Model Based Testing of Embedded Systems

Brian Nielsen
Arne Skou

{bnielsen | ask}@cs.auc.dk
Automated Model Based Conformance Testing

Does the **behavior** of the **blackbox** implementation **comply** to that of the specification?
**Specification Based Testing**

**A Specification**

Timed Automaton = FSM + Clocks (dense) + Guards + Resets

- \( x_1 \sim c, \ x_1 - x_2 \sim c, \) where \( \sim \in \{<,\leq,\geq,>\} \)
- \( x:=c \)
- Semantic state: \((l,\bar{u})\) \(\ (l_1, x = 1.17)\)

**Test cases**

- Click! Wait 1.5 click! DBLClick? (pass)
- Click! Wait 5 click! DBLClick? (fail)
- Click! Wait 0.1 click! DBLClick? (pass)
Exercises
Simple TA-specification

1) compute a timed trace that checks that a "c" can be produced!

2) How many tests do you think are necessary?

Generated test case

12 test cases are generated!
How do we cope with real-life specs?

Philips Sender with collision detection
I conforms-to S ??
I conforms-to S ??

\( i \)

\( \text{coin?} \)
\( \text{token?} \)
\( \text{coffee!} \)
\( \text{tea!} \)

\( ioco \)

\( S \)

\( \text{coin?} \)
\( \text{token?} \)
\( \text{coffee!} \)
\( \text{tea!} \)
I conforms-to $S$ ??
I conforms-to S ??
Implementation Relation $ioco$

$$iocios =_{\text{def}} \forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)$$

$$p \text{ after } \sigma = \{ p' \mid p \xrightarrow{\sigma} p' \}$$

$$p \xrightarrow{\delta} p = p \xrightarrow{L_U} p = \forall x! \in L_U \cup \{\tau\}: p \xrightarrow{x!}$$

$$\text{out}(P) = \{ x! \in L_U \mid p \xrightarrow{!x}, p \in P \}$$

$$\cup \{ \delta \mid p \xrightarrow{\delta} p, p \in P \}$$

$$\text{Straces}(s) = \{ \sigma \in (L \cup \{\delta\})^* \mid s \xrightarrow{\sigma} \}$$
Implementation Relation \textit{ioco}

\[
i \text{ ioco } s = \text{def } \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

\[
\text{out}(i \text{ after } \text{coin?}) = \{ \text{coffee!} \}
\text{out}(s \text{ after } \text{coin?}) = \{ \text{coffee!} \}
\text{out}(i \text{ after } \text{token?}) = \{ \text{tea!} \}
\text{out}(s \text{ after } \text{token?}) = \emptyset
\]

But \ text{token?} \notin \text{Straces}(s)
Implementation Relation

\( i \ ioco \ s \) =\_def \ \forall \sigma \in Straces(s) : \ out(\ i \ afte\ r \ \sigma) \subseteq \ out(\ s \ afte\ r \ \sigma) \)

\begin{align*}
\text{out}(\ i \ afte\ r \ \text{coin?}) & = \{ \text{coffee!} \} \\
\text{out}(\ i \ afte\ r \ \text{token?}) & = \{ \text{tea!} \}
\end{align*}

\begin{align*}
\text{out}(\ s \ afte\ r \ \text{coin?}) & = \{ \text{coffee!} \} \\
\text{out}(\ s \ afte\ r \ \text{token?}) & = \{ \text{tea!} \}
\end{align*}
Implementation Relation $i \ ioco \ s$

$$i \ ioco \ s \ =_{\text{def}} \ \forall \sigma \in Straces(s) : \ out \ (i \ \text{after} \ \sigma) \ \subseteq \ out \ (s \ \text{after} \ \sigma)$$

\[ out(i \ \text{after} \ \text{token?}) = \{ \delta \} \quad \text{out}(s \ \text{after} \ \text{token?}) = \{ \text{tea!} \} \]
Implementation Relation \( \text{ioco} \)

\[
i \text{ioco} \ s \ =_{\text{def}} \ \forall \sigma \in \text{Straces}(s) : \ \text{out}(i \ \text{after} \ \sigma) \subseteq \text{out}(s \ \text{after} \ \sigma)
\]

\[
\text{out}(i \ \text{after} \ \text{coin?}) = \{ \delta, \text{coffee!} \} \quad \text{out}(s \ \text{after} \ \text{coin?}) = \{ \text{coffee!} \}
\]
Timed Conformance??

**Example Traces**

- \(c? .2 .r? .2 .weakC\)
- \(c? .5 .r? .4 .strongC\)

Specification

- **I1** \(rt\)-ioco S
- **I2** \(rt\)-ioco S
Real-Time Conformance

• $\text{TTr}(s)$: the set of *timed traces* from $s$
  - eg.: $\sigma = \text{grasp?}\cdot50\cdot\text{release?}\cdot50$

• $\text{Out}(s \text{ after } \sigma) = \text{possible outputs and delays after } \sigma$
  - eg. $\text{Out} \text{ (interface after grasp?\.220)} = \{\text{touch!}, 0..270\}$

• $\text{rt-ioco } s \overset{\text{def}}{=} \forall \sigma \in \text{TTr}(s): \text{Out}(i \text{ after } \sigma) \subseteq \text{Out}(s \text{ after } \sigma)$
  - $\text{TTr}(i) \subseteq \text{TTr}(s)$

• **Intuition**
  - never produces illegal output, and
  - always produces required output in time
Sample Cooling Controller

IUT-model

Env-model

On!
Off!
Low?
Med?
High?
Env. Modeling

- Realism and Guiding
  - $E_M$: Any action possible at any time
  - $E_1$: Only realistic temperature variations
  - $E_2$: Temperature never increases when cooling
  - $E_L$: No inputs (completely passive)

$$E_L \subseteq E_2 \subseteq E_1 \subseteq E_M$$
## Implementation relation

**Relativized real-time io-conformance**

Let $P$ be a set of states

- $\text{TTTr}(P)$: the set of *timed traces* from states in $P$
- $P \text{ after } \sigma = \text{the set of states reachable after timed trace } \sigma$
- $\text{Out}(P) = \text{possible outputs and delays in } P$

- $i \text{ rt-ioco}_e s = \text{def}$
  - $\forall \sigma \in \text{TTTr}(e): \text{Out}((e,i) \text{ after } \sigma) \subseteq \text{Out}((e,s) \text{ after } \sigma)$

- $i \text{ rt-ioco}_e s \text{ iff } \text{TTTr}(i) \cap \text{TTTr}(e) \subseteq \text{TTTr}(s) \cap \text{TTTr}(e)$

**Intuition, for all relevant environment behaviors**

- Never produces illegal output, and
- Always produces required output in time
- $\sim$timed trace inclusion
Sample Cooling Controller

IUT

\[ C' r \quad \text{rt} \quad \text{ioco} \quad E_M \quad C r \]

Env-model
Test Generation
Touch-sensitive Light-Controller

Diagram: Interface, Switch, Dim, User
Test Generation Principles

- Test purpose based generation
  - “test that light can become max”
  - Formalize purpose
  - Use model to compute sequence that meets purpose (inputs and expected outputs)

- Model Coverage
  - State-coverage
  - Transition
  - Def-Use pairs
  - ...

- Randomized model interpretation
- Fault-Models
- ONLINE (randomized) testing
Time Optimal Real-Time Test Generation using UPPAAL

Anders Hessel,
Paul Pettersson
Uppsala University
Sweden

Kim Larsen,
Brian Nielsen,
Arne Skou
Aalborg University
Denmark
Testing Process

1. System specification
2. Test cases (abstract)
3. Executable test cases
4. Verdict
Test Setup

```
out(IGrasp); //@900
silence(500);
silence(1000);
in(OSetLevel,1,dfTolerance);
silence(1000);
in(OSetLevel,2,dfTolerance);
silence(1000);
```
“Scripts” for LightControl

Events: const IGrasp=0; const int IRelease=1; const int OSetLevel=0;

• void **out**(int eventNo);
  send eventNo to IUT at now();

• void **silence**(int msDelay);
  expect no outputs for msDelay: otherwise fail

• void **in** (int eventNo,int par, int msTolerance);
  expect input event(par) before now()+msTolerance
  otherwise fail

• void **at**(int eventNo, int par, int msTime, int msTolerance);
  expect input eventNo(par) at time msTime from
  start of test+/- msTolerance
Timed Tests

\[ T_{\text{sw}} = 4 \]
\[ T_{\text{idle}} = 20 \]

**INFINITELY MANY SEQUENCES!!!!!!**

**EXAMPLE** test cases for **Interface**

\[ 0 \cdot \text{grasp!} \cdot 210 \cdot \text{release!} \cdot \text{touch?} \cdot \text{PASS} \]
\[ 0 \cdot \text{grasp!} \cdot 317 \cdot \text{release!} \cdot \text{touch?} \cdot 2\frac{1}{2} \cdot \text{grasp!} \cdot 220 \cdot \text{release!} \cdot \text{touch?} \cdot \text{PASS} \]
\[ 1000 \cdot \text{grasp!} \cdot 517 \cdot \text{starthold?} \cdot 100 \cdot \text{release!} \cdot \text{endhold?} \cdot \text{PASS} \]

\[ \bullet \text{Epsilon=200ms} \]
\[ \bullet \text{Delta=500ms} \]

**INFINITELY MANY SEQUENCES!!!!!!**
Optimal Tests

- **Shortest** test for max light??
- **Fastest** test for max light??
- **Fastest** edge-covering test suite??
- Least *power* consuming test??
Test Purposes 1

A specific test objective (or observation) the tester wants to make on SUT

Environment model

System model

TP1: Check that the light can become bright:

\[ E<> L == 10 \]

• Shortest (and fastest) Test:

\[
\begin{align*}
\text{out}(\text{IGrasp}); & \quad \text{silence}(500); \quad \text{in}(\text{OSetLevel}, 0); \quad \text{silence}(1000); \\
\text{in}(\text{OSetLevel}, 1); & \quad \text{silence}(1000); \quad \text{in}(\text{OSetLevel}, 2); \quad \text{silence}(1000); \\
\text{in}(\text{OSetLevel}, 3); & \quad \text{silence}(1000); \quad \text{in}(\text{OSetLevel}, 4); \quad \text{silence}(1000); \\
\text{in}(\text{OSetLevel}, 5); & \quad \text{silence}(1000); \quad \text{in}(\text{OSetLevel}, 6); \quad \text{silence}(1000); \\
\text{in}(\text{OSetLevel}, 7); & \quad \text{silence}(1000); \quad \text{in}(\text{OSetLevel}, 8); \quad \text{silence}(1000); \\
\text{in}(\text{OSetLevel}, 9); & \quad \text{silence}(1000); \quad \text{in}(\text{OSetLevel}, 10); \\
\text{out}(\text{IRlease});
\end{align*}
\]
Test Purposes 2

**TP2:** Check that controller can enter location ‘DnPassive’:
E<> Dim.DnPassive

- If delay=1000
- **Shortest (and fastest) Test:**

```java
out(IGrasp);
silence(500);
in(OSetLevel,0);
out(IRelase);
```
Test Purposes 2

TP2: Check that controller can enter location ‘DnPassive’:
E<> Dim.DnPassive

• If delay=40?

• Shortest Test:
  out(IGrasp);
  silence(500);
  in(OSetLevel,0);
  out(IRelase);
  out(IGrasp);
  silence(500);

• Fastest Test:
  out(IGrasp);
  silence(500);
  in(OSetLevel,0);
  silence(40);
  in(OSetLevel,1);
  silence(40);
  in(OSetLevel,2);
  silence(40);
  in(OSetLevel,3);
  silence(40);
  in(OSetLevel,4);
  silence(40);
  in(OSetLevel,5);
  silence(40);
  in(OSetLevel,6);
  silence(40);
  in(OSetLevel,7);
  silence(40);
  in(OSetLevel,8);
  silence(40);
  in(OSetLevel,9);
  silence(40);
  in(OSetLevel,10);
  silence(40);
**Test Purposes 2**

**TP2**: Check that controller can enter location 'DnPassive':

E<> Dim.DnPassive

- If Wait=1500 and minDelay=400?

Ask a tool
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - Edge coverage,
  - Definition/use pair coverage

![Diagram of a control flow graph with nodes labeled as $l_1$, $l_2$, $l_3$, and $l_4$, and edges labeled with conditions such as $x := 0$, $x \geq 2$, $x < 2$, and $a$, $b$, and $c$.]
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - **Location coverage**,
  - Edge coverage,
  - Definition/use pair coverage
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location coverage,
  - **Edge coverage**,  
  - Definition/use pair coverage
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Location Coverage,
  - Edge Coverage,
  - **Definition/Use Pair Coverage**
Coverage Based Test Generation

- Multi purpose testing
- Cover measurement
- Examples:
  - Locations coverage,
  - Edge coverage,
  - Definition/use pair coverage
  - All Definition/Use pairs

Generated by min-cost reachability analysis of annotated graph
Location Coverage

- Test sequence traversing all locations
- Encoding:
  - Enumerate locations $l_0, \ldots, l_n$
  - Add an auxiliary variable $l_i$ for each location
  - Label each ingoing edge to location $i$ $l_i := \text{true}$
  - Mark initial visited $l_0 := \text{true}$
- Check: $\text{EF}( l_0=\text{true} \land \ldots \land l_n=\text{true} )$
Edge Coverage

- Test sequence traversing all edges
- Encoding:
  - Enumerate edges $e_0, \ldots, e_n$
  - Add auxiliary variable $e_i$ for each edge
  - Label each edge $e_i := \text{true}$
- Check: $\text{EF}( e_0 = \text{true} \land \ldots \land e_n = \text{true} )$
Fastest Edge Coverage

Cost=12600 ms
Mutants

• M1 incorrectly implements switch

synchronized public void handleTouch() {
    if(lightState==lightOff) {
        setLevel(oldLevel);
        lightState=lightOn;
    }
    else { //was missing
        if(lightState==lightOn){
            oldLevel=level;
            setLevel(0);
            lightState=lightOff;
        }
    }
}

• M2 incorrect additional delay in dimmer as if x:=0 was on ActiveUP ↔ ActiveDN transitions
## Outcome

<table>
<thead>
<tr>
<th>Description</th>
<th>Test#</th>
<th>M0</th>
<th>M1</th>
<th>M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxLevel</td>
<td>1</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Short ActiveDn</td>
<td>4</td>
<td>pass</td>
<td>pass</td>
<td>pass</td>
</tr>
<tr>
<td>Resume</td>
<td>5</td>
<td>Pass</td>
<td>Fail</td>
<td>pass</td>
</tr>
<tr>
<td>Edge Cov.</td>
<td>3</td>
<td>Pass</td>
<td>Fail</td>
<td>Fail</td>
</tr>
</tbody>
</table>
Online Testing Of Real-Time Systems
Automated Model Based Conformance Testing

Does the behavior of the (blackbox) implementation comply to that of the specification?
Online Testing

- Test generated and executed event-by-event (randomly)
- A.K.A on-the-fly testing
On-The-Fly Testing

Torx tool

explorer

primer

driver

adapter

IUT

specification

implementation

[Jan Tretmans].
Our Framework

- **UppAal Timed Automata** Network: Env || IUT

- Complete and sound algorithm
- Efficient symbolic reachability algorithms
- **UppAal-TRON**: Testing Real-Time Systems Online
Online Testing

Symbolic state set:
\{<k_0l_0, 0 \leq x \leq 0>\}

EnvOutput: \{coin\}

EnvInput: \emptyset

ImpOutput: \emptyset

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

Symbolic state set:
\[\{(k_0l_0, 0 \leq x \leq 0)\}\]

EnvOutput: \{\text{coin}\}

EnvInput: \{\}

ImpOutput: \{\}

Let's offer input choose (the only) "coin"
Online Testing

Testing–UPPAAL

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

EnvOutput: \{ req \}

EnvInput: \emptyset

ImpOutput: \emptyset

Update the state set and other variables
Online Testing

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
Online Testing

Testing-UPPAAL

Symbolic state set:
\{ (k_1l_1, 5 \leq x \leq 5) \}

EnvOutput: \{ req \}

EnvInput: \emptyset

ImpOutput: \emptyset

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

Symbolic state set:
\{⟨k_1l_1, 5 \leq x \leq 5⟩\}

EnvOutput: \{req\}

EnvInput: ∅

ImpOutput: ∅
Online Testing

Testing–UPPAAL

Symbolic state set:
\{<k_2l_2, 0 \leq x \leq 0>, <k_2l_3, 0 \leq x \leq 0>\}

EnvOutput: \emptyset

EnvInput: \{weakCoffee, strongCoffee\}

ImpOutput: \{weakCoffee, strongCoffee\}

Update the state set
and other variables
Online Testing

Testing–UPPAAL

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

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

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

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

EnvOutput: \( \emptyset \)

EnvInput: \{strongCoffee\}

ImpOutput: \{strongCoffee\}

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

Symbolic state set:
\{ (k2l3, 4 \leq x \leq 4) \}

EnvOutput: \emptyset

EnvInput: \{ strongCoffee \}

ImpOutput: \{ strongCoffee \}

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

Symbolic state set:
\{ (k_2 l_3, 4 \leq x \leq 4) \}

EnvOutput: \emptyset

EnvInput: \{ strongCoffee \}

ImpOutput: \{ strongCoffee \}

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

Testing–UPPAAL

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

EnvOutput: \emptyset
EnvInput: \{ strongCoffee \}
ImpOutput: \{ strongCoffee \}

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

Testing-UPPAAL

Symbolic state set:
\( \emptyset \)

EnvOutput: \( \emptyset \)

EnvInput: \( \emptyset \)

ImpOutput: \( \emptyset \)

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

User Supplied Test Specification

- Closed TA Network partitioned into Env and IUT.
  - IUT model weakly input input enabled
  - Model of Environment
- Designate observable input and output actions.
- Specify amount of real time per one time-unit in model.
Online Algorithm

**Algorithm** \( \text{TestGenExec} \ (\text{TestSpec}) \) returns \{**pass**, **fail**, **inconclusive**\)

\( S:=\{\langle l_0,0 \rangle \} \), continueTesting:= true

**While** continueTesting **do** either

1. \( i:=\text{ChooseAction}(\text{EnvOutput}(S)) \) // Offer an input
   
   **send** \( i \) to SUT
   
   \( Z:=\text{After}(S,i) \)

2. \( \delta:=\text{chooseDelay}(S) \) // Delay and wait for output
   
   **Wait**(\( \delta \))
   
   **if** \( o \) occurred after \( \delta' \leq \delta \) **then**
   
   \( S:=\text{After}(S,\delta') \)
   
   1. **if** \( o \notin \text{ImpOutput}(S) \) **then** return **fail**
   
   2. **if** \( o \notin \text{EnvInput}(S) \) **then** return **inconclusive**
   
   \( S:=\text{After}(S,o) \)

   **else** // no output within \( \delta \) time
   
   \( S:=\text{After}(S,\delta) \)
   
   **if** \( S=\varnothing \) **then** return **fail**

3. continueTesting := false // terminate

**return pass**
T-UppAal: implementation

Graphical User Interface (Java)
- editor
- simulator
- verifier

Uppaal Engine Server (C++)
- Parsing
- Communication
- Control

Zones & Reachability, Etc

State-set explorer
Online Test Generation
Driver

Adapter
System Under Test

Simulator API
Test Setup

T-UPPAAL
On-the-fly
Testing
Host

tcp/ip

Test Interface

LightControllerGUI

setLevel
grasp
release
setLevel
grasp
release

LightController

JavaVM+w2k/Linux

mousePress
mouseRelease
Danfoss EKC Case
Electronic Cooling Controller

Sensor Input
- air temperature sensor
- defrost temperature sensor
- (door open sensor)

Keypad Input
- 2 buttons (~40 user settable parameters)

Output Relays
- compressor relay
- defrost relay
- alarm relay
- (fan relay)

Display Output
- alarm / error indication
- mode indication
- current calculated temperature

- Optional real-time clock or LON network module
Industrial Cooling Plants
Basic Refrigeration Control

- **setpoint**
  - + differential
  - differential

- **setpoint**
  - lowAlarm Deviation

- **highAlarm Limit**
  - highAlarm Deviation

- **lowAlarm Limit**

- **Time**
  - start compressor
  - stop compressor
  - normal
  - min restart time not elapsed
  - min cooling time not elapsed
  - alarm delay
EKC Adaptation 1

- Read and write parameter “database”
- 47 parameters

EKC Software Layering

- Control Software
- Parameter DB (shared variables)
- Device drivers + kernel
- Hardware + Physical I/O

Test Interface

- LON ➔ GW ➔ RS232
- AK-Online (PC SW)
  - configuration
  - supervision
  - logging

win32 + OLE + VB
EKC Adaptation 2

Need better test interface!
- Read-only parameters
- Delay and synchronization

```
22.3 0 1
```

```
16.7 0 0
```

```
22.1 0 1
```

```
"par#4=20.0"
```

```
TRON Engine
```

```
comressorOn  setTemp(20)
```

```
Adaptor
```

```
win32+OLE+VB  Solaris/Linux (C++)
```
Temperature Tracking

"periodic" weighted average:

\[ T_n = \frac{T_{n-1} \times 4 + T_{\text{sampled}}}{5} \]

- EKC calculated temperature
- Model calculated temperature
- Error/uncertainty envelope

Tolerance in sampling time
Tolerance in value computation

compressorOn!
Main Model Components

- 18 concurrent timed automata
- 14 clocks, 14 integers
Reverse Engineering

- Unclear and incomplete specifications
- Method of Working
  1. Formulate hypothesis model
  2. Test
  3. **FAIL**-verdict $\Rightarrow$ Refine model
  4. **(PASS)** $\Rightarrow$ Confirm with Danfoss
- Detects differences between actual and modeled behavior
- *Indicates promising error-detection capability*
- 4 examples
Ex1: Control Period

- Control actions issued when “calculatedTemp” crosses thresholds

  “periodic” weighted average: \[ T_n = \frac{T_{n-1} \times 4 + T_{\text{sampled}}}{5} \]

- No requirements on period given
- Tested to be 1.2 seconds
Clearing the alarm do not switch off alarm state, only alarm relay
Ex2: High Alarm Monitor v2

- Add HighAlarmDisplay action
- Add location for “noSound, but alarmDisplaying”
- (Postpone alarms after defrosting)
Ex3: Defrosting and Alarms

- When defrosting the temperature rises
- Postpone high temperature alarms during defrost
- System parameter alarmDelayAfterDefrost
- Several Interpretations
  1. Postpone alarmDelayAfterDefrost+alarmDelay after defrost?
  2. Postpone alarmDelayAfterDefrost+alarmDelay after highTemp detected?
  3. Postpone alarmDelayAfterDefrost until temperature becomes low; then use alarmDelay

- Option 3 applies!
Ex4: Defrost TimeTolerance

- Defrost relays engaged earlier and disengaged later than expected
- Assumed 2 seconds tolerance
- Defrosting takes long time
- Implementation uses a low resolution timer (10 seconds)
Example Test Run
(log visualization)