Towards a Runtime Verification Framework for the Ada Programming Language

André de Matos Pedro, David Pereira, Luís Miguel Pinho
CISTER & INESC-TEC, ISEP, Porto, Portugal
{anmap,dmrpe,lmp}@isep.ipp.pt

Jorge Sousa Pinto
Haslab & INESC-TEC, University of Minho, Portugal
jsp@di.uminho.pt

June 25, 2014

Talk Outline

Talk Outline

Motivation and Contextualization

The RMF4Ada Runtime Verification Framework
- Architecture and RML Structure
- Monitoring Modes
- Monitor Context-Switches

Contract Language Extension for Runtime Verification
- The RMTL-$\int$ Logic
- Timed Regular Expressions

Experimental Setup

Conclusions and Future Work

Motivation

Design of a Runtime Monitoring Framework and an extension of contracts for Ada programming language.

Contextualization

Main Ideas/Goals

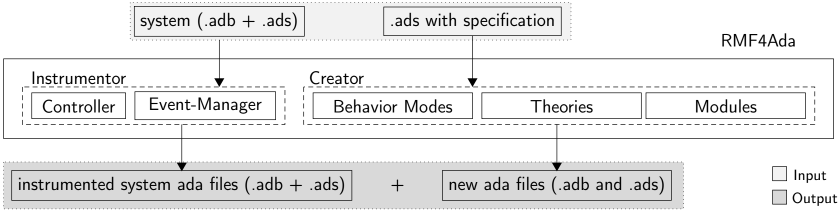
- to complement static analysis approaches, in particular when static verification leads to the state explosion;
- to deal with knowledge gathered at runtime (real data);
- to avoid defensing code;
- to shutdown a malfunction component and give the control to a small and simple formally verified component;
- to use a mathematical (logical) language with rigorous syntax to automatically synthesize monitors;
- to extend the Ada's contract language, allowing both static checking (within the classical constraints) as well as dynamic checking.

Talk Outline

RMF4Ada

"... It is not simply yet another RM framework, since the properties to be verified or enforced are generated from timed specifications written in the supported formal languages, in a correct-by-construction way..."

RMF4Ada Architecture



- RMF4Ada is composed by a set of Ada packages and two external tools (Intrumentor and Creator);
- some packages provide schemes for monitors (possibly executing in different patterns);
- other packages provide data structures to represent formal languages and the evaluation of their formulas/terms;

RML Structure

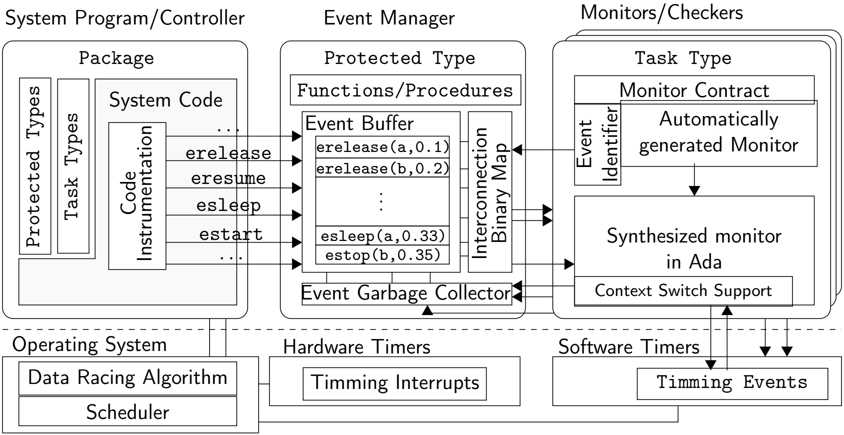
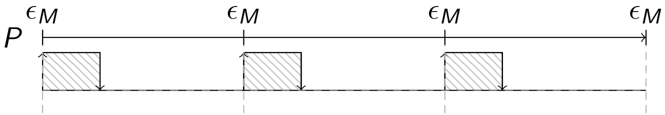


Figure: Illustration of the interconnection of the element blocks provided by the RML

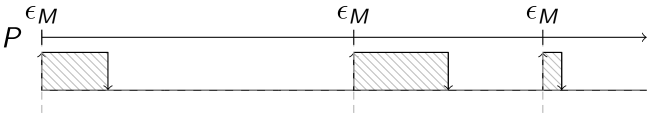
Monitoring Modes of RMF4Ada

Pre-defined behaviors

- Time-Triggered mode – executes as a periodic task;



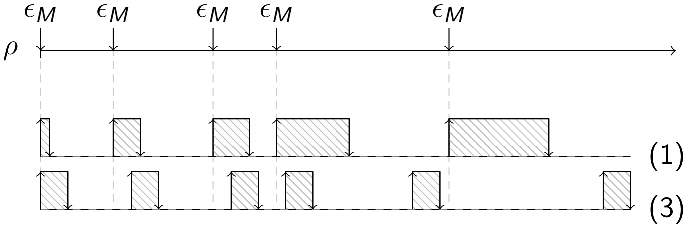
- Event-Based mode – executes as a sporadic task (each inter-arrival time shall be supplied before the execution and statically checked);



Monitor Context-Switches provided by RMF4Ada

Pre-defined conditions

1. step-bounded condition – the execution of the monitor ends when n iterations have been processed or when events have not arrived;
2. time-bounded condition – the execution of the monitor is bounded by t time units, exiting if no events occur;
3. symbol-based condition – the execution ends when one or more symbols of the path are consumed, and the monitor sleeps until a new symbol arrives;



Talk Outline

Contract Language Extension Overview

Weaknesses of the current contract language

- unable to support runtime verification of non-functional properties such as explicit time formulas or temporal logic;
- inexpressive for specification of monitoring behaviors;

Our Proposal

- introduce some contracts of the form:
  - `Monitor_Mode =>` *Mode* — *Mode* should be `Time_Triggered` or `Event_Triggered`, and
  - `Monitor_Case =>` (*Theory*, *Formulas*) — (`RMTLD`, ϕ) or (`TRE`, α);
- use theories such as MTL-$\int$ (Restricted Metric Temporal Logic) and TRE (Timed Regular expressions);

The Language of RMTL-$\int$

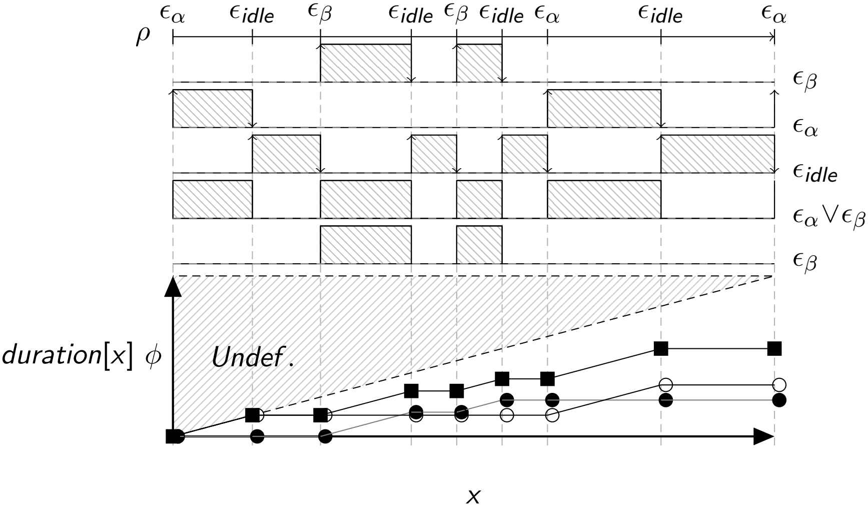
Syntax of RMTL-$\int$ terms and formulas

- Terms:
  $\delta \ ::= \ c \mid x \mid duration[\delta]\phi$
- Formulas:
  $\phi \ ::= \ p \mid \delta_1 \ op \ \delta_2 \mid \phi_1 \ or \ \phi_2 \mid not \ \phi \mid \phi_1 \ U[\gamma] \, \phi_2 \mid \phi_1 \ S[\gamma] \, \phi_2 \mid$ $exists \ x \, \phi$, with $c \in \mathbb{R}$, $x \in \mathcal{V}$, $p \in \mathcal{P}$, $op \in \{=, <, \leq\}$, and $\gamma \in \mathbb{R}_0^+$.

Common Abbreviations

Eventually	:	$eventually[\gamma] \ \phi$	$\equiv$	$p \ or \ not \ p \ U[\gamma] \ \phi$
Always	:	$always[\gamma] \ \phi$	$\equiv$	$not \ (eventually[\gamma] \ not \ \phi)$
Next	:	$next[\phi_1] \ \phi_2$	$\equiv$	$\phi_1 \ U[\infty] \ \phi_2$
Implies Next	:	$\phi_1 \ next \ implies \ \phi_2$	$\equiv$	$not \ \phi_1 \ or \ next[\phi_1] \ \phi_2$

Graphical interpretation over duration terms



- ρ is a path; ϵ_β, ϵ_α, and ϵ_{idle} are events;
- $\phi = \epsilon_\beta$ (—●—), $\phi = \epsilon_\alpha$ (—○—), and $\phi = \epsilon_\beta \vee \epsilon_\alpha$ (—■—)

The Language of Timed Regular Expressions

Syntax of TRE expressions

$$\alpha \ ::= \ 0 \,|\, 1 \,|\, a \in \Sigma \,|\, \alpha + \alpha \,|\, \alpha\alpha \,|\, \alpha^\star \,|\, \langle\alpha\rangle_I,$$

with Σ a the set of all events, and I a time interval of the form $[a..b]$ with $a, b \in \mathbb{R}_0^+$.

Talk Outline

Enforcement of Timing properties - Property 1

Statement

- the properties (1) and (2) have been synthesized manually using evaluation algorithms (future step for Creator tool);
- properties have been manually instrumented in a Mine Drainage Simulator using Ada packages provided by RMF4Ada (future step for Intrumentor tool);

```
task type T_Simulation (period: integer; deadline:
    integer)
  with
  Monitor_Mode => Event_Triggered,
  Monitor_Case => ( RMTLD ,
       T_Simulation'Event(Task_Release) next implies
       duration[T_Simulation'Time(period)]
       T_Simulation'Event(ANY) < T_Simulation'Time(wcet)
     );
```

Enforcement of Timing properties - Property 2

```
protected type Protected_Environment
with
    Monitor_Mode => Time_Triggered
    Monitor_Case => ( TRE,
      ( Protected_Environment.read_CH4'Event(pre) .
        <(Protected_Environment.read_Air_Flow'Event(ANY)
            + Protected_Environment.read_WaterPipe_Flow'
            Event(ANY))*>[0..20] .
        Protected_Environment.read_CO2'Event(post))*
      ) ,
is
    function read_CO2 return CO2_Level_State;
    function read_CH4 return CH4_Level_State;
    function read_Air_Flow return Air_Exhaust_State;
    function read_WaterPipe_Flow return
        WaterPipe_Flow_State;
end;
```

Talk Outline

Main Conclusions I

Positive Aspects

- enables the instrumentation of Ada programs with monitors that enforce RV behavior;
- introduces a small extension to the current Ada contract language for enabling the specification of contracts to be checked at runtime by monitors;
- introduces the monitoring synthesis of duration formulas from a formal language;
- automatically synthesizing a monitor from a formal language reduces the introduction of errors.

Main Conclusions II

Negative Aspects

- heavy-weight syntax for some simple WCET detections (using execution time timers);
- provides verification only for past executions (incomplete)
- who watches the watchdog?

Future Research Directions

Next Steps

- provide Creator and Instrumentor tools;
- combine monitor modes with prior analysis (where we should use event-based mode instead of time-triggered mode);
- optimize monitor context-switches;
- explore further formal systems;
- research adequacy of RMF4Ada for multi-core environments and for COTS as internal black-box components.

The End...

Thank you for watching our presentation.
Please send any comment to anmap@isep.ipp.pt.