Counterpoint: Towards a Proof-Support Tool for VDM

Ken Pierce

School of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, United Kingdom.
K.G.Pierce@ncl.ac.uk

Abstract. This paper is a position paper that presents ideas for an extension to the Overture tool platform that will support the process of proof in the VDM family of formal languages. The intention of the paper is to garner interest in building this extension and to promote discussion of a development road map. While creation of formal specifications in VDM is currently supported by two robust tools —the commercial VDMTools and the open-source Overture tool— these tools focus on execution of specifications and automated testing. This new extension to Overture will focus on support for proof and act as both a counterpart and counterpoint to the existing tools. Hence the name of the extension will be *Counterpoint*. Counterpoint will extend the Overture tool, which is built using the Eclipse framework. It will support the management of discharging a set of proof obligations generated from VDM specifications. It will support both hand-crafted proofs in the natural deduction style and mechanization of proofs through external tools.

1 Introduction

VDM (Vienna Development Method) is a mature and widely-used formal method. It is model based, meaning that specifications in VDM are explicit models of the systems which they represent, with a central notion of state and operations. Operations can be defined both implicitly in terms of pre- and post-conditions and explicitly using constructs familiar to programmers.

VDM is supported by two robust tools: the commercial VDMTools[1] and the open-source Overture tool[2]. These tools can syntax check VDM specifications and include static type checkers. Both tools can also execute a subset of VDM specifications (those with explicit function definitions) and perform dynamic checking at run-time. Single runs can be initiated by the tool user, as well as automated execution of a large number of test cases.

The expressiveness of VDM means however that static type checking is in general undecidable [DHB91]. For example, the consistency of functions with pre-conditions and the satisfiability of implicit functions cannot be checked statically [Ber97]. Where static checks cannot be performed, proof obligations can be generated [Ber97]. These proof obligations must be discharged by hand in order to show that a model is consistent. The generation of these obligations is supported by both VDMTools and Overture, however further tool support for proof is minimal at best.

[1] http://www.vdmtools.jp/en/
[2] http://www.overturetool.org/

Central to the idea of the original VDM-SL[3] language is the notion of refinement [Jon90]. In refinement, abstract specifications are shown to be refined (or reified in VDM terminology) by more concrete specifications. The aim being to eventually reach a concrete specification that can be realised in a programming language, with (relative) correctness to the original abstract specification being preserved by the reification chain.

Typically, this process involves data reification (finding concrete representations for abstract data types such as sets) and operation decomposition (demonstrating that the behaviour of abstract operations is realised by one or more concrete operations) [Jon90]. The correctness of reification is demonstrated by proof.

The author of this paper was first introduced to VDM over seven years ago, during his undergraduate degree. This included a few lectures and a piece of coursework on proof. Later, the author's thesis [Pie09] addressed the correctness of Simpson's four-slot mechanism [Sim90] for asynchronous communication using VDM.

The author also collaborated with colleagues from Newcastle University in an effort to use VDM to verify the "Mondex" electronic purse challenge problem[4] of Grand Challenge 6 [Woo06,SCW00]. This work included the production and checking of a large number of proofs by hand. The team agreed that while it was an excellent learning exercise, undertaking the proof task by hand was a challenge they would not wish to repeat again soon!

The main point of this author's introduction is to assert that proof is (and always has been) central to world of VDM and that the push for industrial adoption and tool-support has drawn focus away from this key aspect. This paper is intended to be the next (albeit initially small) step on the road to redressing that balance. The end of this road will see many new features available in the Overture tool, which —importantly— should compliment the current feature set and be of great benefit to the VDM community. The author has chosen to name this extension to the Overture tool "*Counterpoint*".

While the choice of another musical term may seem a little tongue-in-cheek, the name should hopefully evoke the idea of two seemingly different parts in a musical piece working together to form a richer whole — as execution and testing can work together with proof in verification. A second (older) meaning to the term, that of a counter argument, evokes the notion of proof. Finally, the name is a simple English word that rolls off the tongue easily.

In the remainder of this paper, Section 2 proposes a set of features for Counterpoint, Section 3 reiterates the aims of the Overture initiative and finally Section 4 draws conclusions and provides a look at the way ahead.

2 Counterpoint: a Proof-Support Tool for VDM

Counterpoint will be an extension to the Overture tool. It will provide a *Proof* perspective to complement the current *VDM* (editing) and *Debug* (execution) perspectives. The main view (an element of a perspective in Eclipse terminology) provided by Counterpoint will be a proof obligation manager. The proof obligation manager will display a

[3] VDM++ and VDM-RT are object-oriented extensions of VDM-SL.

[4] A technical report describing this work is in preparation at the time of writing.

list of the proof obligations that must be discharged for the current project. Proof obligations that assert a model's consistency can already be generated from Overture and VDMTools [LLR+10,Ber97]. It is expected that the current underlying proof obligation generator in Overture will remain unchanged, with the graphical interface being superseded by Counterpoint's proof perspective.

For refinement proofs, it will be necessary to define a retrieve function (or designate a function from a specification) and to define the relationship between abstract and concrete operations. Counterpoint will generate proof obligations based on this information. This will likely require some changes to the underlying framework to allow management of multiple specifications and the relationships between them.

Each proof obligation can be discharged by associating it with a proof artifact. Counterpoint will support three types of proof artifact: *automated proof* results, generated by plug-ins and external tools; *natural deduction proofs*, constructed in the Counterpoint editor; and *other evidence*, for less formal proofs. These are explained in greater detail below.

Support for "as automated as possible" proofs (with fully automatic proving being the ultimate goal) is important if the aim is to have proof adopted in industry as a verification technique[5]. The author also believes that the act of crafting proofs and seeing the process is an excellent reason to undertake the task by hand. The author therefore believes that Counterpoint should be a platform in which automated proof and handcrafted proof can coexist, providing the user with the best choice of tools for *their* purpose.

The proof manager will give a visual mnemonic indicating the status of proof obligations. Red will indicate that a proof obligation is yet to be discharged and green indicates a proof obligation that has been discharged. A blue colour will be used for proof obligations that the user asserts to be true, but which cannot be checked automatically by the tool (i.e. other proof evidence). This approach is also taken by the Rodin tool for Event-B [Abr07], however Counterpoint will ensure that some form of evidence is associated with a 'blue' proof.

Full automation of proofs for VDM is unlikely (because of the need to find witness values for existential quantifications, for example), therefore some form of user-guided proof is likely to be necessary. It is key therefore that the various Counterpoint features give understandable and constructive feedback to the user (for both automated and hand crafted proofs).

The three types of proof artefact are now explained in greater detail.

Automated proof Counterpoint's support for automated proof will be based on plug-ins that link to external theorem provers. As noted in [LBF+10], there is currently no VDM-specific theorem prover. Off-the-shelf tools such as HOL [SN08] can however be used for VDM specifications, as seen in [Ver07,VHL10]. In the future of course, a VDM-specific theorem prover may be built and could be integrated into Counterpoint in the same way.

[5] It should be noted that industrial use of VDM has done very well using the existing tool support.

Automated proof artifacts in Counterpoint will contain information required to discharge the proof (e.g. version information, tactics used) as well as information produced by the external prover. Plug-ins should ensure that this feedback to the user is useful, particularly when an automated proof fails. Counterpoint will support the ability to perform a brute force attempt to discharge proof obligations and will automatically attempt to re-discharge proof obligations when specifications are changed.

On the issue of feedback from external proof tools, the author would recommend that feedback is mapped back into VDM syntax (as in PROPSER [AS99,DCN+00]), as opposed to being presented in prover-specific form (as in Vermolen's work [Ver07]). This will ensure that the user is not required to learn the syntax of external provers with which they may not be familiar. This should ensure a smoother, more integrated user experience, especially if (or hopefully, when) multiple prover plug-ins become available. Of course, this puts a greater burden on plug-in developers, especially during initial development or time-limited student projects. Therefore the author suggests a compromise is reached where mature plug-ins with VDM-syntax feedback are included in the official Overture releases, with experimental or early-development plug-ins available as optional extras for keen users.

The emphasis on the initial development of Counterpoint will be to provide extension points for plug-ins (and not on the creation of plug-ins themselves). Extension points will provide an interface between plug-ins and Counterpoint. This will in theory give plug-ins a consistent look and feel for feedback and prover configuration (such as supplying user-defined tactics) and allow plug-ins to access models through the Overture AST (Abstract Syntax Tree). Creation of these plug-ins will be an excellent source for student projects.

Natural deduction proofs The core of Counterpoint's natural deduction proof support will be an editor for crafting proofs. The editor will include automated line numbering to reduce the tedium of updating evolving proofs and will lay out proofs (including hypotheses, conclusions and justifications) in an intuitive way. An ASCII syntax will be defined for the editor, however a more complex file format than plain text may be needed for proofs (e.g. XML).

Counterpoint will support a number of useful tools based around this core proof editor. It will include a directory of theorems, initially taken from [BFL+94]. A view of this directory will be provided that allows users to easily browse and search for theorems, e.g. searching theorems by name, listing all theorems relating to sets, or finding a theorem with a specific conclusion.

In the course of producing proofs, users may create theorems that will be useful to others. During the Mondex work in VDM for example, much time was spent on a lemma which stated that the sum of the values in a set of natural numbers yields a natural number. This could be useful to others in future. To harness this process, Counterpoint will be linked to an online repository of theorems that will allow the 'theory base' [BFL+94] of VDM to expand through shared effort.

Counterpoint will also provide a proof checker, which will use the directory of theorems and the specification of the model in order to check the validity of proofs. The proof checker will check proofs as they are constructed, much like the syntax and type

checker of the VDM core of Overture. The proof checker will also be executed automatically if the specification changes.

Counterpoint will also provide pretty-printing support[6] through generation of LaTeX source using the VDM macros (which already support natural deduction proofs). Generation of a LaTeX sources / PDFs of the entire proof effort of all the proofs of a project will also be supported.

Finally, Counterpoint will offer the possibility to incorporate user-guided proof support, much like that the mural tool [JJLM91]. This support could in fact comprise a reimplementation of the mural engine; or integration of results from research projects such as AI4FM [GJ10]; or take on new ideas of keen students. Or better yet a combination of all of the above. The author believes this is a particularly rich vein for research.

Other evidence This category of proof artifact is designed for less formal proofs and recognizes that not all proofs will necessarily be taken to the fully formal level. These proofs could range from assertions of correctness based on inspection, through to semi-formal proofs and structured arguments such as those given in [Pie09]. Some proof obligations required for consistency are simple or trivial [Ber97]. This mode will support plain text, LaTeX source (including VDM macros), and PDF and image files (e.g. JPG, PNG) as proof artifacts. PDF files and images could be used where current proofs exist in other formats, such as early proofs from scanned manuscripts, for example.

As noted previously, Counterpoint could not automatically check this type of proof, so a blue colour will mark artifacts that the user asserts to be valid. The tool could however mark proofs that must be manually checked after the specification in a way that could affect the validity of those proofs. Because the proofs cannot be checked by Counterpoint, there is an onus on the user to be correct. Counterpoint will at least require that some information is provided as a proof artifact, which means that the user cannot simply dismiss a proof without providing some explanation. This will aid traceability and confidence in the proof effort.

Figure 1 shows a representation of the components of Counterpoint proposed above. There are three types of element in the diagram (based on line colour and fill colour). Shaded (green) elements are already complete, because they are part of the Overture tool already. The author suggests that initial development phase should be focused on the white elements with solid outlines, since these are achievable and form a foundation of the framework. The elements with dashed outlines require this basic framework to exist and likely require more significant research, therefore the author suggests that they be tackled in later phases of development.

A mockup of a possible *Proof* perspective for Overture is given in Figure 2. It shows a Proof explorer view (top-left) that lists the proof obligations of the current model. Each proof obligation has two icons, indicating its type (automated or hand-crafted) and status: red for unproved, green for proved and blue for marked as proved by the user. In the large editor window (right), a natural deduction proof is being constructed. In the workbench (bottom), a search of the 'theory base' is being performed.

[6] As an aside, the author would also be interested in a plug-in for pretty-printing and pretty-editing of VDM specifications (i.e. Math syntax) from inside the Overture tool.

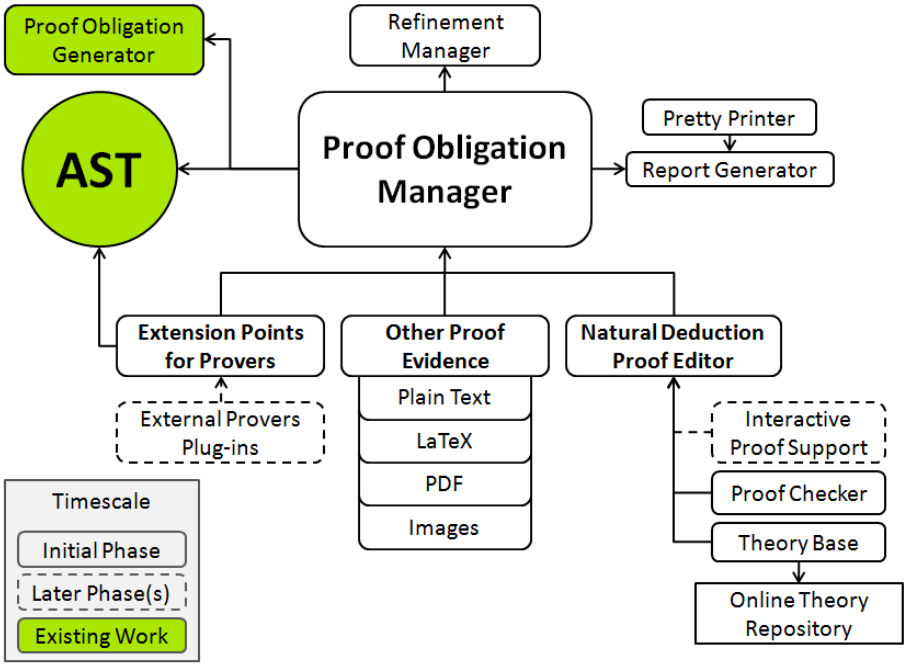

Fig. 1. Overview of proposed components for Counterpoint

3 Aims of the Overture Initiative

Because Counterpoint will be an extension to the Overture tool, the author wishes to reiterate the core values of the Overture initiative[7] (or now perhaps the Overture / Counterpoint initiative):

- to promote and enhance the use of VDM (and formal proof)
- to provide industrial strength tools
- to be open-source
- to be community-driven
- to provide opportunities for research
- to provide opportunities for teaching
- to be a fun project to be involved in!

4 Conclusions and The Way Forward

This position paper presented some ideas for an extension to the Overture tool platform that will support the process of proof in the VDM family of formal languages.

[7] `http://www.overturetool.org/?q=About`

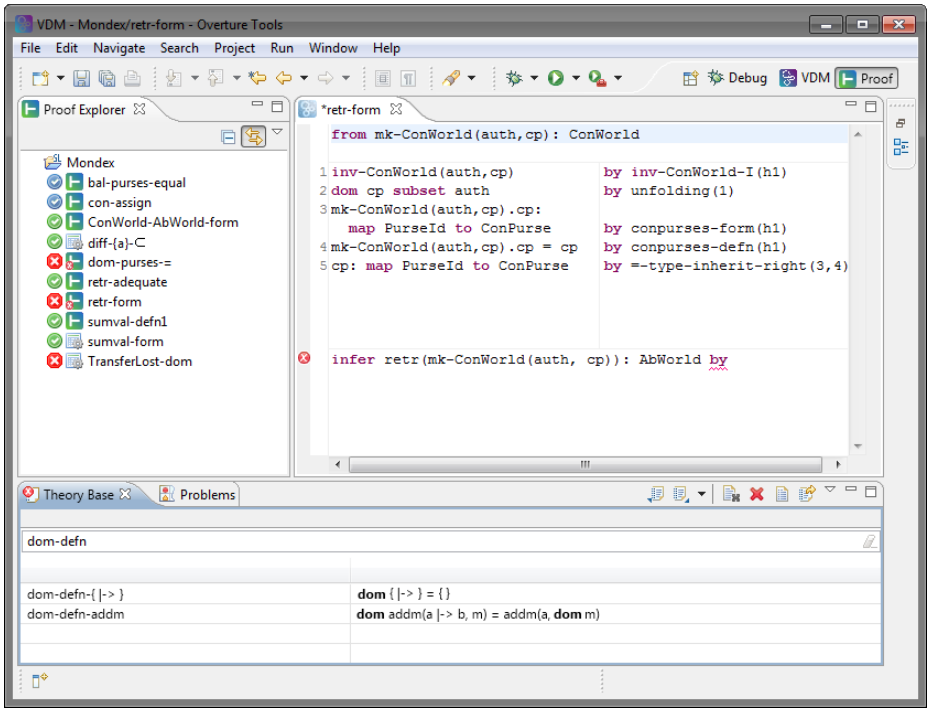

Fig. 2. A mockup of a Proof perspective for Overture

Tool-support for the creation, execution and testing of VDM specifications is currently strong, however tool-support for proof is lacking. The author of the paper hopes that together we can bring tool support for proof up to the current standard of Overture, to provide Overture and our users with an even richer set of features for the specification of systems with VDM.

Since this extension will not simply be a single plug-in, but a whole new perspective (both literally and figuratively) to the Overture tool, the author has chosen to name this extension Counterpoint. Counterpoint will provide support for managing proof obligations; for discharging proofs through external theorem provers; for hand crafting proofs in the natural deduction style; as well as a number of other features to aid the proof process.

The author did not consider model checking in this paper, since he sees it as orthogonal (though complementary) to proof, and that perhaps it would exist in some future *Model Checking* perspective. It would clearly be another excellent string to the Overture tool's bow however, so the author would of course welcome contributions on the topic.

The author hopes that this paper will encourage discussion within the Overture and VDM community. As a next step, the author suggests that the above ideas be formed into a concrete set of requirements, taking community feedback into account. These requirements can then be used to prioritize, plan and measure development effort.

As a rough plan, the author suggests that the creation of a *Proof* perspective and proof obligation manager would be a good first step. Initially, simple proof artifacts (i.e. LaTeX source, PDF files) will be supported. This will create a platform upon which the more complex automated and natural deduction proof support can be built. For natural deduction proofs, basic support will require a definition of a file format and creation of an editor, which should be a relatively simple step in this direction. Creation of a proof checker and user-guided proof support will require further research.

For interfacing with automated theorem provers, the key will be to define extension points that allow plug-ins to interface with Counterpoint and access the necessary model information through the Overture AST. It will be necessary to think carefully about how external theorem prover plug-ins will need to interact with the tool and models. The author suggests that integrating the work of [Ver07,VHL10] into the emerging Counterpoint framework would be a useful pilot project, from which requirements for extension points could be generalised.

Acknowledgements

The author wishes to thank his colleagues Jeremy Bryans, Richard Payne, and Zoe Andrews, who participated in a focus group after the Mondex work at Newcastle. The author also wishes to thank John Fitzgerald and Cliff Jones for discussions on the topic of proof tools, as well as the two reviewers who helped improve this paper. The author's work is supported by the EU FP7 project DESTECS.

References

[Abr07] Jean-Raymond Abrial. Rodin Tutorial. `http://deploy-eprints.ecs.soton.ac.uk/10/1/tutorials-2007-10-26.pdf` (Unpublished), 2007.

[AS99] S. Agerholm and K. Sunesen. Formalizing a Subset of VDM-SL in HOL. Technical report, IFAD, April 1999. http://www.vdmportal.org/twiki/pub/Main/VDMpublications/FormailizingVDM-SL.pdf.

[Ber97] Bernhard K. Aichernig and Peter Gorm Larsen. A Proof Obligation Generator for VDM-SL. In John S. Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, September 1997. ISBN 3-540-63533-5.

[BFL+94] Juan Bicarregui, John Fitzgerald, Peter Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practitioner's Guide*. FACIT. Springer-Verlag, 1994. ISBN 3-540-19813-X.

[DCN+00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard Boulton, Konrad Slind, Graham Robinson, Mike Gordon, and Tom Melham. The PROSPER Toolkit. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Germany, March/April 2000. Springer-Verlag, Lecture Notes in Computer Science volume 1785.

[DHB91] Flemming M. Damm, Bo Stig Hansen, and Hans Bruun. On type checking in vdm and related consistency issues. In *Proceedings of the 4th International Symposium of VDM Europe on Formal Software Development-Volume I: Conference Contributions - Volume I*, VDM '91, pages 45–62, London, UK, 1991. Springer-Verlag.

[GJ10] Gudmund Grov and Cliff B. Jones. Ai4fm: A new project seeking challenges! In Rajeev Joshi, Tiziana Margaria, Peter Mueller, David Naumann, and Hongseok Yang, editors, *VSTTE 2010*, August 2010.

[JJLM91] C. B. Jones, K. D. Jones, P. A. Lindsay, and R. Moore. *mural: A Formal Development Support System*. Springer-Verlag, 1991.

[Jon90] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall International, second edition, 1990.

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

[LLR+10] Peter Gorm Larsen, Kenneth Lausdahl, Augusto Ribeiro, Sune Wolff, and Nick Battle. Overture VDM-10 Tool Support: User Guide. Technical Report TR-2010-02, The Overture Initiative, www.overturetool.org, May 2010.

[Pie09] Ken G. Pierce. *Enhancing the Usability of Rely-Guarantee Conditions for Atomicty Refinement*. PhD thesis, Newcastle University, 2009.

[SCW00] Susan Stepney, David Cooper, and Jim Woodcock. An electronic purse: Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000.

[Sim90] H.R. Simpson. Four-slot fully asynchronous communication mechanism. *Computers and Digital Techniques, IEE Proceedings* -, 137(1):17–30, Jan 1990.

[SN08] Konrad Slind and Michael Norrish. A brief overview of hol4. In Otmane Mohamed, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer Berlin / Heidelberg, 2008.

[Ver07] S. D. Vermolen. Automatically Discharging VDM Proof Obligations using HOL. Master's thesis, Radboud University, Nijmegen, 2007. Draft.

[VHL10] Sander Vermolen, Jozef Hooman, and Peter Gorm Larsen. Automating Consistency Proofs of VDM++ Models using HOL. In *Proceedings of the 25th Symposium On Applied Computing (SAC 2010)*, Sierre, Switzerland, March 2010. ACM.

[Woo06] Jim Woodcock. Verified software grand challenge. In *FM*, pages 617–617, 2006.