

Triumphs and Challenges for Model-Oriented Formal Methods: the VDM⁺⁺ experience (Abstract)

John S. Fitzgerald*, Peter Gorm Larsen†

* School of Computing Science, Newcastle University, UK

Email: John.Fitzgerald@newcastle.ac.uk

† Engineering College of Aarhus, Denmark

Email: pgl@iha.dk

The term “lightweight formal methods” gained currency just over ten years ago, following the publication of a round-table article “An Invitation to Formal Methods” edited by Hussein Saiedian in the April issue of IEEE Computer. Two contributions came under the heading “Formal Methods Light”. In one article Jones [1] argued for a “rigorous” approach emphasising the systematic description of system state as an aid to early validation of models, rather than the fully formal development of a complete system. Jackson and Wing [2] argued that more emphasis should be placed on the tractability of formal specification languages than on their expressiveness; that specifications should cover significant parts of systems rather than aiming for comprehensiveness, and that partial analysis might be preferred to proof. Both articles, in different ways, were calling on the formal methods community to develop methods and tools that provide at least some of the benefits of formalism, without requiring the wholesale application of highly specialised technology.

A decade on, we might ask how the community has responded to the idea of lightweight formal methods, and how much has been achieved. In this paper, we draw on experience in developing the semantics and tool support for VDM, and in applying VDM technology in industry, to identify achievements and challenges in providing lightweight but effective formal methods.

I. VDM AND VDM⁺⁺

The Vienna Development Method (VDM) [3], [4] is one of the longest established and best known formal methods. Originating in compiler development and language design work at IBM’s Vienna Laboratories [5], it consists of a formal modelling language VDM-SL, standardised with a denotational semantics in 1996 [6], a proof theory [7] and a refinement theory [3]. Recent developments in VDM have included its extension to support object-oriented design and concurrency in VDM⁺⁺ [8] and providing a capability for modelling real-time distributed systems [9].

VDM is *model-oriented* – the formal language is used to construct a model of the system of interest, given in terms of data and functionality. Data is expressed in terms of base types such as numeric types, structureless tokens and enumerations. Structured types may be built from these basic ones using type constructors which include collections such as sets, sequences

and mappings. Distinguished state variables model persistent data if required. Invariants can be expressed for both types and over instance variables. Functionality is defined in terms of referentially transparent functions over the defined data types, or operations over the state variables. Abstraction is provided in data by the unconstrained character of the basic types and type constructors: for example, there is no maximum integer, and sets have unconstrained, though finite, cardinality. Both functions and operations may be specified implicitly in terms of preconditions and postconditions, or explicitly in terms of expressions and statements.

The typed Logic of Partial Functions (LPF), described in [7], underpins VDM. However, so far only the MURAL tool [10] has provided a relatively pure implementation of LPF for VDM specifications.

The VDM community has generally been reluctant to prescribe a methodology. Indeed, the technology has been used in a very flexible way in research activity, notably in the development of approaches to concurrency, as well as in industrial practice. However, we concentrate here on the latter strand. The aim of the work considered here has been to raise the level of abstraction and precision in industrial software development.

Experience in applying VDM in industry projects convinced us that successful, cost-effective adoption requires a lightweight approach. A comparative study of developments with and without formal techniques [11] encouraged us to believe that a lightweight model-oriented formalism could be embedded successfully in an industrial development process under certain conditions. Three of these conditions were particularly significant for subsequent work:

- 1) *Modelling*: The formalism should be treated as a precise modelling language: a tool for use at levels of abstraction determined by the user. We should not try to force the developer into using a particular (e.g. refinement-based) methodology.
- 2) *Accessibility*: The formalism should be presented in an accessible way; tools should link to existing tool support and not require training in new platforms.
- 3) *Automation*: Tool support should be powerful but lightweight; always favouring automation over comprehensiveness.

We have tried to realise these conditions using VDM and its

tool support. Our first attempts to do this led to the approach described in [4] and later extended in [8].

II. TOOL SUPPORT FOR VDM AND VDM⁺⁺

VDM's recent industrial application has been closely tied to its tool support. Early tools, such as Adelard's SpecBox [12], were largely confined to basic static checking and pretty-printing of specifications. However, alongside the development of the denotational semantics given in the ISO VDM-SL Standard, Elmström, Larsen and Lassen developed a toolset that implemented an operational semantics [13], [14]. In accordance with lightweight principles, the product that ultimately resulted from this work, VDMTools, was primarily aimed at cost-effective industrial use rather than expressive completeness.

VDMTools is currently under active development and use, the technology having passed from IFAD to the Japanese corporation CSK¹. The tools support syntax and type checking for the full object-oriented VDM⁺⁺ language, and contain an interpreter for test-based analysis of specifications written within an executable subset. Testing is further supported by coverage analysis tools. The interpreter has a dynamic link library feature allowing external (non-VDM) code to be incorporated. Automatic code generation to C++ and Java are also supported. However, although proof obligations can be automatically generated, VDMTools do not yet contain proof support. Experimental evaluation of HOL-based discharging of proof obligations from VDM models suggests that, for certain models, around 90% of obligations can be discharged automatically [15].

III. INDUSTRIAL EXPERIENCES

There have been many industry-based applications of VDM⁺⁺ in the ten years since the term "Formal Methods Light" was coined. Here we give brief details of three of the more large scale or significant ones that we are allowed to discuss. In each case, we relate the application back to the three conditions for successful application that we identified above. In all three cases, formal modelling is only part of a larger development process and only part of the application is modelled formally at all.

A. DustExpert (1995-97)

Adelard² was awarded a UK government contract to develop a safety-related knowledge-based system to advise on the construction of industrial plants that contain potentially explosive dusts. Initial requirements were expressed as a standard from the UK Health and Safety Executive some 450 pages long. A VDM model (12kLOC, excluding comments) of the tool was used as the basis for construction of hand proofs of safety properties, test-based analysis using the VDMTools interpreter and test coverage tool. Alongside statistically significant testing, these contributed directly to the system's safety case. The implementation was 16kLOC of Prolog, plus 18kLOC of

C++ graphical user interface (excluding comments). Adelard claimed that the use of VDMTools contributed to productivity and fault density far better than industry norms for safety related systems. Defect density was reported at less than 1 defect/kLOC, with 1 design error uncovered to date and 1 safety-related error [16].

Let us compare this project against the three conditions for industrial usage mentioned above. Here VDM was primarily used as a modelling technique and the VDM model served as an oracle against the corresponding implementation. The key developers already knew VDM so no training was required. Finally, the automation support provided by the interpreter was essential in order to achieve a cost-effective development.

B. TradeOne, CSK Systems (2000-2005)

JFITS, Japan, now CSK Systems³ developed the TradeOne back-office system for securities trading. Two subsystems managing options and tax exemptions were developed using VDM⁺⁺. Metrics were reported in 2005 [8]. The two subsystems combined are 78,637 DSI in C++ and Java. In the options business logic and business logic libraries, 18501 DSI of VDM⁺⁺ gave rise to 48029 DSI of C++. The developers believed that, with increased experience, they could have rapidly halved the size of the VDM⁺⁺ model by eliminating repetition. The effort profile for the options subsystem is shown in Figure 1

Phase	% effort
Education	4.7
Analysis	41.6
Design	11.6
Implementation (incl. unit test)	27.3
Test	14.8

Fig. 1. Effort Profile for TradeOne Options Subsystem developed using VDM⁺⁺

The defect rates for the tax exemption subsystem were 0.65 defects/kDSI⁴ and for the options subsystem 0.71 defects/kDSI. By comparison, defect rates for the whole TradeOne product, the majority of which was developed without formal techniques, average 1.12 defects/kDSI. Development costs were compared against COCOMO estimates. For the tax exemption subsystem, development took 36% of the COCOMO estimated effort; development of the options subsystem took 40% of estimated effort.

The project was considered a success. CSK subsequently bought the rights to the VDMTools technology that was used in the TradeOne development when IFAD, its original developer, moved out of the business area.

Let us again compare this project with our three conditions for industrial usage. In this project VDM was once more used as a modelling technique rather than as a specification language for subsequent refinement. A few of the key developers already had VDM expertise so training was limited

¹Downloads are available via www.vdmbok.com

²www.adelard.co.uk

³www.csk.com/systems/

⁴DSI = Delivered Source Instructions.

to the newcomers in the project. Here it was particularly important that VDMTools support Unicode so that the users could write the VDM models directly in Japanese. As with DustExpert, the automation support provided by the interpreter was essential. However, here the possibility of incorporating functionality from a standard library into the interpreter using a dynamic link library facility was also a contribution to cost-effectiveness.

C. FeliCa Networks

FeliCa Networks Inc.⁵ in Japan recently used VDM⁺⁺ in the development of a next generation mobile IC chip, based on a contactless card technology developed and promoted by Sony. The card is connected to networks via cellular telephones and can be used for identification and e-commerce purposes via wireless communication with devices such as ticket barriers in transport networks; the chip has an associated secure communications protocol. VDM⁺⁺ was introduced into the development process with the aim of improving requirements quality. FeliCa reports a process of staged development of product specifications from informal natural language requirements involving the augmentation of natural language requirements with UML notations and then the construction of an executable formal model in VDM⁺⁺. Review processes are applied to all three forms of model (inspection on Natural Language requirements, static type analysis and testing on the VDM⁺⁺ models) [17].

The specification process has developed over 100kLOC of VDM⁺⁺ in a 677 page document. Validation tests on the model have provided 100% coverage by over 10 million tests. Review statistics show 440 contradictions and faults in requirements and specifications discovered to date; 278 of these are attributed to activities that rely on the development and analysis of a formal model. Of the 278, 162 are attributed to review of the model and 116 are attributed to the ability to build an executable formal model that could be analysed dynamically.

The FeliCa development team here includes more than 50 people and the three year project has been completed on time, which is remarkable in itself. The product is to be produced in a high volume, with potentially high recall costs. The application of VDM⁺⁺ is viewed as a success. Effort and final product defect rate figures are not publicly available.

Once again, let us compare this project with our three conditions for industrial usage. In this project VDM was again used as a modelling technique alongside an established UML-based development. Almost none of the developers had VDM expertise prior to the project; training and documentation was provided for these newcomers in Japanese so it was easily accessible to them. It was again important that VDMTools support Unicode so that the users could write the VDM models directly in Japanese. As in the other projects the automation support provided by the interpreter was essential in order to achieve a cost-effective development. Due to the very large

number of test cases the efficiency of the interpreter became essential and CSK managed to improve its speed by a factor of more than 100 in order to do this.

IV. FUTURE DIRECTIONS

Future work in lightweight application of VDM⁺⁺ will be influenced by developments in applications and tools technology. In the applications sphere, a goal for model-oriented methods is to bridge the gap between systems and software engineers. The challenges here relate to the distributed, real-time character of systems and the need to make continuous and stochastic models work in concert with discrete logical models. In the tools sphere, the goals of enhanced proof-based analysis that does not require technical prover-driving skills, and improving tools interoperability, are of increasing significance.

In the lightweight view, formal methods are simply tools to be used in the wider systems development process. Such a view helps to bridge the gap between traditionally separated engineering disciplines such as control engineering and software development. For example, from a tools perspective it is relatively straightforward to couple an animation of a formal model of a controller with a tool modelling the continuous behaviour of the environment. In a similar vein, we are beginning to experiment with coupling stochastic models of faults with system models describing fault tolerance strategies.

Modelling and analysing real-time distributed systems remains a significant challenge for the formal methods community generally. The major complexities lie in the accurate simulation of the run-time environment, scheduling and distribution. A lightweight simulation-based approach may provide an opportunity to at least identify bottlenecks at early development stages. Extensions have been proposed to VDM⁺⁺ to permit the modelling and execution-based analysis of timing properties [9]. Automated proofs of end-to-end properties remain a research goal.

Classically, formal methods involve proof, or at least exhaustive model checking. Proof support for VDM was one focus of the MURAL tool [10], as well as the TIAPS [18], [19] and PROSPER [20] projects. Until recently, we have not seen a strong enough industrial case for bringing proof support into VDMTools because of the computational overheads and also the possibility of having to build an interface for user guidance of the specialised proof process. However, proof technology has advanced to the point where we believe that an automated prover can be adapted to discharge automatically generated VDM proof obligations in the background⁶. In accordance with our “Automation” principle, we would leave undischarged obligations for inspection. Subsequent work could lead to the development of an acceptable interactive proof interface, but we see this as a lower priority for industrial use than automation.

A further area of interest is the growing importance of tools interoperability, notably supported by technologies such as

⁵www.felicanetworks.co.jp

⁶This is a developing capability of B tools in the Rodin framework: www.rodin.cs.ncl.ac.uk

XML and the Eclipse framework. We have recently begun the Overture Initiative⁷, a community-based open source project aiming to develop new tools for a VDM⁺⁺-like language based on an architecture designed to promote such interoperability.

V. CONCLUSIONS

There have been some triumphs for lightweight model-oriented formal methods, but there are also new challenges. Industrial experience suggests that model-oriented formalisms, although general purpose, are well suited to serious industrial applications in a variety of contexts, provided they are applied in a lightweight way, with tools that support lightweight application. Further, we do not advocate model-oriented formalisms as a universal solution: our goal is limited to using these techniques in applications that suit their strengths, notably the abstract modelling of data and functionality.

Alongside VDM⁺⁺, we believe that similar stories can be told for other model-oriented formal methods such as B and Z. Although there are syntactic and semantic differences between such formalisms, cost-effective industrial usage is much more heavily influenced by the differences between the forms of tool support available. For example, enhanced proof support for VDM could make it viable to implement a form of refinement-based development process, again stressing automated discharging of obligations over interactive proof.

New challenges for these methods arise from both emerging application areas and the improving technology underpinning tool support. We have identified the potential of using lightweight model-oriented formalisms to promote closer coupling between systems and software design, as well as the potential for automated proof support and future development environments composed of interoperable specialised tools.

REFERENCES

- [1] C. B. Jones, "A Rigorous Approach to Formal Methods," *IEEE Computer*, vol. 29, no. 4, pp. 20–21, April 1996.
- [2] D. Jackson and J. Wing, "Lightweight Formal Methods," *IEEE Computer*, vol. 29, no. 4, pp. 22–23, April 1996.
- [3] C. B. Jones, *Systematic Software Development Using VDM*, 2nd ed. Englewood Cliffs, New Jersey: Prentice-Hall International, 1990.
- [4] J. S. Fitzgerald and P. G. Larsen, *Modelling systems: practical tools and techniques in software development*, 1998.
- [5] C. B. Jones, "Scientific Decisions which Characterize VDM," in *FM'99 - Formal Methods*, ser. Lecture Notes in Computer Science, J. Wing, J. Woodcock, and J. Davies, Eds., vol. 1708. Springer-Verlag, 1999, pp. 28–47.
- [6] D. J. Andrews, Ed., *Information technology – Programming languages, their environments and system software interfaces – Vienna Development Method – Specification Language – Part 1: Base language*. International Organization for Standardization, December 1996, international Standard ISO/IEC 13817-1.
- [7] J. C. Bicarregui, J. S. Fitzgerald, P. A. Lindsay, R. Moore, and B. Ritchie, *Proof in VDM: A Practitioner's Guide*, ser. FACIT. Springer-Verlag, 1994.
- [8] J. S. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object-oriented Systems*. Springer-Verlag, 2005.
- [9] M. Verhoef, P. G. Larsen, and J. Hooman, "Modeling and Validating Distributed Embedded Real-Time Systems with VDM⁺⁺," in *FM 2006*, ser. Lecture Notes in Computer Science, J. Misra and T. Nipkow, Eds., vol. 4085. Springer-Verlag, 2006.
- [10] C. Jones, K. Jones, P. Lindsay, and R. Moore, Eds., *mural: A Formal Development Support System*. Springer-Verlag, 1991.
- [11] P. G. Larsen, J. S. Fitzgerald, and T. Brookes, "Applying Formal Specification in Industry," *IEEE Software*, vol. 13, no. 3, pp. 48–56, May 1996.
- [12] R. Bloomfield, P. Froome, and B. Monahan, "SpecBox: A Toolkit for BSI-VDM," *SafetyNet*, no. 5, pp. 4–7, 1989.
- [13] P. G. Larsen and P. B. Lassen, "An Executable Subset of Meta-IV with Loose Specification," in *VDM '91: Formal Software Development Methods*, VDM Europe. Springer-Verlag, March 1991.
- [14] R. Elmström, P. G. Larsen, and P. B. Lassen, "The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications," *ACM Sigplan Notices*, vol. 29, no. 9, pp. 77–80, September 1994.
- [15] N. Terada, "Application of formal methods to automatic train control systems," in *Symposium on Formal Methods for Railway Operation and Control Systems (FORMS 2003)*, 2003, l'Harmattan Hongrie.
- [16] T. Clement, I. Cottam, P. Froome, and C. Jones, "The development of a commercial "shrink-wrapped application to safety integrity level 2: the dust-expert story," in *Safecomp '99*. Toulouse, France: Springer Verlag, September 1999, INCS 1698, ISBN 3-540-66488-2.
- [17] T. Kurita, T. Oota, and Y. Nakatsugawa, "Formal Specification of an Embedded IC Chip for Cellular Phones," in *Proceedings of Software Symposium 2005*. Software Engineers Associates of Japan, June 2005, pp. 73–80, (in Japanese).
- [18] S. Agerholm, "Translating specifications in VDM-SL to PVS," in *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '96)*, ser. LNCS, J. von Wright, J. Grundy, and J. Harrison, Eds., vol. 1125. Springer-Verlag, 1996.
- [19] S. Agerholm and J. Frost, "An Isabelle-based theorem prover for VDM-SL," in *Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics (TPHOLs '97)*, ser. LNCS. Springer-Verlag, August 1997, also available as technical report IT-TR: 1997-009 from the Department of Information Technology at the Technical University of Denmark.
- [20] L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham, "The PROSPER Toolkit," in *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer-Verlag. Berlin, Germany: Lecture Notes in Computer Science volume 1785, March/April 2000.

⁷www.overturetool.org