Run-Time Validation of Timing Constraints for VDM-RT Models

Augusto Ribeiro    Kenneth Lausdahl    Peter G. Larsen
{ari,kel,pgl}@iha.dk

Aarhus School of Engineering – Denmark

October 24, 2011
Introduction

VDM-RT

Motivation

Timing Invariants

Invariant Types

Invariant Instances and Decommissioning Policies

Invariants Definition

Case Study and Results

Concluding remarks
The Real-Time dialect is an extension to VDM++ where object oriented structure and concurrency was introduced.

**Real-Time extensions adds**

- Simulated real time; all instructions take time.
- Hardware architecture definition; CPUs and buses.
- Distribution; objects are deployed onto CPUs connected with buses.
Time in VDM-RT

- Statements take time to execute
Time in VDM-RT

- Statements take time to execute
- `duration`

**Duration example**

```plaintext
public AdjustVolumeUp : () ==> ()
AdjustVolumeUp () ==
  duration (10)
  (if volume < MAX then ( volume := volume + 1;
    RadNavSys`mmi.UpdateScreen(1));)
```

Ribeiro, Lausdahl, Larsen

Timing Invariants in VDM-RT
Time in VDM-RT

- Statements take time to execute
- duration

Duration example

```java
public AdjustVolumeUp : () => ()
AdjustVolumeUp () ==
duration (10)
(if volume < MAX then ( volume := volume + 1;
RadNavSys`mmi.UpdateScreen(1)));
```

- Duration is typically an estimation
Log is produced when a VDM-RT model is interpreted.

```plaintext
1 CPDdecl -> id: 1 cpname: "RadNavSys" name: "CPU1" time: 0
2 CPDdecl -> id: 2 cpname: "RadNavSys" name: "CPU2" time: 0
3 CPDdecl -> id: 3 cpname: "RadNavSys" name: "CPU3" time: 0
4 BUSdecl -> id: 1 topo: (1,2,3) name: "BUS1" time: 0
5 DeployObj -> objref: 0 clname: "RadNavSys" cpumn: 0 time: 0
6 DeployObj -> objref: 1 clname: "MMI" cpumn: 1 time: 0
7 DeployObj -> objref: 2 clname: "Radio" cpumn: 2 time: 0
8 DeployObj -> objref: 3 clname: "Navigation" cpumn: 3 time: 0
9 DeployObj -> objref: 10 clname: "World" cpumn: 0 time: 0
10 DeployObj -> objref: 2147483696 clname: "INIT" cpumn: 0 time: 0
11 ...
12 ...
13 OpRequest -> id: 6 cpname: "HandleKeyPress(nat)" objref: 1 clname: "MMI" cpumn: 0 async: true time: 10000000000000000
14 MessageRequest -> busid: 0 fromcpu: 0 tocpu: 1 magid: 1 clname: "HandleKeyPress(nat)" objref: 1 size: 3 time: 10000000000000000
15 MessageCompleted -> magid: 1 time: 10000000000000000
16 ThreadSwingIn -> id: 6 objref: 1 clname: "MMI" cpumn: 1 overhead: 0 time: 10000000000000000
23 MessageCompleted -> id: 6 cpname: "HandleKeyPress(nat)" objref: 1 clname: "MMI" cpumn: 1 true time: 10000000004545910
24 MessageCompleted -> magid: 1 time: 10000000004545910
25 ThreadSwapOut -> id: 6 objref: 1 clname: "MMI" cpumn: 1 overhead: 0 time: 10000000004545910
27 MessageCompleted -> magid: 2 time: 1000000004573688
28 MessageCompleted -> magid: 2 time: 1000000004573688
29 MessageCompleted -> magid: 2 time: 1000000004573688
00 Message <- id: 7 cpname: "Radio" objref: 2 clname: "Radio" cpumn: 2 time: 1000000004573688
```
Log is produced when a VDM-RT model is interpreted.
Overture tool to visualise RT logs - [Verhoef]
RT Log Viewer

- Overture tool to visualise RT logs - [Verhoef]

- Permits visually analysis of a model execution
Real-time systems not only have functional requirements
Motivation

- Real-time systems not only have functional requirements
- Timing requirements cannot actually be recorded in VDM-RT explicitly
Motivation

- Real-time systems not only have functional requirements
- Timing requirements cannot actually be recorded in VDM-RT explicitly
- Perform validation of these timing requirements
First Idea for Timing Invariants

The idea of timing invariants for VDM-RT was originally described in:

[Fitzgerald et al., 2007] Fitzgerald, Larsen, Tjell, Verhoef
Validation Support for Real-Time Embedded Systems in VDM++
First Idea for Timing Invariants

The idea of timing invariants for VDM-RT was originally described in:

[Fitzgerald et al., 2007] Fitzgerald, Larsen, Tjell, Verhoef
Validation Support for Real-Time Embedded Systems in VDM++

- Post analysis of the RTLog
First Idea for Timing Invariants

The idea of timing invariants for VDM-RT was originally described in:

[Fitzgerald et al., 2007] Fitzgerald, Larsen, Tjell, Verhoef

*Validation Support for Real-Time Embedded Systems in VDM++*


- Post analysis of the RTLog
- It only exists at specification level
Timing Invariants - Basic Ingredients

Basic Ingredients:

- Trigger Event - $T_t$
Timing Invariants - Basic Ingredients

Basic Ingredients:

- Trigger Event - $T_t$
- Ending Event - $T_e$

$T_t$ and $T_e$ represent specific events in time, with $t$ and $e$ denoting the associated time intervals.
Timing Invariants - Basic Ingredients

Basic Ingredients:

- Trigger Event - $T_t$
- Ending Event - $T_e$
- Interval - time
Timing Invariants - Basic Ingredients

Invariants are relations between Trigger Time ($T_t$), Ending Time ($T_e$) and the Interval ($i$).

Typically something like:

$T_e - T_t \subseteq i$

Ribeiro, Lausdahl, Larsen - Timing Invariants in VDM-RT
Timing Invariants - Basic Ingredients

- Invariants are relations between Trigger Time ($T_t$), Ending Time ($T_e$) and the Interval ($i$)
Timing Invariants - Basic Ingredients

- Invariants are relations between Trigger Time ($T_t$), Ending Time ($T_e$) and the Interval ($i$)
- Typically something like: $T_e - T_t \sqsubseteq i$
Timing Invariants - Deadline

Deadline - $T_e - T_t \leq i$
Timing Invariants - Separate

Separate - $T_e - T_t > i$
Timing Invariants - Separate

- Separate - $T_e - T_t > i$

- Separate required - ending event is demanded
## Invariant instances - lifecycle of the invariant

### Lifecycle of an Invariant

- **Trigger event occurs** - an invariant instance is activated
- **Ending event occurs** - instance is deactivated and evaluated

### Problem

Invariants can be triggered several times and instances of the same invariant coexist

### Definition of decommission policies of instances
Invariant instances - lifecycle of the invariant

Lifecycle of an Invariant

- Trigger event occurs - an invariant instance is activated
Invariant instances - lifecycle of the invariant

Lifecycle of an Invariant

- Trigger event occurs - an invariant instance is activated
- Ending event occurs - instance is deactivated and evaluated
Invariant instances - lifecycle of the invariant

**Lifecycle of an Invariant**
- Trigger event occurs - an invariant instance is activated
- Ending event occurs - instance is deactivated and evaluated

**Problem**
Invariant instances - lifecycle of the invariant

Lifecycle of an Invariant
- Trigger event occurs - an invariant instance is activated
- Ending event occurs - instance is deactivated and evaluated

Problem
- Invariants can be triggered several times and instances of the same invariant coexist
### Lifecycle of an Invariant

- **Trigger event occurs** - an invariant instance is activated
- **Ending event occurs** - instance is deactivated and evaluated

### Problem

- Invariants can be triggered several times and instances of the same invariant coexist
- Definition of decommission policies of instances
Decommissioning Policies - Non-selective

- Non-selective

Time
Decommissioning Policies - Non-selective

- Non-selective

\[ P_{\text{trigger'}} \]

\[ \text{Time} \]
Decommissioning Policies - Non-selective

- Non-selective

\[ P'_{\text{trigger}} \]
\[ P''_{\text{trigger}} \]

Time
Decommissioning Policies - Non-selective

- Non-selective

\[ P_{\text{trigger'}} \quad P_{\text{trigger''}} \quad P_{\text{ender}} \]

Time

Ribeiro, Lausdahl, Larsen

Timing Invariants in VDM-RT
Deacommissioning Policies - Non-selective

- Non-selective

\[ \text{P}_{\text{trigger}'} \quad \text{P}_{\text{trigger}''} \quad \text{P}_{\text{ender}} \]

\[ \text{Time} \]
Decommissioning Policies - Matching

\( P_{\text{trigger}'} \)

Time
Decommissioning Policies - Matching

Matching

\[ P_{\text{trigger}}' \]

\[ P_{\text{trigger}}'' \]
Decommissioning Policies - Matching

Matching

$P_{\text{trigger}}'$

$P_{\text{trigger}}''$

$P_{\text{ender}}'$

Time
Decommissioning Policies - Matching
Decommissioning Policies - Matching

\[ P_{\text{trigger}}', P_{\text{trigger}}'', P_{\text{ender}}', P_{\text{ender}}'' \]

Matching
Decommissioning Policies - Matching

Matching

\[ P_{\text{trigger}}', P_{\text{trigger}}'', P_{\text{ender}}', P_{\text{ender}}'' \]

Time
Decommissioning Policies - Others

Other decommissioning policies could be be:

- Matching thread
- Matching object
What can an event be? - Operation Events

- Operation Events

Request (#req)  
Finish (#req)  
Activate (#act)  

Ribeiro, Lausdahl, Larsen  
Timing Invariants in VDM-RT  
18/29
What can an event be? - Operation Events

- Operation Events

- Reference operation phases on classes or instances
What can an event be? - Predicate Events

- Predicates over instance variables
What can an event be? - Predicate Events

- Predicates over instance variables
- Predicates are evaluated when the variable changes
Timing Invariants - Syntax

**General Syntax**

```
property(trigger, ending, interval)
```

**Examples**

```
deadlineMet(
    ( #req(MMI`HandleKeyPressUp),
      RadNavSys`radio.volume < Radio`MAX
    ),
    #fin(MMI`AdjustVolumeUp),
    100 ms)
```
Introduction

VDM-RT

Motivation

Timing Invariants

Invariant Types

Invariant Instances and Decommissioning Policies

Invariants Definition

Case Study and Results

Concluding remarks
Case study - In car navigation radio

- Case appears in M. Verhoef PhD thesis
- Car radio with 3 CPUs
- Process Traffic Message Channel (TMC) and controls volume
Case study - Time Invariants

C1: A volume change must be reflected in the display within 35 ms.

\[
\text{deadlineMet} (\\text{fin}(\text{Radio} \text{\`AdjustVolumeUp}),\\text{fin}(\text{MMI} \text{\`UpdateScreen}),35 \text{ ms})
\]
Case study - Time Invariants

C1: A volume change must be reflected in the display within 35 ms.

`deadlineMet`

```plaintext`
#fin(Radio`AdjustVolumeUp),
#fin(MMI`UpdateScreen),
35 ms)
```

C2: The screen should be updated no more than once every 500 ms.

`separate`

```plaintext`
#fin(MMI`UpdateScreen),
#fin(MMI`UpdateScreen),
500 ms)
```
Case study - Time Invariants

C3: If the volume is to be adjusted upwards and it is not currently at the maximum, the audible change should occur within 100 ms.

\[
\text{deadlineMet}(
  ( \ #req(\text{MMI} \ '\text{HandleKeyPressUp}),
  \text{RadNavSys} \ '\text{radio.volume} < \text{Radio} \ '\text{MAX} \\
  ),
  \text{fin}(\text{MMI} \ '\text{AdjustVolumeUp}),
  100 \ \text{ms})
\]
Achieved Results
Timing Invariants
System Extension Suggestion

- Time invariants are system invariants
- They could possibly be added to the `system` class.
Timing Invariants

System Extension Suggestion

- Time invariants are system invariants
- They could possibly be added to the `system` class.

**Examples**

```plaintext
system Sys
...
timing invariants

deadlineMet(evTrigger1, evEnder1, 400 ms);
...
separate(evTrigger2, evEnder2, 1000 ms);
```
Agenda

1. Introduction
   - VDM-RT
   - Motivation

2. Timing Invariants
   - Invariant Types
   - Invariant Instances and Decommissioning Policies
   - Invariants Definition

3. Case Study and Results

4. Concluding remarks
Concluding remarks and future work

- **What we have done:**

- **Answered questions:**

- **Future work:**
Concluding remarks and future work

- What we have done:
  - Prototype of time invariant checking during run-time

- Answered questions:

- Future work:
Concluding remarks and future work

- **What we have done:**
  - Prototype of time invariant checking during run-time
  - Defined behavior semantics of the invariants
- **Answered questions:**

- **Future work:**
Concluding remarks and future work

- What we have done:
  - Prototype of time invariant checking during run-time
  - Defined behavior semantics of the invariants

- Answered questions:
  - Should time invariants be included in the system?

- Future work:
Concluding remarks and future work

- What we have done:
  - Prototype of time invariant checking during run-time
  - Defined behavior semantics of the invariants

- Answered questions:
  - Should time invariants be included in the system?
  - Are the defined properties enough?

- Future work:
What we have done:
- Prototype of time invariant checking during run-time
- Defined behavior semantics of the invariants

Answered questions:
- Should time invariants be included in the system?
- Are the defined properties enough?
- Should the policies of decommission be selected by the user?

Future work:
Concluding remarks and future work

- What we have done:
  - Prototype of time invariant checking during run-time
  - Defined behavior semantics of the invariants

- Answered questions:
  - Should time invariants be included in the system?
  - Are the defined properties enough?
  - Should the policies of decommission be selected by the user?
  - Should violations mean the interruption of the interpretation

- Future work:
Concluding remarks and future work

What we have done:
- Prototype of time invariant checking during run-time
- Defined behavior semantics of the invariants

Answered questions:
- Should time invariants be included in the system?
- Are the defined properties enough?
- Should the policies of decommission be selected by the user?
- Should violations mean the interruption of the interpretation

Future work:
- Including run-time time invariant checks in the Overture interpreter.
Questions?