Requests for Modification of periodic thread definitions and duration and cycles statements

Ken Pierce[1] and Kenneth Lausdahl[2]

[1] School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, United Kingdom
K.G.Pierce@ncl.ac.uk
[2] Aarhus School of Engineering, Dalgas Avenue 2
DK-8000 Aarhus C, Denmark
kel@iha.dk

1 Overview

This note describes two RMs (Requests for Modification to the Language Board) relating to VDM-RT that were submitted by Ken Pierce and Kenneth Lausdahl in March 2011. Both RMs are related. They originated while the authors were building real-time controller models for the DESTECS[3] project. The issue is that two key timing-related constructs in VDM-RT (periodic thread definitions and duration / cycles statements) *only* permit numeric literals to be used to define timing behaviour. This means that "magic numbers" must be hard coded into specifications. We suggest that this is overly restrictive and represents poor coding practice forced by the language, meaning that specifications are harder to read and maintain.

The proposals therefore suggest that these constructs should allow a wider range of expressions to be used instead, in order to increase flexibility and usability. We suggest that at the very least *values* (i.e. constants) should be permitted in addition to numeric literals. This is only a small change to the syntax and has no effect on the semantics. As a further step, these constructs could allow references to instance variables, which would allow more object-level flexibility. For example, this would allow instances of the same class to have different periodic thread behaviour. This would introduce semantic questions however, such as when references are read and if they can be modified during execution.

Clearly, depending on the choices made, the semantics may or may not be affected by the proposed changes. The authors therefore suggest that this choice be discussed with the community, however we are keen to see (at least) values and instance variables permitted in periodic thread definitions.

In the remainder of this note, Section 2 attempts to motivate the need for changes with a view from the DESTECS project, followed by more details of the RMs in Section 3 and 4.

[3] http://www.destecs.org/

2 Motivation for requests

Our main reason for requesting these changes comes from our attempts (within the DESTECS project) to build real-time controller models for use in co-simulation with continuous-time plant (or environment) models.

Parts of the controllers that we wish to model in DESTECS —such as low-level PID controllers— need to know the sample time (i.e. the thread period) in order to calculate the control output. In a model where other time-related calculations are required (such as calculation of discrete integrals), this information needs to be available in multiple places. Typically, one would follow good programming practice and define a constant (**`value`**) in order to ensure that the value used is correct in all places.

The necessity to hard code the thread period as a numeric literal permits the possibility that the actual thread period and the value that the controller uses to calculate control actions can differ. This typically results in (often wildly) incorrect simulation results. This situation can arise if the modeller changes one of the values and forgets to change the other. This is particularly easy if the model is being altered by someone new who didn't originally write it. Therefore we believe that allowing constants to be used in periodic thread definitions is entirely justified.

In the DESTECS project, we are also interested in design space exploration (DSE) using co-simulation. This is where a set of candidate designs are evaluated by comparing the results of co-simulation. The best design is chosen according to some parameters, for example, the design that meets the requirements at the cheapest projected cost of components.

To increase the ability to evaluate designs, we wish to introduce automation where possible. Essentially, we would like the ability to alter certain parameters of controller models, run simulations to gather results, then compare them. This is somewhat similar to the combinatorial testing offered by the Overture tool already.

Parameters should include the number of controllers, their loop period and their deployment architecture. In one case study (the ChessWay personal transporter [FLP+10]) for example, a small safety monitor runs on it's own CPU at a speed much higher than the main controller, in order to ensure the safety of the rider. We might wish to run multiple simulations with differing main controller and safety controller speeds, in order to find the most effective combination.

The hard-coded nature of periodic threads and duration/cycle statements makes this difficult, in addition to the class-level nature of periodic threads. Currently, the user must edit the model in between each simulation run (ensuring that they update the relevant values in all places in which they appear). This is far from convenient and can lead to errors. Permitting instance variables to be referenced by periodic thread definitions would allow threading behaviour to be specified through object constructors. This fits with the object-oriented nature of VDM-RT and more easily permits automation.

Object-level periodic threads would also help in another aim of the DESTECS project, which is to provide libraries of common components for building real-time controllers in VDM (to help users unfamiliar with VDM to begin building working examples quickly). We would like to do this by providing classes that can be instantiated as objects by users. It would be preferable to allow periodic behaviour to be config-

urable via a constructor, rather than requiring new users to begin using concepts such as inheritance straight away.

3 Expressions in periodic thread definitions (ID: 3220182)

The DESTECS project [BLV+10] focuses on co-simulation of discrete-event controllers written in VDM-RT with continuous-time models described in 20-sim [Bro97,Kle06]. The controller models that we wish to produce are typically real-time controllers that must perform an action at a regular interval. For example, reading sensors and producing control actions at a frequency of 1000Hz (or a period of 1 millisecond).

The best way to achieve this in VDM-RT is with a periodic thread definition:

```
thread
periodic(period,delay,jitter,offset)
```

The values of *period*, *delay*, *jitter* and *offset* can only be hard coded as numeric literals. Therefore to run a controller thread at 1000Hz (one thousand times per second, or 1 millisecond per cycle), the following definition could be used:

```
thread
periodic(1,0,0,0)
```

3.1 Extension 1: Using `values` (and simple calculations)

The following definition currently couldn't be used, however it is perhaps a more intuitive way to define the behaviour and better coding practice. Here a constant called `FREQUENCY` is used to set the frequency (in Hz), with the conversion to period (in milliseconds) handled in the periodic definition:

```
values
  FREQUENCY: nat1 = 1000

thread
  periodic(1000/FREQUENCY, 0, 0, 0)
```

3.2 Extension 2: Using `instance variables`

Note that threads also are class-level (as opposed to object-level). This means that each object instance of a periodic class must have the same periodic behaviour. In order to model two copies of a controller running at the same time but at *different* speeds (or perhaps more likely different jitter or delay), it is necessary to define a controller class without a thread and then create two subclasses that only describe the periodic behaviour, for example:

```
-- this object will run normally
class MyControllerA is subclass of MyController

thread
  periodic(1, 0, 0, 0)

end MyControllerA

-- this object is more jittery
class MyControllerB is subclass of MyController

thread
  periodic(1, 10, 0, 0)

end MyControllerB
```

By permitting instance variable expressions to appear in periodic definitions, periodic behaviour can become object-level and permit instances of the same class do have different thread behaviour. Consider this example:

```
class MyController

instance variables
  private frequency: nat1 := 1000

thread
  periodic(1000/frequency, 0, 0, 0)

end MyController
```

Here, the value of frequency can be set in the constructor. This makes practices such as automated testing possible, including automated exploration of alternative designs and deployments which is one of the aims of the DESTECS approach. This modification would also permit objects instantiated from libraries of classes to have their periodic behaviour configured through a constructor:

```
class LibController

instance variables
  -- default frequency
  private frequency: nat1 := 1000

operations
  public LibController: nat1 ==> LibController
```

```
  LibController(freq) ==
    -- user-configurable frequency
    frequency := freq

thread
  periodic(1000/frequency, 0, 0, 0)

end LibController
```

A clear issue is when the value of frequency is evaluated and whether or not it can be changed. In order to ensure that a periodic thread's behaviour is unchanging over the course of an execution (i.e. by not allowing frequency to be assigned during run-time), we suggest that something like a **`final`** keyword (or equivalent) is introduced. In the Java language [GJSB05], a final variable can be assigned to *at most once* during construction of an object and must be assigned to during object construction. If this concept were adopted into VDM-RT, then periodic definitions could be restricted only to permit instance variables that are declared final:

```
instance variables
  private final frequency;
```

4 Values in duration / cycles statements (ID: 3220223)

The issue with duration and cycles is very similar to that of periodic threads described above. These two statements delay the internal clock of VDM-RT to simulate actions taking time. This delay can be based on the speed of the (simulated) CPU, using **`cycles`**, or based on (simulated) time, using **`duration`**. The following assignment statements would therefore take 10 (simulated) clock cycles to complete:

```
cycles(10) ( x := 1; y := 2; z := true )
```

Or the statement could be modified to take 2 (simulated) milliseconds, regardless of the (simulated) CPU speed:

```
duration(2) ( x := 1; y := 2; z := true )
```

As with periodic threads however, only numeric literals are permitted for describing the number of cycles or duration. Therefore the following is not valid:

```
values
  CYCLES: nat = 7
```

```
operations
  public op1: () ==> ()
  op1() ==
    cycles(CYCLES) (
      x := 1; y := 2; z := true
    )
```

So again magic numbers must be hard coded into specifications and durations / cycles statements cannot be altered through object-construction (they are static by class). We therefore request that expressions of time in duration and cycles statements not be restricted to numeric literals. Again there is a question of how flexible we want the language to be (values, instance variables, or functions, etc.) and when and how often the expressions should be evaluated (i.e. once during object construction or every time the statement is executed). We therefore suggest that the community should discuss these issues together and modify the RMs accordingly.

Note there are also related RMs to the two described here, namely 3220437: Extend duration and cycles (allow intervals + probabilities) and 3220324: Sporadic thread definitions.

Acknowledgements

The author's work is supported by the EU FP7 project DESTECS.

References

[BLV+10] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and Wouters F. Design support and tooling for dependable embedded control software. In *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*. ACM, April 2010.

[Bro97] Jan F. Broenink. Modelling, Simulation and Analysis with 20-Sim. *Journal A Special Issue CACSD*, 38(3):22–25, 1997.

[FLP+10] John Fitzgerald, Peter Gorm Larsen, Ken Pierce, Marcel Verhoef, and Sune Wolff. Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems. In D. Méry and S. Merz, editors, *IFM 2010, Integrated Formal Methods*, volume 6396 of *Lecture Notes in Computer Science*, pages 12–26. Springer-Verlag, October 2010.

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification, Third Edition*. Addison-Wesley Longman, Amsterdam, 3 edition, June 2005.

[Kle06] Christian Kleijn. Modelling and Simulation of Fluid Power Systems with 20-sim. *International Journal of Fluid Power*, 7(3), November 2006.