly:

1. if $\text{EnvOutput}(\mathcal{Z}) \neq \emptyset$ // offer an input
   randomly choose $a \in \text{EnvOutput}(\mathcal{Z})$
   send $a$ to IUT
   $\mathcal{Z} := \mathcal{Z}$ After $a$

2. randomly choose $\delta \in \text{Delays}(\mathcal{Z})$ // wait for an output
   sleep for $\delta$ time units and wake up on output $o$
   if $o$ occurs at $\delta' \leq \delta$ then
     $\mathcal{Z} := \mathcal{Z}$ After $\delta'$
     if $o \notin \text{ImpOutput}(\mathcal{Z})$ then return fail
     else $\mathcal{Z} := \mathcal{Z}$ After $o$
   else $\mathcal{Z} := \mathcal{Z}$ After $\delta$

3. $\mathcal{Z} := \{(s_0, e_0)\}$, reset IUT // no output within $\delta$ delay
   if $\mathcal{Z} = \emptyset$ then return fail else return pass //reset and restart

sound and complete in limit
Testing Online in Action

Symbolic state set:
\[ \{\langle k_0 l_0, 0 \leq x \leq 0 \rangle\} \]

EnvOutput: \{coin\}

EnvInput: \emptyset

ImpOutput: \emptyset

Wait for output (delay) or offer input?
Testing Online in Action

Symbolic state set:
\[ \{ \langle k_0 l_0, 0 \leq x \leq 0 \rangle \} \]

EnvOutput: \{coin\}
EnvInput: \{\}
ImpOutput: \{\}

Let’s offer input choose (the only) "coin"
Testing Online in Action

Symbolic state set:
\( \{k_1 l_1, 0 \leq x \leq 0\} \)

EnvOutput: \( \{\text{req}\} \)

EnvInput: \( \emptyset \)

ImpOutput: \( \emptyset \)

Update the state set and other variables
Testing Online in Action

Testing–UPPAAL

Symbolic state set:
\[ \{ (k_1 l_1, 0 \leq x \leq 0) \} \]
EnvOutput: \{ req \}
EnvInput: \emptyset
ImpOutput: \emptyset

Wait or offer input?
Let’s wait for 5 units
Testing Online in Action

Symbolic state set:
\{\langle k_1 l_1, 5 \leq x \leq 5 \rangle\}

EnvOutput: \{\text{req}\}

EnvInput: \emptyset

ImpOutput: \emptyset

..no output so far: update the state set..
Testing Online in Action

Symbolic state set:
\{<k_1 l_1, 5 \leq x \leq 5>\}

EnvOutput: \{req\}

EnvInput: \emptyset

ImpOutput: \emptyset

Wait or offer input? let’s offer "req"
Testing Online in Action

Symbolic state set:
\[ \{ k_2 l_2, 0 \leq x \leq 0 \}, \{ k_2 l_3, 0 \leq x \leq 0 \} \]

EnvOutput: \( \emptyset \)
EnvInput: \{ weakCoffee, strongCoffee \}
ImpOutput: \{ weakCoffee, strongCoffee \}

Update the state set and other variables
Testing Online in Action

Symbolic state set:
\[ \{\langle k_2 l_2, 0 \leq x \leq 0 \rangle, \langle k_2 l_3, 0 \leq x \leq 0 \rangle\} \]

EnvOutput: \(\emptyset\)

EnvInput: \{weakCoffee, strongCoffee\}

ImpOutput: \{weakCoffee, strongCoffee\}

Wait or offer input? Let’s wait for 4 units
Testing Online in Action

Testing–UPPAAL

Symbolic state set:
\[ \langle k_2 l_3, 4 \leq x \leq 4 \rangle \]

EnvOutput:  \( \emptyset \)

EnvInput:  \{ strongCoffee \}

ImpOutput:  \{ strongCoffee \}

...no output so far: update the state set..
Testing Online in Action

Testing-UPPAAL

Symbolic state set:
\[ \{ (k_2l_3, 4 \leq x \leq 4) \} \]

EnvOutput: \( \emptyset \)

EnvInput: \( \{ \text{strongCoffee} \} \)

ImpOutput: \( \{ \text{strongCoffee} \} \)

Wait or offer input?
Let’s wait for 2 units
Testing Online in Action

Symbolic state set:
\[ \langle k_2, l_3, 4 \leq x \leq 4 \rangle \]

EnvOutput: \( \emptyset \)

EnvInput: \{ strongCoffee \}

ImpOutput: \{ strongCoffee \}

got output after 0 delay: update the state set
Testing Online in Action

Testing–UPPAAL

Symbolic state set:
\{\langle k_2 l_3, 4 \leq x \leq 4 \rangle\}

EnvOutput: ∅

EnvInput: \{strongCoffee\}

ImpOutput: \{strongCoffee\}

Adapter

Implementation

(what if there is a bug?)
Let’s wait for 2 units
Testing Online in Action

Symbolic state set:
- $\emptyset$
- EnvOutput: $\emptyset$
- EnvInput: $\emptyset$
- ImpOutput: $\emptyset$

"..no output so far: update the state set.. (!)"

http://www.cs.aau.dk/~marius/tuppaal
Symbolic Techniques in UPPAAL

Zone is a conjunction of clock constraints of the form:
\[ \{ x_i - x_j < c_{ij} \} \cup \{ a_i < x_i \} \cup \{ x_j < b_j \} \quad \text{where} \quad \prec \in \{ <, \leq \} \]

\[ z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)] \]
Symbolic Techniques in UPPAAL

- **Zone** is a conjunction of clock constraints of the form:
  \[ \{x_i - x_j < c_{i,j}\} \cup \{a_i < x_i\} \cup \{x_j < b_j\} \text{ where } \prec \in \{<, \leq\} \]

- **Difference bound matrix** – compact representation.

\[ z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)] \]

\[
\begin{array}{ccc}
0 & y & x \\
0 & -5 & 0 \\
y & \infty & -4 \\
x & 3 & \infty & -
\end{array}
\]
Symbolic Techniques in UPPAAL

- **Zone** is a conjunction of clock constraints of the form:
  \[ \{x_i - x_j < c_{ij}\} \cup \{a_i < x_i\} \cup \{x_j < b_j\} \text{ where } \prec \in \{<, \leq\} \]

- **Difference bound matrix** – compact representation.

- Symbolic state set \( \mathcal{Z} = \{\langle l_1, z_1\rangle, \ldots, \langle l_n, z_n\rangle\} \)

\[
z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)]
\]
Symbolic Techniques in UPPAAL

- **Zone** is a conjunction of clock constraints of the form:
  \[ \{x_i - x_j < c_{ij}\} \cup \{a_i < x_i\} \cup \{x_j < b_j\} \text{ where } \prec \in \{<, \leq\} \]

- **Difference bound matrix** – compact representation.

- **Symbolic state set** \( \mathcal{Z} = \{\langle l_1, z_1\rangle, \ldots, \langle l_n, z_n\rangle\} \)

- **Action transition**: \( \langle l, z \rangle \xrightarrow{a} \langle l', (z \land g)_r \land I(l') \rangle \) **iff**:
  \( l \xrightarrow{g,a,r} l' \) is \( a \)-action transition and \( z \land g \neq \emptyset, (z \land g)_r \land I(l') \neq \emptyset \).

\[
z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)]
\]

<table>
<thead>
<tr>
<th>( x )</th>
<th>0</th>
<th>5</th>
<th>( x )</th>
</tr>
</thead>
</table>
| 0 | -5 | 0 | \( x \leq 3 \)
| \( y \) | -4 | \( y \) |
| \( x \) | 3 | \( x \) | \( y := 4 \)

---

Symbolic Techniques in UPPAAL

- Zone is a conjunction of clock constraints of the form:
  \[ \{ x_i - x_j < c_{ij} \} \cup \{ a_i < x_i \} \cup \{ x_j < b_j \} \text{ where } \prec \in \{ <, \leq \} \]

- Difference bound matrix – compact representation.

- Symbolic state set \( Z = \{ \langle \bar{l}_1, z_1 \rangle, \ldots, \langle \bar{l}_n, z_n \rangle \} \)

- Action transition: \( \langle \bar{l}, z \rangle \xrightarrow{a} \langle \bar{l}', (z \land g)_r \land I(\bar{l}') \rangle \text{ iff: } l \xrightarrow{g,a,r} l' \) is an action transition and \( z \land g \neq \emptyset, (z \land g)_r \land I(\bar{l}') \neq \emptyset \).

- Delay transition: \( \langle \bar{l}, z \rangle \xrightarrow{\delta} \langle \bar{l}, z^+ \delta \land I(\bar{l}) \rangle \text{ iff } z^+ \delta \land I(\bar{l}) \neq \emptyset. \)

\[ z = [(y - x \leq 4) \land (y \geq 5) \land (x \leq 3)] \]

![Diagram](http://www.cs.aau.dk/~marius/tuppaal)
T-UPPAAL Implementation Details

Reachability algorithms for afterDelay and afterAction

\[
\text{Closure}_{\delta_T}(\mathcal{Z}, d) = \bigcup_{0 \leq \delta \leq d} \left\{ \langle \bar{l}', z' \rangle \mid \langle \bar{l}, z \rangle \in \mathcal{Z}, \langle \bar{l}, z \rangle \xrightarrow{\delta} \langle \bar{l}', z' \rangle \right\}
\]

\[
\mathcal{Z} \text{ After } d = \left\{ \langle \bar{l}, z' \rangle \mid \langle \bar{l}, z \rangle \in \text{Closure}_{\delta_T}(\mathcal{Z}, d), z' = (z \land (t == d))_{t:=0} \right\}
\]
Mutant Experiment: Error Detection Capability

- Specification: train-gate example of 9 timed automata.
- Implementation: 4 threads with a shared queue in C++.
- 7 mutants: M1-M6 with seeded error, M0 correct.

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Number of input actions</th>
<th>Duration, ( mtu )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Avg</td>
</tr>
<tr>
<td>M1</td>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>M2</td>
<td>2</td>
<td>4.6</td>
</tr>
<tr>
<td>M3</td>
<td>2</td>
<td>4.7</td>
</tr>
<tr>
<td>M4</td>
<td>6</td>
<td>8.5</td>
</tr>
<tr>
<td>M5</td>
<td>4</td>
<td>5.6</td>
</tr>
<tr>
<td>M6</td>
<td>2</td>
<td>14.1</td>
</tr>
<tr>
<td>M0</td>
<td>3565</td>
<td>3751.4</td>
</tr>
</tbody>
</table>
Benchmark Data: Computing Performance (instances)
Benchmark Data: Computing Performance (means)

after delay

after action

Symbolic state set size

mean CPU time, micro-s

Symbolic state set size

mean CPU time, micro-s

http://www.cs.aau.dk/~marius/tuppaal

October 13, 2004. BRICS – p.15/20
# Benchmark Data: Summary

**Executed on Sun SPARC, 8x900MHz, 32GB RAM, Sun Solaris 9.**

<table>
<thead>
<tr>
<th>Mutant</th>
<th>Number of states in ( \mathcal{Z} )</th>
<th>CPU execution time, ( \mu s )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>After (delay)</td>
<td>After (action)</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>Max</td>
</tr>
<tr>
<td>M1</td>
<td>2.3</td>
<td>18</td>
</tr>
<tr>
<td>M2</td>
<td>2.3</td>
<td>22</td>
</tr>
<tr>
<td>M3</td>
<td>2.2</td>
<td>22</td>
</tr>
<tr>
<td>M4</td>
<td><strong>2.8</strong></td>
<td><strong>24</strong></td>
</tr>
<tr>
<td>M5</td>
<td><strong>2.8</strong></td>
<td><strong>24</strong></td>
</tr>
<tr>
<td>M6</td>
<td>2.7</td>
<td>27</td>
</tr>
<tr>
<td>M0</td>
<td><strong>2.7</strong></td>
<td><strong>31</strong></td>
</tr>
</tbody>
</table>

[BRICS](http://www.cs.aau.dk/~marius/tuppaal)
Conclusions

- Online real-time testing theoretically sound and complete.
- Relativized conformance allows to minimize cost of testing.
- Environment assumptions should be known and explicit.
- Implemented in T-UPPAAL using efficient algorithms from UPPAAL.
- Encouraging error detection capability and performance.
- T-UPPAAL allows abstract and non-deterministic specifications.
- Extreme non-determinism may degrade performance.
Summary and Future Work

TestSpec: MEnv || MImp

Diagnostic Data
traces, verdicts

T-UPPAAL online tester

System under test

adapter
Summary and Future Work

Emulator
env. simulator

System
under test

TestSpec:
MEnv || MImp

Monitor
test oracle

Diagnostic Data
traces, verdicts
Summary and Future Work

Guiding hints

Selector cov. into guiding

Emulator env. simulator

TestSpec: MEnv || MImp

Monitor test oracle

System under test

Adapter

Coverage facts

Diagnostic Data

traces, verdicts

BRICS
Summary and Future Work

Guiding hints
Selector cov. into guiding

Emulator env. simulator
TestSpec: MEnv || MImp
Monitor test oracle

Converter trace to TAN
Coverage facts

System under test
Coverage criteria

Diagnostic Data
deadend-states
branching points
traces, verdicts

Model Checker
breadth-first reach.

Generator offline tester

Concrete Simulator
cov. display, trace exp.
Summary and Future Work

**Guiding hints**

- Selector cov. into guiding

**Emulator env. simulator**

- Test Spec: MEnv || MImp

**Monitor test oracle**

- Diagnostic Data deadend-states branching points traces, verdicts

**System under test**

- Coverage criteria

**Generator offline tester**

- Model Checker breadth–first reach.

- Best traces

**Converter**

- Trace to TAN

**Concrete Simulator**

- Cov. display, trace exp.

**Prototyping via simulation:**

- Emulator env. simulator

- Emulator imp. simulator

- Sys. Spec: MEnv || MImp

**Execution monitoring:**

- Component 1
  - Input
  - In execution

- Component 2
  - Output
  - In execution

- Sys. Spec: MC1 || MC2

- Monitor test oracle

This diagram outlines the flow of information and components involved in the prototyping and execution monitoring processes.
Future Work

- Clock synchronization, event observation uncertainty.
- Value passing mechanisms in specification and adapters.
- New UPPAAL features (broadcast, committed, U-Code).
- Termination of testing (specify property expressions?).
- Relativized conformance in practice: specialized applications of generalized controllers, test-case guiding, debugging.
- T-UPPAAL in monitoring, testing via simulation and monitoring.
- Coverage estimation, use coverage in guiding.
- Relativized conformance in interface compatibility: unit testing.
- Diagnostic information display; model learning during experiment.
- Industrial case studies.
Danfoss Case Study: EKC – Refrigeration Controller