Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods

J. S. Fitzgerald and P. G. Larsen

TECHNICAL REPORT SERIES

No. CS-TR-999 January, 2007

TECHNICAL REPORT SERIES

No. CS-TR-999 January, 2007

riumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods

J. S. Fitzgerald and P. G. Larsen

Abstract

The "lightweight formal methods" paradigm emphasises the use of abstract modelling as an aid to understanding and design of computer-based systems. It advocates careful targeting of formal methods technology on specific system parts or aspects, rather than large-scale application. The challenge of implementing the lightweight paradigm was taken up a decade ago by the community working with the Vienna Development Method (VDM), developing its semantics, tools and encouraging industry application. This paper reports industrial successes over that period, and identifies challenges for industrial formal methods application in the near future.

This paper is to appear in the proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006).

Bibliographical details

FITZGERALD, J. S., LARSEN, P. G..

Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods
[By] J. S. Fitzgerald, P. G. Larsen.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2007.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-999)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-999

Abstract

The "lightweight formal methods" paradigm emphasises the use of abstract modelling as an aid to understanding and design of computer-based systems. It advocates careful targeting of formal methods technology on specific system parts or aspects, rather than large-scale application. The challenge of implementing the lightweight paradigm was taken up a decade ago by the community working with the Vienna Development Method (VDM), developing its semantics, tools and encouraging industry application. This paper reports industrial successes over that period, and identifies challenges for industrial formal methods application in the near future.

This paper is to appear in the proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006).

About the author

John Fitzgerald is a Reader in Computing Science at Newcastle University. His research concerns the application of formal methods, especially machine-assisted proof. A particular area of interest is predictable dynamic resilience – the design of systems which reconfigure in response to threats. He leads work on resilience-explicit computing in the European Network of Excellence on Resilience in IST and is co-Investigator in the EPSRC Trustworthy Ambient Systems project.

Peter Gorm Larsen is Professor of Computer Technology and Embedded Systems at The Engineering College of Aarhus, Denmark. He is an authority on system modelling, and particularly the Vienna Development Method. Alongside research on VDM's foundations, he has pioneered the development of industrial-strength tool support for model-oriented specification languages, latterly heading the group that developed what are now the CSK VDMTools(R)products. He also works as an independent consultant.

Suggested keywords

FORMAL METHODS,
SOFTWARE ENGINEERING,
DEPENDABILITY,
SOFTWARE TOOLS,
INDUSTRIAL APPLICATION

Triumphs and Challenges for the Industrial Application of Model-Oriented Formal Methods

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

***Abstract*— The "lightweight formal methods" paradigm emphasises the use of abstract modelling as an aid to understanding and design of computer-based systems. It advocates careful targeting of formal methods technology on specific system parts or aspects, rather than large-scale application. The challenge of implementing the lightweight paradigm was taken up a decade ago by the community working with the Vienna Development Method (VDM), developing its semantics, tools and encouraging industry application. This paper reports industrial successes over that period, and identifies challenges for industrial formal methods application in the near future.**

I. Introduction

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 1996 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. Section II gives a brief introduction to the formalism at the core of VDM and the principles underpinning its application as a lightweight formal method. A short description of tool support (Section III) leads to an account of several major industrial applications of VDM (Section IV). Future directions and some conclusions are discussed (Sections V and VI).

II. 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 over both types and state variables. Functionality is defined in terms of referentially transparent functions over the defined 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.

Figures 1, 2 and 3 give examples of extracts based on a VDM model of a railway interlocking system developed by Natsuki Terada [10] of the Railway Technical Research Institute, Japan. The examples are given in the ASCII syntax accepted by the main VDM tool set. We have found that the non-mathematical syntax is more acceptable to industry users, who are already familiar with programming languages, than the more dense traditional notations. Figure 1 shows the definition of a data type `Route` as a composite record structure defined using constructors for sets and nonempty sequences. The data type invariant, stated as a predicate after the keyword `inv`, is an integral part of the type definition: a record not respecting the invariant is not an element of the `Route` type.

```
types

TrackId = token;

Route :: allroute   : seq1 of TrackId
         partroute  : seq1 of TrackId
         points     : PointDir
         route_lock : set of RouteId

inv len allroute > 1 and
    elems allroute subset dom points and
    (forall i,j in set inds allroute &
       i <> j =>
         allroute(i) <> allroute(j)) and
     exists i in set inds allroute &
       partroute =
         allroute(i,...,len allroute);
```

Fig. 1. Example VDM type definitions with an invariant.

Figure 2 shows an explicit function definition. Such referentially transparent executable function specifications typically utilise the rich set of underlying operators on the types and type constructors. For example, the function defined in the figure uses operators for set intersection (`inter`) and extracting the range of finite mappings (`rng`). Figure 3 shows the use of state variables to model persistent data: the variables `head` and `body` represent persistent data constrained by an invariant. An operation with side effects on the variables is described implicitly, in terms of a pre- and postcondition. The postcondition typically has to refer to the values of state variables both before and after the operation is applied, the "before" value being denoted by a "˜" decoration on the relevant variable. In this example the postcondition characterises a result state uniquely, and so the operation could have been specified explicitly.

```
Route_Connected:(map TrackId to Track) *
                (map RouteId to Route)
                 -> bool
Route_Connected(trackcs, routes) ==
  forall r in set rng routes &
    forall i in set inds r.allroute &
      i + 1 in set inds r.allroute =>
        trackcs(r.allroute(i)).seg
        (r.points(r.allroute(i))) inter
        trackcs(r.allroute(i+1)).seg
        (r.points(r.allroute(i+1))) <> {}

pre Route_Defined(trackcs, routes);
```

Fig. 2. An explicit function definition in VDM.

```
state Train of
  head : Node;
  body : seq1 of TrackId;
inv forall i,j in set inds body &
       i <> j => body(i) <> body(j)
end

operations

  AddTrack(newhead,tcid) ==
  ext wr head : Node;
      wr body : seq1 of TrackId
  pre tcid not in set elems body
  post body = [tcid] ^ body~ and
       head = newhead;
```

Fig. 3. A VDM state definition with implicitly specified operation

VDM models can, of course, be checked statically for basic forms of type correctness. However, the expressiveness of the language, including the use of unconstrained predicates as invariants, means that such checking is necessarily limited. A full static analysis of a VDM model generates a series of *proof obligations*, each of which represents a check that could be performed by a theorem prover with human guidance. For example, the operation specification `AddTrack` in Figure 3 gives rise to a proof obligation to show that the operation is satisfiable. The following conjecture must be proved:

```
forall t~:Train, n:Node, ti:TrackId &
  pre-AddTrack(n,ti,t~) =>
  exists t:Train &
    post-AddTrack(n,ti,t~,t)
```

The typed Logic of Partial Functions (LPF), described in [7], underpins VDM. However, so far only the MURAL tool [11] 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. In this paper, we concentrate here on the latter strand of work, the aim of which 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 [12] 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].

Many different kinds of model are used in systems development, and no one model is fit for all purposes [13]. In VDM, the purpose of a model is the main factor governing the particular abstraction decisions made. The value of a model is assessed in terms of the insight and the assurance it gives with respect to its declared purpose. There is no doubt that this insight depends heavily on the quality of analysis that can be performed on a model. If this is so, the quality of available tool support is the major factor influencing the cost (in terms of engineer time) of obtaining insight into the model. Only by comparing the effort spent and the insight gained can it be determined whether the time spent on any kind of modelling and analysis is worthwhile.

III. 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 [14], 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, Elmstrøm, Larsen and Lassen developed a toolset that implemented an operational semantics [15], [16], [17]. 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 its original developer IFAD to the Japanese corporation CSK Systems[1]. 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 and an application programmer interface, allowing the model to be animated directly without the need for translation to a programming language. 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 [10].

IV. 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 a number of 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 five 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)

Adelard[2] 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 the manual construction of proofs of safety properties, test-based analysis using the VDM-Tools 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 that 1 defect/kLOC, with 1 design error uncovered to date and 1 safety-related error [18].

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. SIC2000, GAO (1997 - 1998)

GAO (Gesellschaft für Automation und Organisation), now integrated into its parent company Giesecke & Devrient, controls 75% of the world market for banknote processing machinery at central banks, and is currently rapidly expanding into the commercial banking and cash handling market. A banknote processing system (BPS) presents a heterogeneous computing environment, typically including several communication buses and numerous embedded processors which are subject to hard real-time constraints and which control and monitor banknote feature measurement sensors, and banknote transport, stacking, banding, and destruction activities.

[1]Downloads are available via `http://www.vdmtools.jp/en/`

[2]`www.adelard.co.uk`

Four different projects at GAO made use of VDM-SL and VDMTools. The largest of these was the development of SIC2000 [19], a new generation of banknote processing machines. Because of the complex and strategic nature of the project, and due to a persistent emphasis in the sensor development department on the importance of software quality, the decision was made to develop the software with the aid of VDM. Eight developers were involved in different parts of the project which in total took about two person-years of effort. One person-year was spend on the formal specification, the subsequent implementation only took three person-months and testing of this took additional four person-weeks. Several errors were detected, all but one attributable to an imperfect translation of the specification into code. In the case of one error, the specification had to be revised.

In addition 6 person-months were spent on interfacing to low-level sensors and two person-months on testing that integration. Again several errors were detected but none of them resulted in the VDM specification needing modification. This project was considered a success and efforts were made to introduce VDM as an integral part of the software development process at GAO. For example, parts of the specification have been reused for different products without the need for modifications.

The total size of the VDM-SL document (with annotations etc.) was around 150 pages. For efficency reasons the C code was produced manually from this specification. In one instance two lines of VDM are implemented by almost 4000 lines of code spread among 15 different modules. This is mainly because this particular part is responsible for looking up information for specific telegrams with a rate of 375000 entries per second.

Let us again compare this project against the three conditions for industrial usage mentioned above. Again VDM was used as an abstract modelling technique where many aspects of the embedded and distributed system was abstracted away. The CORBA-based API was also important for the integration testing where event protocols could be gathered from the real components and translated into VDM. Finally automation was the key to the financial success of this project in that the interpreter was able to take test cases and deliver results automatically.

C. Cava, BeoLogic/BAAN (1998 - 1999)

At Beologic (the company was sold to BAAN while this project was undertaken) an application was developed more in line with the origin of VDM. This VDM project was for the core of an advanced sales configuration application which was developing support for a language called Cava (an object-oriented constraint-based language inspired from Java). For this language a complete abstract syntax was defined and over that a static semantics and a dynamic semantics (a constraint solved) was defined using VDM-SL.

The development team was distributed on locations in The Netherlands, the US and Denmark. This is always challenging and the use of a common notation such as VDM here helped in gaining consensus during decision-making. The development tasks were divided into a number of components and the kernel ones were developed using a VDM model. Each of these was more than 100 pages long and quite complex, reflecting the complexity of the Cava language. Since the efficiency of the actual implemented constraint solver was paramount the VDM model of this part was rather low-level, but it enabled the developers rapidly to explore alternative implementation strategies.

Comparing this project against our three conditions for industrial use, again VDM was used as an abstract modelling technique but here the model was used as a prototype to gain consensus on the language decisions. Employees from the different locations had a one week training course and after that were able to express themselves using the VDM notations. With respect to the accesibility it was essential that it was possible to write the VDM model directly inside a Word document. This was done in such a way that the VDM chunks where tagged as hidden code which could be displayed by those readers who wanted to see the details and ignored by those for whom the textual description was sufficient. The interpreter from VDMTools was again key to the success of the project because in essence it provided the development team with an efficient prototyping facility.

D. TradeOne, CSK Systems (2000-2005)

JFITS, Japan, now CSK Systems[3] 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[4] 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 4.

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

Fig. 4. Effort Profile for TradeOne Options Subsystem developed using VDM^{++}

The defect rates after system release for the tax exemption subsystem were zero defects/kDSI and for the options subsystem 0.05 defects/kDSI. By comparison, defect rates for the whole TradeOne product, the majority of which was developed without formal techniques, average 0.67 defects/kDSI. Development costs were compared against COCOMO estimates. For the tax exemption subsystem, development took 36% of

[3]`www.csk.com/systems/`

[4]DSI = Delivered Source Instructions.

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 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.

E. FeliCa Networks (2005 - 2006)

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) [20].

The specification development process was carried out in three phases:

1) Writing an informal definition of the requirements in Japanese (383 pages).
2) Creating UML diagrams was made originally based on this document.
3) Modelling the system in VDM^{++} with over 100kLOC of VDM^{++} (677 pages).

Validation tests on the VDM^{++} model provided 100% coverage by peforming over 10 million tests. During phases 1 and 2 reviews found only 93 contradictions and faults in requirements and specifications in total. In phase 3, 162 faults were found through the process of writing and reviewing the VDM^{++} model. In addition 116 faults were found by executing the formal model in VDMTools. Finally, an extra 69 faults were found by combining the evaluation team and the specification writing teams in reviews.

[5] `www.felicanetworks.co.jp`

The FeliCa development team included 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 in case of defects). By the end of November 2006 more than one million chips had been shipped. 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 felt to be essential to achieving cost-effective development. Given the very large volume of tests to be applied to the model, CSK invested effort to improve the interpreter's speed by a factor of more than 100.

F. Application Domains for VDM in General

There are many more applications of VDM and VDM^{++} than we are able to describe here. Perhaps the most quoted application using VDM is a display system for the London Air Traffic Control Centre is not mentioned here at all, because none of the authors have been involved with that project at all. However, others have reported analysis data about that project in the past [21], [22]. Some applications that we can not report are with large companies like Lockheed-Martin, Boeing, BAE Systems, Matra, Dassault and Aerospatiale. In addition there are several applications in which the user, for commercial reasons, does not want competitors to know that they have used a particular technology with the aim of gaining a cometitive advantage.

The variety of domains covered in the examples that we have reported suggests that model-oriented formalisms have the potential to be applied widely. However, there is a tendency to restrict the use of this technology to areas in which high quality is the dominant concern, or the complexity of functionality or data threatens the success of a development. One important direction for applied research is towards wider lightweight use of formalism.

V. 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 [11], as well as the TIAPS [23], [24] and PROSPER [25] 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 background[6]. 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 XML and the Eclipse framework. We have recently begun the Overture Initiative[7], a community-based open source project aiming to develop new tools for a language based on VDM^{++}. The tools'architecture designed to promote such interoperability.

VI. 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.

We have followed the "lightweight" formal methods vision advocated in 1996 in some respects, but not all. Considering Jones' advocacy of rigour rather than full formality [1], we have not forced users to discharge proof obligations or to pursue development through refinement or formal verification. However, we have encouraged the use of formality in so far as models are constructed in a formal notation. The analysis of models involves the generation of proof obligations which may be discharged at a level of rigour determined by the developer. Jackson and Wing [2] envisaged languages of limited expressiveness but used in a focussed manner. Again, we have only partially adopted this notion. VDM remains a highly expressive language. We have, however, sought to provide tool support that makes it cost-effective to use in a precisely targeted way within existing development practices.

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.

Acknowledgements

We are grateful to the many collaborators with whom we have worked on the projects described in this paper. We thank Shin Sahara, Marcel Verhoef, Richard Payne and Zoe Andrews for their valuable comments on earlier drafts; Tiziana Margaria and Bernhard Steffen for the motivation to prepare the paper; and the organisers of the ISoLA 2006 Symposium. Fitzgerald particularly acknowledges support from the European Network of Excellence on Resilience for Survivability in Information Society Technologies (ReSIST).

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.

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

[7]`www.overturetool.org`

[4] J. S. Fitzgerald and P. G. Larsen, *Modelling systems: practical tools and techniques in software development*. Cambridge University Press, 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] 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.

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

[12] 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.

[13] A. Davis, *Software Requirements: Objects, Functions and States*. Upper Saddle River, NJ: Prentice Hall, 1990.

[14] R. Bloomfield, P. Froome, and B. Monahan, “SpecBox: A Toolkit for BSI-VDM,” *SafetyNet*, no. 5, pp. 4–7, 1989.

[15] 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.

[16] R. Elmstrøm, 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.

[17] P. Mukherjee, “Computer-aided Validation of Formal Specifications,” *Software Engineering Journal*, pp. 133–140, July 1995.

[18] 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, lNCS 1698, ISBN 3-540-66488-2.

[19] P. R. Smith and P. G. Larsen, “Applications of VDM in Banknote Processing,” in *VDM in 2000!*, J. Fitzgerald and P. G. Larsen, Eds., September 1999, pp. 67–79.

[20] 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).

[21] A. Hall, “Using formal methods to develop an atc information system,” *IEEE Software*, vol. 12, no. 6, pp. 66–76, March 1996.

[22] S. Pfleeger and L. Hatton, “Investigating the influence of formal methods,” *IEEE Computer*, no. 2, pp. 33–42, 1997.

[23] 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.

[24] 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.

[25] 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.