**Overture – Open-source Tools for Formal Modelling TR-2010-05**
**February 2010**

# Necessary adjustments of the Overture/VDM interpreter

by

P.G. Larsen, IHA
S. Wolff, Terma/IHA
K. Lausdahl, IHA
M. Verhoef, Chess
N. Battle, Fujutsu Services



# Contents

# Document History

| Revision | Notes | Date |
|---|---|---|
| 0.1 | Initial version | 02.2010 |

# 1 Introduction

This small memo aims to provide an overview of the adjustments to the Overture/VDM interpreter in order for the multi-threaded and multi-processor interpretation to work appropriately and subsequently pave the way for the co-simulation with 20-sim in the DESTECS project.

# 2 Main issues that needs addressing

The current version of the Overture/VDM interpreter have a number of issues which definitely needs to be addressed to make it useful for modeling and validation of multi-threaded VDM++ and VDM-RT models. The main points are:

1. Whenever a multi-threaded VDM model is present, the interpreter make use of multiple Java threads resulting in a non-deterministic execution of such models where errors envisaged can be hard to reproduce.

2. Whenever the interpretation is stopped with a breakpoint in the interpretation of a multi-threaded VDM model only the thread reaching the breakpoint is stopped (or threads that are created in a break state). This means that other threads continue their execution. This is not the way this should work since these other threads will be able to advance much longer than envisaged and it is impossible to inspect the exact position of the thread execution and inspect the value of the instance variables and local variables.

3. The permission predicates are only reevaluated periodically (in relation to the real time of an interpretation) and not whenever its value have potentially changed due to the dependencies of the history counters and the instance variables adjusted. This means that windows of opportunities where the permission predicates can enable the blocked threads to continue are not exploited.

4. Only explicit duration and cycle statements will move the simulated time. This severely hinders the realistic expectations one can have to the model execution in relation to an execution of a subsequent implementation. Thus, default durations needs to be introduced for all parts of the language and these will need to be taken into account.

# 3 Main issues that would be nice to address

In addition to the issues that needs to be solved there are a number of issues which have been discussed at different points of time and are desirable in the longer term to enable as well. The list of these issues include:

1. Handling of static instance variables in instances of classes that are deployed to multiple cpus are not done appropriately right now (nor in VDMTools). In essence, if such an instance

variable is updated in one cpu it is instantly also available at the other cpus where it is deployed. This is naturally cheating compared to a real world. This may require a language change so this needs to be clarified and a proposal will have to be made for the LB when this is done.

2. At the moment the deployment of instances of classes happen statically in the system class constructor (also in VDMTools). When one wish to use the VDM-RT dialect to describe a system where the different instances actually physically move and change their deployment dynamically this is not appropriate. A new MSc student will explore whether it is possible to incorporate concepts for dynamic allocation (for example inspired from the CREDO project) in a VDM-RT context. This again will probably require language changes so it needs to be approved by the LB but it would be advantageous to be able to conduct the research for such an extension before deciding upon whether it shall be permanently incorporated into Overture or not.

3. It would be desirable to be able to use duration intervals instead of simply a fixed amount, and then be able to run the execution in different modes (smallest number, random but controlled by seed or largest numbers).

4. At the moment both Overture/VDM and VDMTools have a single virtual cpu containing all the environment functionality. It would be desirable to explore the possibility of defining multiple virtual cpus for the system class.

5. Incorporation of predicates of events happening over time (either dynamically or with post-analysis after the execution based on a logfile) would be desirable. Whenever such timed predicates are broken the user should easily be able to determine the cause of the violation (either if stopped immediately when it was broken or graphically in visualizations of log-files). Another option would be to incorporate a special keyword: *maxDuration* for use in post conditions, to describe the maximum allowed duration of the given operation.

# 4 Suggested Solutions

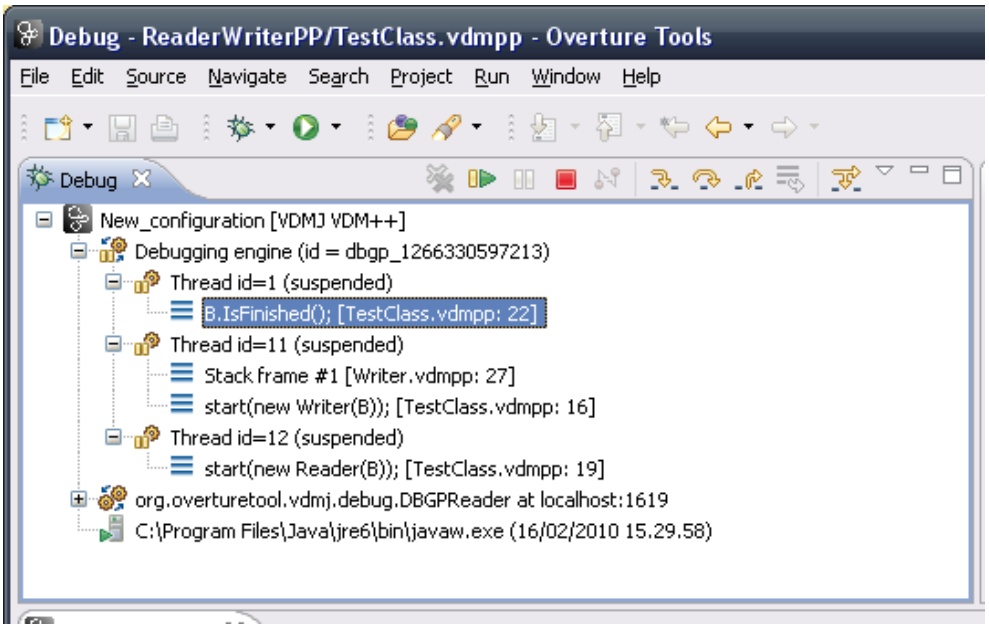
It is suggested to keep the use of Java threads in the Overture/VDM interpreter but have a master thread which controls all threads. This master will then act as the overall scheduler in case multi-threaded VDM models are to be interpreted. Thus there will at any point of time just be one thread executing and thus the nondeterminism arising from the Java virtual machine will be eliminated. In addition, the problem about stopping threads at breakpoints will disappear. Concerning the reevaluation of permission predicates, additions will need to be inserted in the Overture/VDM interpreter for this. Essentially this boils down to a static analysis of all the permission predicates where dependency information is derived at initialisation time. Subsequently, this information is to be used by the scheduler whenever any of the things dependent upon for a block permission predicate is updated.

Talking about the VDM-RT interpreter, the master thread will need to carry out a round-robin investigation of what functionality each resource (cpu and bus) can execute before updating the time. Once all have been processed the smallest time step will be chosen and all threads will be moved forward with this amount of time. For the treads that have a longer duration this means that the remaining duration will be decremented accordingly. For threads that have durations equal to the minimum duration the transactions performed in that local thread will be committed and made available to all other threads. This is also the place where the co-simulation will be incorporated (before the cpus are asked to progress time with the smallest time step, the continuous-time model will be asked to progress this amount of time. Either the continuous-time model will increase time by the negotiated amount and values will be exchanged between the two models, or an event happen at an earlier time and then all cpus are asked to progress by that amount).

# 5 Handling of Breakpoints

Definitions:

- Resume
- Suspend
- Terminate
- Step Into
- Step Over
- Step Return



When debugging and a breakpoint is reached the thread where the breakpoint is reached should be marked and the source location of the breakpoint should be shown.

- When a breakpoint is reached all threads shall stop before the next eval.

When the user selects:

- Terminate: The model is killed. The root process should die.
- Resume: The current thread and all other is resumed and will only stop again it the Suspend button is pressed or a breakpoint is reached.
- Step:
  - All other threads than the selected one is resumed. The one selected is allowed to eval one time of setp over. Eventually the other threads will stop because of synchronization constraints. New periodic threads must not be allowed to start before the debugging thread is stepped enough time steps to allow a new scheduling of a new periodic thread.