Overture – Open-source Tools for Formal Modelling TR-2009-02
December 2009

User Guide for the Overture VDM Tool Support

by

Peter Gorm Larsen, Kenneth Lausdahl, Augusto Reibero and Sune Wolff
Engineering College of Aarhus
Dalgas Avenue 2, DK-8000 Århus C, Denmark
Nick Battle
Fujutsu Services

Contents

ABSTRACT

This document is a user manual for the Overture Integrated Development Environment (IDE) open source tool for VDM. It can serve as a reference for anybody wishing to make use of this tool with one of the VDM dialects (VDM-SL, VDM++ and VDM-RT). This tool support is build of top of the Eclipse platform. The objective of the Overture open source initiative is to enable a platform that both can be used for experimentation of new subsets or supersets of VDM dialects as well as new features analysing such VDM models in different ways. The tool is entirely open source so anybody can join the development team and influenece the future developments. The long term target is also to ensure that stable versions of the tool suite can be used for large scale industrial applications of the VDM technology.

1 Introduction

The Vienna Development Method (VDM)is one of the longest established model-oriented formal methods for the development of computer-based systems and software [Bjørner&78a, Jones90, Fitzgerald&08a]. It consists of a group of mathematically well-founded languages for expressing system models during early design stages, before expensive implementation commitments are made. The construction and analysis of the model using Overture help to identify areas of incompleteness or ambiguity in informal system specifications, and provide some level of confidence that a valid implementation will have key properties, especially those of safety or security. VDM has a strong record of industrial application, in many cases by practitioners who are not specialists in the underlying formalism or logic [Larsen&95, Clement&99, Kurita&09]. Experience with the method suggests that the effort expended on formal modeling and analysis can be recovered in reduced rework costs arising from design errors.

VDM models are expressed in a specification language (VDM-SL) that supports the description of data and functionality [ISOVDM96, Fitzgerald&98, Fitzgerald&09]. Data are defined by means of types built using constructors that define structured data and collections such as sets, sequences and mappings from basic values such as Booleans and numbers. These types are very abstract, allowing the user to add any relevant constraints as data type invariants. Functionality is defined in terms of operations over these data types. Operations can be defined implicitly by preconditions and postconditions that characterize their behavior, or explicitly by means of specific algorithms. An extension of VDM-SL, called VDM++, supports object-oriented structuring of models and permits direct modeling of concurrency [Fitzgerald&05]. An additional extension to VDM++ is called VDM Real Time (VDM-RT) (formerly called VDM In a Constrained Environment (VICE)) [Mukherjee&00, Verhoef&06]. All these different dialects are supported by the unified tool called Overture.

Since the VDM modeling languages have a formal mathematical semantics, a wide range of analyses can be performed on models, both to check internal consistency and to confirm that models have emergent properties. Analyses may be performed by inspection, static analysis, testing or mathematical proof. To assist in this process, Overture supply tool support for building models in collaboration with other modeling tools, to execute and test models and to carry out different forms of static analysis [Larsen&10]. It can be seen as an open source version of the commercial tool called VDMTools [Elmstrøm&94, Fitzgerald&08b] although that also have features to generate executable code in high-level programming languages which are not yet available in Overture.

This guide explains how to use the Overture IDE for developing models for different VDM dialects. This user manual starts with explanantion about how to get hold of the software in Ssection 2. This is followed in Section 3 with an introduction to the concepts used in the different Overture perspectives based on Eclipse terminology. In Section 4 it is explained how projects are managed in the Overture IDE. In Section 5 the features supported when editing VDM models are explained. This is followed in Section 6 with an explainantion of the interpretation and debugging capabilities in the Overture IDE. Section 7 then illustates how test coverage information can be gathered when models are interpreted. Afterwards Section 8 shows how models with and without test coverage information can be generated to the text processing system LaTeX and automatically

converted to `pdf` format if one have `pdflatex` installed on the computer. Afterwards from Section 9 to Section 13 different VDM specfic features are explained. In Section 9 the use of the notion for proof obligations and its support in Overture is explained. In Section 10 a notion of combinatorial testing and the automation support for that in Overture is presented. In Section 11 support for mapping between object-oriented VDM models to and from UML models is presented. In Section 12 it is illustrated how one can move from a VDM++ project to a new VDM-RT project. In Section 13 it is shown how support to analysing and displaying logs from executing such VDM-RT models. After these sections the main part of the user manual is completed in Section 14 with an explanantion of the features from Overture that also is available from a command-line interface. Finally in Appendix 15 an index of significant terms used in this user manual can be found.

2 Getting Hold of the Software

The Overture project is managed on SourceForge. The best way to run Overture is to download a special version of Eclipse with the Overture functionality already pre-installed. If you go to:

`http://sourceforge.net/projects/overture/files/`

you can find pre-installed versions of Overture for Windows, Linux and Mac. At a later stage it will also be possible to use an update site to install it from directly in Eclipse. However, at the moment only stand-alone versions are distributed because the risk of version problems and dependencies with other plug-ins is much smaller this way.

Zip files with a large collection of existing VDM-SL, VDM++ and VDM-RT projects can be downloaded from

`http://sourceforge.net/projects/overture/files/Examples/`

Such existing projects can be imported as described in subsection 4.1.

3 Using the Overture Perspective

3.1 Getting into the Eclipse Terminology

Eclipse is an open source platform based around a *workbench* that provides a common look and feel to a large collection of extension products. Thus if a user is familiar with one Eclipse product, it will generally be easy to start using a different product on the same workbench. The Eclipse workbench consists of several panels known as *views*, such as the Script Explorer view at the top left of Figure 1. A collection of panels is called a *perspective*, for example Figure 1 shows the standard Overture perspective. This consists of a set of views for managing Overture projects and viewing and editing files in a project. Different perspectives are available in Overture as will be described later, but for the moment think about a perspective as a useful composition of views for conducting a particular task.

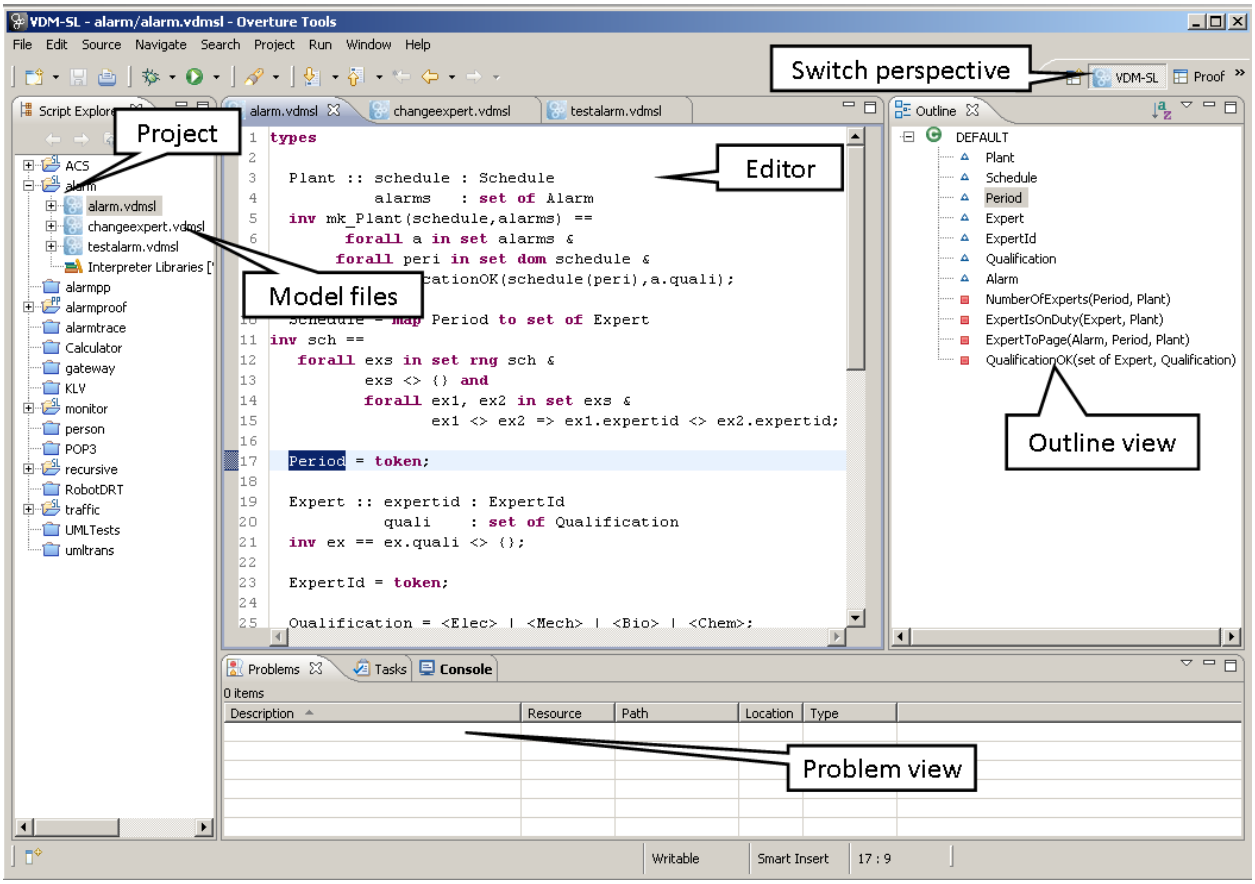


Figure 1: The Overture Perspective

The *Script Explorer view* lets you create, select, and delete Overture projects and navigate between the files in these projects.

Depending upon the dialect of VDM used in a given project, a corresponding Overture editor will be available here. A new VDM-SL project is created choosing the *File* → *New* → *Project*. Then Figure 2 will appear and *Next* can be used and then a name needs to be given to the project.

The *Outline view*, to the right of the editor (see Figure 3), presents an outline of the file selected in the editor. The outline displays any declared VDM-SL modules, as well as their state components, values, types, functions and operations. In case of a flat VDM-SL model the module is called `DEFAULT`. Figure 1 shows the outline view on the right hand side. Clicking on an operation or function will move the cursor in the editor to the definition of the operation. At the top of the outline view there is a button to optionally sort what is displayed in the outline view, for instance it is possible to hide variables.

The *Problems view* gathers information messages about the projects you are working on. This includes information generated by Overture, such as warnings and errors.

Most of the other features of the workbench, such as the menus and toolbars, are similar to

Figure 2: Creating a New VDM-SL Project

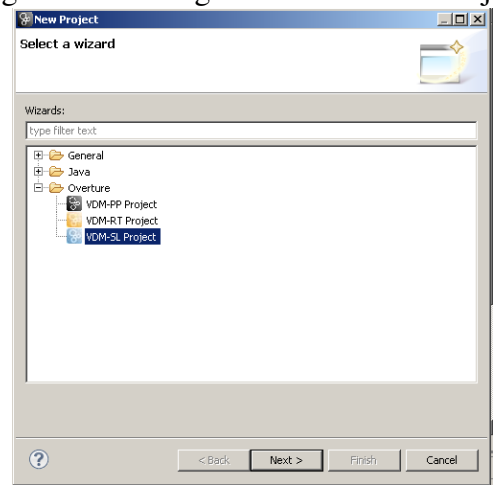
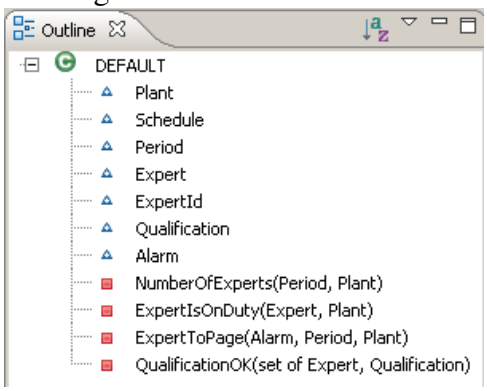


Figure 3: The Outline View



other Eclipse applications, though note that there is a special menu with Overture specific functionality. One convenient feature is a toolbar of shortcuts to switch between different perspectives that appears on the right side of the screen; these vary dynamically according to context and history.

3.2 Additional Eclipse Features Applicabe in Overture

4 Managing Overture Projects

4.1 Importing Overture Projects

4.2 Creating a New Overture Project

1. Create a new project by choosing *File* → *New* → *Project* → *Overture*;

2. Select the VDM dialect you wish to use (VDM-SL, VDM-PP or VDM-RT);
3. Type in a project name
4. Chose whether you would like the contents of the new project to be in your workspace or outside from existing source files and
5. click the finish button (see 4).

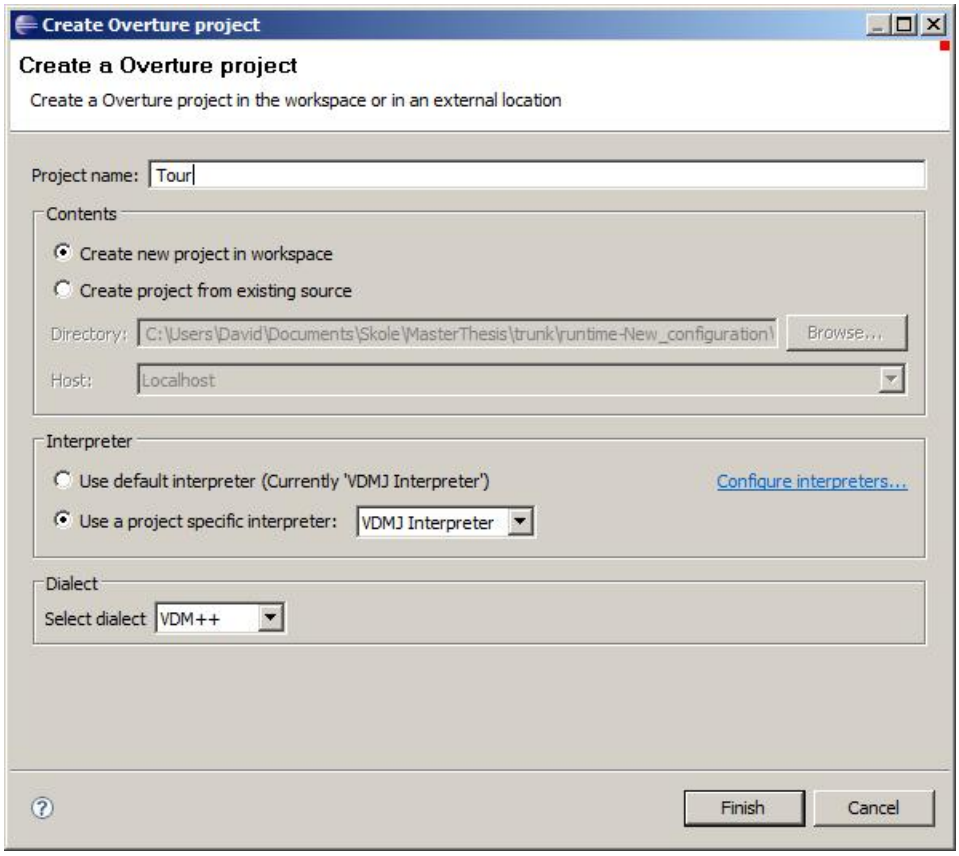


Figure 4: Create Project Wizard

4.3 Creating Files

Switching to the Overture perspective will change the layout of the user interface to focus on the VDM development. To change perspective go to the menu window → open perspective → other. . . and choose the Overture perspective. When the developer is in the Overture Perspective the user can create files using one of the following methods:

1. Choose *File* → *New* → *VDM-SL Module* or *VDM-PP Class* or *VDM-RT Class* or
2. Right click on the Overture project where you would like to add a new file and then choose *New* → → *VDM-SL Module* or *VDM-PP Class* or *VDM-RT Class*.

In both cases one needs to choose a file name and optionally choose a directory if one does not want to place the file in the directory for the chosen Overture project. Then a new file with the appropriate file extension according to the chosen dialect will be created in the selected directory. This file will use the appropriate module/class template to get the user started with defining the module/class meant to be placed in this new file. Naturally keywords for kinds of definitions that will not be used can be deleted.

4.4 Setting Project Options

5 Editing VDM models

6 Interpretation and Debugging in Overture

This section describes how to debug a model using the Overture IDE.

6.1 Debug configuration

Debugging the model under development is done by creating a debug configuration from the menu *Run* → *Debug configuration* . . . The debug configuration dialog requires the following information as input to start the debugger: the project name, the class, the starting operation/function and the file containing the starting operation/function. Figure 5 shows a debug configuration, clicking one of the browse buttons will open a dialog which give the user a list of choices. The class and operation/function are chosen from the dialog with the list of expandable classes, if the operation or function have arguments these must be typed in manually.

6.2 Debug Perspective

The Debug Perspective contains the views needed for debugging in VDM. Breakpoints can easily be set at desired places in the model, by double clicking in left margin. When the debugger reaches the location of the breakpoint, the user can inspect the values of different identifiers and step through the VDM model line by line.

The debug perspective shows the VDM model in an editor as the one used in the Overture Perspective, but in this perspective there are also views useful during debugging. The features provided in the debug perspective are described below. The Debug Perspective is illustrated on figure 6

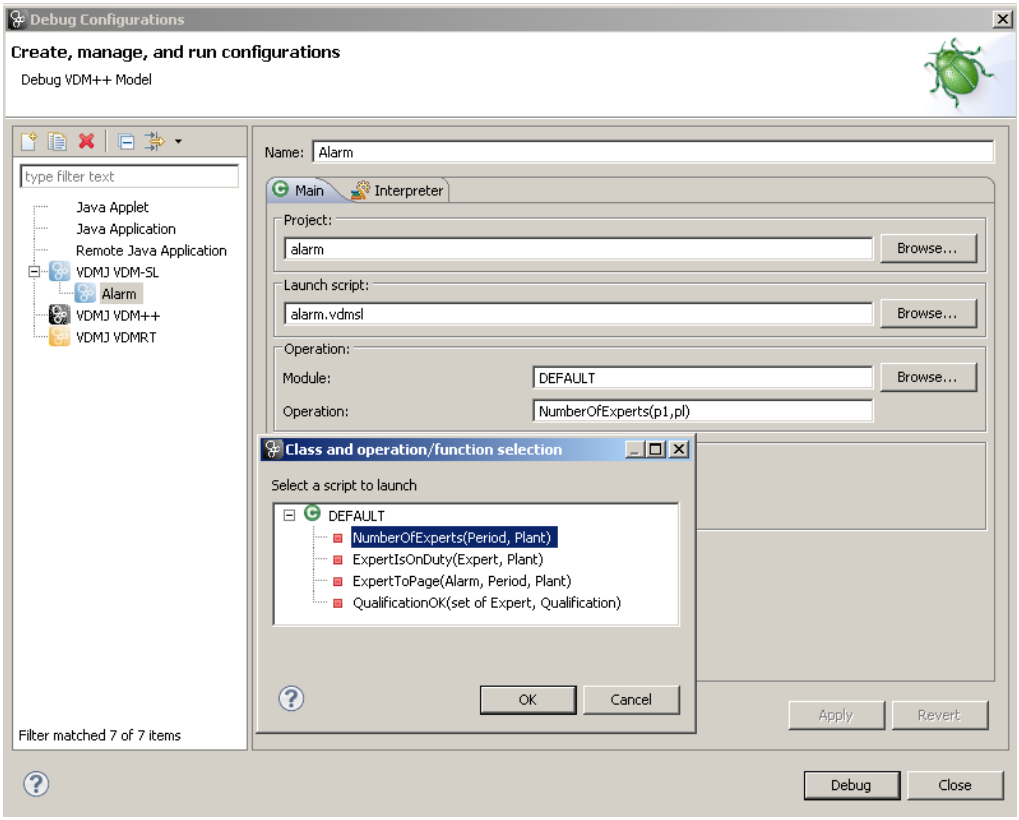


Figure 5: The debug configuration dialog

6.2.1 Debug View

The debug View is located in the upper left corner in the Debug Perspective - see figure 6. The debug view shows all running models and the call stack belonging to them. It also displays whether a given model is stopped, suspended or running. In the top of the view buttons for debugging such as; stop, step into, step over, resume, etc. are located. All threads are also shown, along with their running status. It is possible to switch between threads from the Debug View.

6.2.2 Variables View

This view shows all the variables in a given context, when a breakpoint is reached. The variables and their values displayed are automatically updated when stepping through a model. The variables view is by default located in the upper right hand corner in the Debug Perspective. It is also possible to inspect complex variables, expanding nested arrays and so forth.

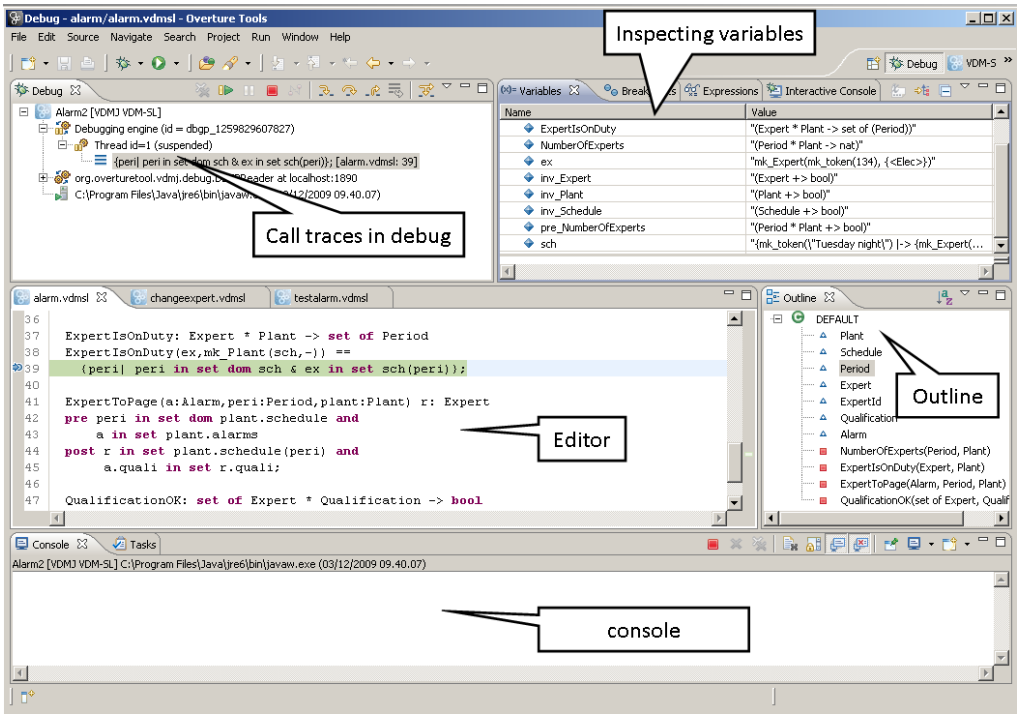


Figure 6: Debugging perspective

6.2.3 Breakpoints View

Breakpoints can be added both from the edit perspective and the debug perspective from the editor view. In the debug perspective however, there is a breakpoints view that shows all breakpoints. From the breakpoints view the user can easily navigate to the location of a given breakpoint, disable, delete or set the hit count or a break condition. In figure 6 the Breakpoints View is hidden behind the Variables View in the upper right hand corner in a tabbed notebook. Section 6.2.6 explains how to use conditional breakpoints.

6.2.4 Expressions View

The expressions view allows the user to write expressions, as for the variables view, the expressions are automatically updated when stepping. Watch expressions can be added manually or created by selecting 'create watch expression' from the variables view. It is of course possible to edit existing expressions. Like the Breakpoints View this view is hidden in the upper right hand corner.

6.2.5 Interactive Console View

While the Expressions View allows to easily inspect values, the functionality is somewhat limited compared with the functionality provided by VDMTools. For more thorough inspections the Interactive Console View is more suited. Here commands can be executed on the given context, i.e.

where the debugger is at a breakpoint. The Interactive console keeps a command history, so that already executed commands can be run again without actually typing in the command all over. Figure 7 shows the interactive console.

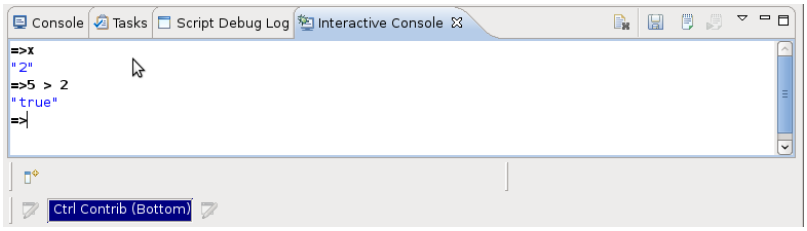


Figure 7: The interactive console

6.2.6 Conditional breakpoints

Conditional breakpoints can also be defined. These are a powerful tool for the developer since it allows specifying a condition for one or more variables which has to be true in order for the debugger to stop at the given breakpoint. Apart from specifying a break condition depending on variables, a hit count can also be defined. A conditional breakpoint with a hit count lets the user specify a given number of calls to a particular place at which the debugger should break.

Making a breakpoint conditional is done by right clicking on the breakpoint mark in the left margin and select the option Breakpoint properties... This opens a dialog like the one shown in figure 8. It is possible to choose between two different conditional breakpoints, a hit count condition and one based on an expression defined by the user.

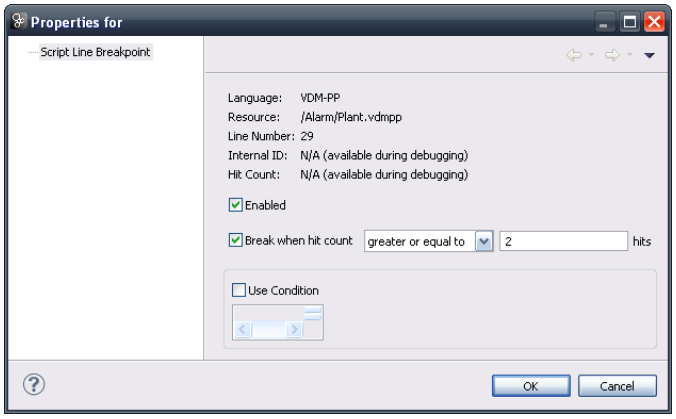


Figure 8: Conditional breakpoint options

7 Collecting Test Coverage Information

8 Pretty Printing to LaTeX

9 Managing Proof Obligations

10 Combinatorial Testing

11 Mapping VDM++ back and forth to UML

12 Moving from VDM++ to VDM-RT

13 Analysing and Displaying Logs from VDM-RT Executions

14 A Command-Line Interface to VDMJ

References

[Bjørner&78a] D. Bjørner and C.B. Jones, editors. *The Vienna Development Method: The Meta-Language*. Volume 61 of *Lecture Notes in Computer Science*, Springer-Verlag, 1978.

This was the first monograph on *Meta-IV*. See also entries: [Bjørner78b], [Bjørner78c], [Lucas78], [Jones78a], [Jones78b], [Henhapl&78]

[Bjørner78b] D. Bjørner. Programming in the Meta-Language: A Tutorial. *The Vienna Development Method: The Meta-Language*, 24–217, 1978.

An informal introduction to *Meta-IV*

[Bjørner78c] D. Bjørner. Software Abstraction Principles: Tutorial Examples of an Operating System Command Language Specification and a PL/I-like On-Condition Language Definition. *The Vienna Development Method: The Meta-Language*, 337–374, 1978.

Exemplifies so called **exit** semantics uses of *Meta-IV* to slightly non-trivial examples.

[Clement&99] Tim Clement and Ian Cottam and Peter Froome and Claire Jones. The Development of a Commercial "Shrink-Wrapped Application" to Safety Integrity Level 2: the DUST-EXPERT Story. In *Safecomp'99*, Springer Verlag, Toulouse, France, September 1999. LNCS 1698, ISBN 3-540-66488-2.

[CTManual, 09] User Manual for the Overture Combinatorial Testing Plug-in. 2009.

[Elmstrøm&94] René Elmstrøm and Peter Gorm Larsen and Poul Bøgh Lassen. The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications. *ACM Sigplan Notices*, 29(9):77–80, September 1994. 4 pages.

[Fitzgerald&05] John Fitzgerald and Peter Gorm Larsen and Paul Mukherjee and Nico Plat and Marcel Verhoef. *Validated Designs for Object–oriented Systems*. Springer, New York, 2005.

[Fitzgerald&08a] J. S. Fitzgerald and P. G. Larsen and M. Verhoef. Vienna Development Method. *Wiley Encyclopedia of Computer Science and Engineering*, 2008. 11 pages. edited by Benjamin Wah, John Wiley & Sons, Inc.

[Fitzgerald&08b] John Fitzgerald and Peter Gorm Larsen and Shin Sahara. VDMTools: Advances in Support for Formal Modeling in VDM. *Sigplan Notices*, 43(2):3–11, February 2008. 8 pages.

[Fitzgerald&09] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, Second edition, 2009. ISBN 0-521-62348-0.

[Fitzgerald&98] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[Henhapl&78] W. Henhapl, C.B. Jones. A Formal Definition of ALGOL 60 as described in the 1975 modified Report. In *The Vienna Development Method: The Meta-Language*, pages 305–336, Springer-Verlag, 1978.

One of several examples of ALGOL 60 descriptions.

[ISOVDM96] Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language. December 1996.

[Jones78a] C.B. Jones. The Meta-Language: A Reference Manual. In *The Vienna Development Method: The Meta-Language*, pages 218–277, Springer-Verlag, 1978.

[Jones78b] C.B. Jones. The Vienna Development Method: Examples of Compiler Development. In Amirchachy and Neel, editors, *Le Point sur la Compilation*, INRIA Publ. Paris, 1979.

[Jones90] Cliff B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall International, Englewood Cliffs, New Jersey, second edition, 1990. 333 pages. ISBN 0-13-880733-7.

This book deals with the Vienna Development Method. The approach explains formal (functional) specifications and verified design with an emphasis on the study of proofs in the development process.

[Kurita&09] T. Kurita and Y. Nakatsugawa. The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip. *Intl. Journal of Software and Informatics*, 3(2-3), October 2009.

[Larsen&10] Peter Gorm Larsen and Nick Battle and Miguel Ferreira and John Fitzgerald and Kenneth Lausdahl and Marcel Verhoef. The Overture Initiative – Integrating Tools for VDM. *ACM Software Engineering Notes*, 35(1):, January 2010. 6 pages.

[Larsen&95] Peter Gorm Larsen and Bo Stig Hansen. Semantics for Underdetermined Expressions. *Accepted for "Formal Aspects of Computing"*, 7(??):??, January 1995. 14 pages.

[Lucas78] P. Lucas. On the Formalization of Programming Languages: Early History and Main Approaches. In *The Systematic Development of Compiling Algorithm*, INRIA Publ. Paris, 1978.

An historic overview of the (VDL and other) background for VDM.

[Mukherjee&00] Paul Mukherjee and Fabien Bousquet and Jérôme Delabre and Stephen Paynter and Peter Gorm Larsen. Exploring Timing Properties Using VDM++ on an Industrial Application. In J.C. Bicarregui and J.S. Fitzgerald, editors, *Proceedings of the Second VDM Workshop*, September 2000. Available at www.vdmportal.org.

[Verhoef&06] Marcel Verhoef and Peter Gorm Larsen and Jozef Hooman. Modeling and Validating Distributed Embedded Real-Time Systems with VDM++. In Jayadev Misra and Tobias Nipkow and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 147–162, Lecture Notes in Computer Science 4085, 2006.

15 Index