The Evolution of VDM Tools from the 1990s to 2015 and the Influence of CAMILA

Peter Gorm Larsen\textsuperscript{a}, John Fitzgerald\textsuperscript{b}

\textsuperscript{a}Department of Engineering, Aarhus University, Denmark, pgl@eng.au.dk
\textsuperscript{b}School of Computing Science, Newcastle University, Newcastle upon Tyne, UK, john.fitzgerald@newcastle.ac.uk

Abstract

The Vienna Development Method (VDM) is one of the most mature formal methods, with a history of cost-effective industrial deployment. One important route for this has been the development of robust tools supporting the construction of models, and their animation. We trace the history of this strand of work from the mid-1990s to 2015, taking as our starting point challenges for the industrial usage of formal methods set out by José Nuno Oliveira in 1997. We describe five generations of VDM tools: the IFAD VDM Toolbox, VDMTools, Overture, Crescendo and Symphony, emphasising the influence that the goal of industry usage has had on their features and architectures. We chart the move from a single-formalism tool focussed on executable VDM specifications to a platform for multi-tool analysis of a wider range of models, and look forward to the growth of integrated multidisciplinary toolchains from the ongoing INTO-CPS project. We briefly compare the VDM tool story with the approaches taken by other formalisms that have been applied in industry.

Keywords: VDM, CAMILA, tool support, history, industrial applications

1. Introduction

Decades of persistent research on applicable formal methods, often in the face of scepticism, is bearing fruit, and there are now many notable successes to report \cite{1,2}. These are in large part due to the research and practitioner community better tuning formalisms to industrial needs, and developing the robust tools that are a \textit{sine qua non} for effective application. It was not always clear that the goal of stronger tooling would be pursued, but the vision and contribution of scientists...
such as José Nuno Oliveira have been fundamental to achieving the positive state of formal methods today.

The Vienna Development Method (VDM) originated in the pioneering work of IBM’s Vienna Laboratory on the challenges of defining programming language semantics and of designing trustworthy compilers. In his 1999 review of the scientific history of VDM, Jones remarked that “In spite of a number of efforts, projects to provide tool support have never been easy to justify” \cite{3}. In the years since, the VDM community has set itself the goal of creating a formal method that is supported well by tools that deliver the capabilities needed for industry deployment. A measure of its success is to be found in significant applications like that of the FeliCa chip firmware, deployed in more than 259 million mobile telephones \cite{4} using VDMTools \cite{5}.

We would argue that three principles have tacitly guided VDM tools work in the last 25 years. First, priorities for tooling have gone hand-in-hand with the needs of targeted industry application, leading to an emphasis on model construction and simulation. Second, tools robustness is as important as capability, with the consequence that a release discipline has been developed even for what is sometimes unfunded tool development. Third, tools should be integrated with, rather than supplant, established workflows; it is consequently necessary to understand workflows before deployment of tools.

This paper reviews the recent history of tools development for VDM, with its focus on support for model construction and analysis through simulation and industrial usage. In Section 2 we first elaborate concerns on the industrial take-up for formal methods, as articulated by Oliveira in 1997. These provided significant motivation for both the CAMILA framework and the emerging IFAD VDMTools (Section 3). We describe the influence of this work on the development first of VDMTools (Section 4) and then its open-source cousin Overture (Section 5), and discuss how that framework is being extended today to provide formal model-based tools for the design and analysis of increasingly demanding classes of product, notably Systems of Systems and Cyber-Physical Systems (Section 6). Finally, we briefly relate the VDM tool development to other formal methods (Section 7), discuss the extent to which we have succeeded or failed to meet Oliveira’s challenges, and look to future directions (Section 8). Throughout, we aim to provide an extensive bibliography, particularly relating to the history of VDM’s tool support.
2. Enabling Industrial use of Formal Methods

In May 1997, Oliveira gave a presentation at the United Nations University International Institute for Software Technology [6], in which he listed his main concerns for industrial usage of formal methods as follows:

**Simplicity:** Industry will never absorb formal methods based on complex mathematical theories.

**Compatibility:** Formal methods cannot replace traditional methods altogether.

**Tools:** Formal methods should be supported by tools and environments.

**Flexibility:** Formal method tools should be easily portable across different machine platforms and be able to communicate with existing (traditional) tools.

**Modularity:** Common sense should be applied to software development; warning: informal modularity is worse than formal monolithic development.

**Reusability:** Requires a formal classification scheme; otherwise, repositories become full of things we will never find.

**Back to “good engineering habits”:** In school physics we are taught a universal strategy for problem solving: understand the problem, build a mathematical model of your understanding of it, reason in this model, upgrade your model, if necessary, and calculate a solution. Why don’t we do likewise in software development?

The authors of this paper were making similar observations to Oliveira at around the same time [7]. We were motivated by the experience of applying VDM at British Aerospace (Systems & Equipment), in an early comparative study of software development with and without formal models [8]. Our strongest focus was perhaps on Oliveira’s compatibility and tools issues. With others we advocated a pragmatic “lightweight” approach to formal methods in which methods would remain fully formal but would be applied to subsystems and system features that merited the investment [9, 10]. We also observed that industrial users

---

1Bernhard Steffen once remarked to Fitzgerald that, although our approach might be called “lightweight”, the specific gravity of VDM remained the same – in fact we had provided the machinery for lifting it!
rarely develop systems from scratch; instead they often build on existing solutions or use existing components from other projects. Thus, there is a clear need to lower the barriers to the use of formal techniques where legacy features exist.

Finding an appropriate balance between the effort spent on producing formal models and the value they bring either in the form of new insight or in the form of a product is paramount to the industrial application of formal methods [11]. In order to gain value from formal models it is important to supply efficient tool support that enables users rapidly to understand models, identify weaknesses and explore alternatives, and here the notion of executable models becomes important.

A model-based formal method such as VDM has many features that appear familiar to software engineers with experience of imperative and functional programming. In working with engineers in industry we rarely found difficulties in understanding the elements of VDM as a modelling language; the most challenging skill to teach is the crucial one of abstraction [15]. Abstraction decisions should be governed by the purpose of the model [16], but models must remain rich enough to be competent in the sense that engineers should have confidence that the outcomes of model-based analysis will reflect the properties of the realisation. A focus on abstraction skills in formal methods education is both essential and, we argue is enhanced by the use of tools [17].

3. CAMILA/SETS and IFAD VDM-SL Toolbox Developed in Parallel

The CAMILA initiative at the University of Minho, led by Oliveira (the original project for its development was from 1990 to 1993) [18, 19, 20, 21] developed a formal modelling notation inspired by functional programming and set theory. The focus here was on the value of such a notation in education, and significant effort was put into the categorical basis and the capability for formal refinement capabilities by calculation. However, it contained very strong support for interpretation of models as a form of prototyping. This also included support in the interpreter for calling external code. In order to illustrate the features of the notation, and compare with VDM, we use extracts of the small Bank Account Management System (BAMS) example from [19]. The data types for this example are shown in Figure 1 and a little of the functionality is shown in Figure 2. The interest in formal approaches for Oliveira was a natural continuation from his PhD thesis [22].

\footnote{Many model-based formal methods now define executable subsets, but the value of executability was debated because of the risk of compromising abstraction [12, 13, 14].}
Withdraw : AccId × Amount → Void

Withdraw(n, m) ≜

\[ bams' = bams^\dagger \text{ let } \text{subm} = \lambda(x). \text{ let } b = B(x) \]
\[ c = \begin{cases} b \geq 0 & \Rightarrow b - m \\ \neg(b \geq 0) & \Rightarrow b \end{cases} \]
\[ \text{in } Account(H(x), c) \]
\[ \text{in } * \leftrightarrow \text{subm}(bams | \{n\}) \]

GoodCustomers : Amount → 2^{AccHolder}

GoodCustomers(m) ≜
\[ \bigcup \{H(bams(k)) \mid k \in \text{dom}(bams) \land B(bams(k)) \geq m\} \]

Figure 1: The types of the BAMS example in CAMILA.

Figure 2: Extract of functionality from the BAMS example in CAMILA.
In parallel with the CAMILA development, the IFAD VDM-SL Toolbox actually began with the creation of an interpreter for a subset of VDM-SL [23, 24] at the company IFAD A/S in Denmark. This was used in the Incremental Prototyping Technology for Embedded Real-Time Systems (IPTES) project to simulate the nodes of Data Flow Diagrams (DFDs). At that time, DFDs were a widely used structured notation and the purpose of the toolbox was to give engineers the flexibility to augment structured methods by formal specification ensuring, in Oliveira’s terms, compatibility between the two.

The first publication on the IFAD VDM-SL Toolbox as a whole had been made in 1994 [25]. Additional features were subsequently developed including a type checker [26], connection of the interpreter to legacy code [27] inspired by the CAMILA work, and proof obligation generation [28, 29]. Industrial applications covered satellite launching software [30, 31], door control software for metro rail systems [32], satellite software [33], banknote processing control [34] and auction software [35]. It is worth noting that the developers of the IFAD VDM-SL Toolbox “took their own medicine” in that formal models where produced for the core parts of the tool. Since some of these VDM specifications were large, this meant that performance issues with the tool were discovered first by the development team and mostly fixed before they reached industrial users. Finally it may be worth noting that in parallel to this the ISO standardisation of the VDM-SL notation was also conducted [36, 37].

It may be instructive to compare a VDM-SL rendering of the BAMS example with the CAMILA version in Figure 1. The VDM-SL is as follows:

```
state BAMS of
  bams : map AccId to Account
init b == b = mk_BAMS({|->})
end

types

Account ::
  H : set of AccHolder
  B : Amount;

AccId = seq of char;
AccHolder = seq of char;
Amount = int
```

There is a high degree of similarity, the main difference being that BAMS is mod-
elled as a persistent set of state variables that can be manipulated by specified
operations. In addition, it is worth noting that the type representation for sets
of elements is substantially different. If we compare the functionality shown in
Figure 2 in CAMILA the similar representation in VDM-SL would be as follows:

```
operations

Withdraw: AccId * Amount ==> ()
Withdraw(n,m) ==
  let acc = bams(n),
      b = acc.B
  in
    bams(n) := mu (acc, B |-> if b >= m
                   then b - m
                   else b)

pre n in set dom bams;

functions

GoodCustomers: map AccId to Account * Amount -> set of AccHolder
GoodCustomers(m,amount) ==
  dunion { m(id).H | id in set dom m & m(id).B >= amount}
```

Again the similarity is remarkable, although it is worth noting that CAMILA did
not make use of the design by contract principles with invariants and pre- and
post-conditions that are commonplace in VDM-SL. It is also worth noting that
the VDM-SL representation uses an ASCII notation. Early versions of the IFAD
VDM-SL Toolbox also used the mathematical syntax in a \LaTeX{} pretty printer,
but from the first industrial application, we learned that mathematical symbols
presented a barrier to many industrial practitioners. To some extent this is aligned
with Oliveira’s recommendation for simplicity, albeit at a superficial level.

One of the main reasons for the successful use of both CAMILA and the IFAD
VDM-SL Toolbox was the strength of tool support. However, it is fair to say that
the commercial effort spent on the IFAD VDM-SL Toolbox resulted in a more
mature and usable tool. The IFAD VDM-SL Toolbox was eventually also used at
Minho University for teaching.
4. The VDMTools Era

In the mid-1990s, the VDM++ language was developed in the EC Framework Programme (FP) 3 project “Applying Formal Methods to Real-Size Object-Oriented Designs in Technical Environments” (AFRODITE). Once IFAD was involved with this project the tool support for VDM++ was combined with the code base for the IFAD VDM-SL Toolbox. Towards the end of the decade the IFAD VDM-SL Toolbox was renamed to VDMTools \[5\] and was improved in many directions with a focus on industrial usage \[38\], including links to Microsoft Word and Rational Rose \[3\]. This included code generation to C++ and Java \[43\], and mapping between VDM++ and UML \[44\]. These approaches align well with the flexibility and modularity concerns raised by Oliveira. However as can be seen from Figure 3 the tool was centred around one single large specification manager. From a development perspective, the complex dependencies between components were making the build system of the corresponding executables ever more difficult.

In 2004, VDMTools was taken over by one of its customers in Japan \[45\].

\[3\] Significant research on the addition of proof support was also conducted \[39, 40, 41, 42\].
Japan Future Information Technology Systems (JFITS) had experienced impressive productivity results in the development of key subsystems of a large application called TradeOne (see [46], Chapter 11). This was the main reason for taking the technology in-house, developing it further, and promoting it in particular to Japanese industry. Subsequently, VDMTools were to prove particularly successful for Felica Networks (a subsidiary of the Sony corporation), who developed the firmware (100 kLOC) stored in a Near-Field-Communication (NFC) chip that has been deployed in total in more than 259 million mobile phones [47, 4, 48]. It is perhaps worth noting that this beneficial take-up of VDM technology was accelerated by investment in some apparently prosaic activities, such as the translation of manuals to local languages (i.e. Japanese), and the development of versions of the tools that operated with local character sets (i.e. using Unicode).

5. Moving to Open Source: Overture

After Larsen left IFAD it was decided to start a new open source initiative for a tool that was suited both to industrial use and as a research platform. The Overture initiative [49] was started by the co-authors of the 2005 VDM++ book [46], and later merged with the VDMJ tool developed by Battle [50]. Initially the main part of the Overture development was carried out through Masters thesis projects on the Eclipse platform [51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62]. VDMJ was a purely command-line tool, so the combination with Overture made sense. Many of the features that existed in VDMTools were redeveloped for the Overture platform, including the connection to UML [63], code generation [64, 65], proof obligation generation [66], the interpreter [67] and connections of the interpreter to external code [68]. Entirely new features such as combinatorial testing were developed for both VDMTools and Overture [69]. Proof-of-concept versions of features for theorem proving [70], connection to JML [71], partitioning for hardware/software co-design [72] and Hardware-in-the-Loop simulation [73] were also developed.

As suggested in Figure 4, increased emphasis was placed on following a plug-in architecture, making it easier to develop and build, and to provide a basis for future contributions. It is also worth noting that the core development is entirely independent of the Eclipse-based user interface. This means that we also have a command-line version of Overture, enabling a future change to the front-end if

---

4The company was later renamed to CSK and then to SCSK.
One of the major developments in this period was a Real-Time dialect for VDM (VDM-RT\textsuperscript{6})\textsuperscript{6} [78, 79, 80]. This supported the development of distributed embedded systems and included a notion of time as well as a notion of a basic system architecture in which processes were deployed onto virtual CPUs connected by buses. A first process for the systematic development of such systems was also proposed [81] with experimental tool support for visualising and analysing properties over traces of such animations [82, 83].

6. Extensions of Overture: Symphony, Crescendo and Beyond

The recent history of Overture has been one of evolution from a mono-disciplinary VDM tool to one that serves as an enabling platform, promoting the desired\textsuperscript{5}.

\textsuperscript{5}A new Master’s thesis project for developing a WebIDE of Overture is scheduled for completion in 2016, taking inspiration from VDMPad [74, 75].

\textsuperscript{6}The origin of VDM-RT lies in the “VDM In Constrained Environments (VICE) project [77], but that version was only intended for a single CPU; the dialect was initially also supported by VDMTools.
collaborative use of other well-founded tools dealing with complementary system aspects. Such a platform is essential to support the forms of multidisciplinary engineering inherent in future embedded and “smart” systems. In this section, we outline three areas in which this aspect of Overture has grown in recent years: Systems of Systems (SoSs) engineering, embedded systems, and Cyber-Physical Systems (CPSs).


An SoS contains independently owned and managed systems that together offer an emergent service on which reliance is placed [84]. Examples include infrastructure such as smart grids and transport networks, as well as applications in healthcare, emergency response, and defence. The SoS engineer faces challenges arising from the independence of constituent systems, the need to validate emergent properties [85], and the diversity of stakeholders [86]. SoS engineering is thus inherently collaborative, and tools for model-based SoS engineering should be Collaborative Development Environments which is a special kind of IDE supporting the joint production of models that can be systematically analysed even though one constituent system’s owner might not wish (or be able) to disclose every part of their system’s model [87, 88, 89].

The FP7 project COMPASS\textsuperscript{7} sought to provide the first formal foundations, methods and tools specifically targeting the challenges of SoS engineering. Following our principle of integration with established development workflows, we realised the need to integrate architectural models of SoS structure and constituent system capabilities with formal contractual (assume, guarantee) models of constituent system interfaces. The architectural modelling was carried out using SysML and contractual specifications were delivered through a special-purpose modelling language (the COMPASS Modelling Language, CML [90]). CML incorporates data and functional modelling from VDM, adding communication and concurrency constructs derived from Circus and CSP [91, 92]. CML provides a semantic model for SoS descriptions, and so the provision of its own semantics has been a challenging and rewarding test for the Unifying Theories of Programming (UTP) [93]. In CML the data type definitions for the BAMS example from Figure 1 would appear as follows:

\footnotesize

\begin{verbatim}
process BAMS =
begin
\end{verbatim}

\end{footnotesize}

\footnotesize

\textsuperscript{7}See \url{http://www.compass-research.eu/}.
state
  bams : map AccId to Account := {||->}

types
  Account ::
    H : set of AccHolder
    B : Amount
  AccId = seq of char
  AccHolder = seq of char
  Amount = int

The main difference here is that states reside inside CSP/Circus-like processes, but otherwise this is virtually syntactically identical to the VDM-SL version. The functionality described in Figure 2 but with a reactive part showing the actions that can be performed at different stages:

Withdraw: AccId * Amount ==> ()
Withdraw(n,m) ==
  let acc = bams(n),
      b = acc.B
  in
    bams(n) := mk_Account(acc.H, if b >= m
        then b - m
        else b)
  pre PreWithdraw(n,bams)

functions

GoodCustomers: map AccId to Account * Amount -> set of AccHolder
GoodCustomers(m,amount) ==
  dunion { m(id).H | id in set dom m @ m(id).B >= amount}

PreWithdraw: AccId * map AccId to Account -> bool
PreWithdraw(n,bams) ==
  n in set dom bams

actions
  BANK =
    ([] n in set AccIdSet @
      withdraw?m -> [PreWithdraw(n,bams)] & Withdraw(n,m); BANK)
... // other potential actions...
@ init -> BANK

The model of functionality is also close to that of VDM-SL (modulo the need to use record constructor expressions inside the Withdraw operation). CML supports inter-process communication on channels such as init and withdraw. This is just an extract: in the full model, the BANK action would have other operations available. The Withdraw operation is guarded by its pre-condition (if the pre-condition is false the operation call is not enabled) and semantically this blocks until it is satisfied. If all possible actions block like this, a deadlock has been reached.

In COMPASS, Overture was extended to form the core of the Symphony IDE [88, 94, 89]. Symphony (Figure 5) links tools addressing SysML modelling, formal CML model-based analysis, and test automation as well as additional external tools. Construction, maintenance and analysis of CML models is supported by the Symphony tools implemented on the Overture platform. Test automation is supported by the RT-Tester tool which identifies test cases, traces them to re-

---

8See http://symphonytool.org/
quirements and generates concrete test data by means of an integrated constraint solver [95]. There are also dependencies on the Isabelle theorem prover and on the Microsoft FORMULA model checker [96]. Distributed simulation was supported experimentally [97], as was the combination of the interpreter with the ProB tool [98] for interpreting implicit specifications [99]. It is worth noting that the simulator plugin is also able to co-simulate models that contain external components; this is done via libraries that embedded in the external component, and which allow for communication with the simulator plugin.

The range and depth of the analyses that can be accomplished using this Overture-based framework was demonstrated in COMPASS through a series of industry case studies in audio/video streaming networks [100] and emergency response.

6.2. Embedded Systems: Co-modelling and Co-simulation with Crescendo

One of the archetypal application domains demanding multidisciplinary design is that of embedded systems. It appears to be increasingly understood that the premature allocation of responsibilities to software and physical engineering disciplines in the design of embedded systems has unfortunate consequences. Separate design disciplines using diverse notations (discrete event notations such as VDM rich in data and logic abstractions on the one hand, and continuous time and value notations defining behaviour as systems of differential equations on the other) can lead to miscommunication. Further, mismatches and bottlenecks may only be discovered when the two sides are brought together in physical prototypes. While single hybrid modelling notations go some way to remedying this, they can entail asking developers to abandon notations that have served them well for decades or more.

The FP7 project DESTECS [101] developed methods and tools for collaborative modelling and co-simulation, combining Discrete-Event (DE) models of controllers with Continuous-Time (CT) models of controlled plant. The two types of model continue to use their own specialist rich notations, but are kept in harness by a co-simulation engine that implements an operational semantics linking two simulations, managing the passage of time and communication of state data between the two. The collaborative development environment created in DESTECS is named Crescendo [9]. Its co-simulation engine links DE models in VDM-RT to models of controlled plant expressed using the 20-sim tool [10]. To the user, the

---

9 See http://crescendotool.org/
10 See http://www.20sim.com/
Overture-based Crescendo tool serves as a point of entry to the co-model and its constituent parts, launching 20-sim to deliver simulations of the controlled plant (its architecture can be seen in Figure 6). Features of the development environment include the ability to perform sweeps over design parameters, allowing Design Space Exploration (DSE) in the search for optimal designs. The vision here is to allow trade-off decisions to be made easily between cyber and physical elements, rather than requiring the early implementation commitments that so hamper rapid innovation.

Industrial case studies showed that the approach was not only viable, but that it demonstrably reduced the number of physical prototypes needed in product development [102]. Co-models allowed early experimentation, for example with complex control logic and alternative safety strategies and architectures. After the completion of the DESTECS project additional industrial use of the technology has been reported [103, 104, 105].

While we have proven the concept of cost-effective multidisciplinary model-based design using formal techniques, Crescendo is limited to just DE models in
VDM and CT models in 20-sim. Equally importantly, the models are limited in size as well as scope, and we lack the architectural structuring needed to describe larger and more heterogeneous systems. This is related both to Oliveira’s tool support and modularity issues discussed in Section 2.

6.3. From Tools to Toolchains: INTO-CPS

In 1997, Oliveira argued for communication between formal tools and external tools. Over fifteen years on, we suggest that there is a pressing need not only for formal tools that can exchange data, but for semantically integrated toolchains. The motivation for this arises from our concern that modern systems and products are cyber-physical in character, and hence inherently multidisciplinary. In January 2015 the project INTO-CPS began what we hope will be a shift in emphasis from individual tools to coherent toolchains that support realistic workflows.

CPSs range from networked embedded systems to large-scale applications such as smart grids and integrated transport systems. The foundations, methods and tools of CPS engineering should incorporate both discrete models of computing and the continuous-value abstractions of physical (e.g. mechanical, electrical, electronic) engineering. Given the augmentation of engineered systems with computing capabilities [106], one may envisage even more diverse models covering human, social and economic aspects. If dependable CPSs are to be engineered economically, the design process must therefore be collaborative and multi-disciplinary, while also raising assurance through simulation, testing and verification. There have been repeated calls for better notations for model-based CPS engineering [107, 108], but there are many challenges [109]. While much research in formal methods delivers individually effective tools, multi-disciplinary CPS design requires coherent toolchains formed from diverse tools, each optimised for a given purpose.

Overture has grown to support not only model-based design in VDM, but also collaborative and multidisciplinary design by linking other external tools, and has been increasingly engineered for extensibility [76, 110, 111, 112, 113]. In Symphony we showed the possibility of using Overture as a platform for a range of externally developed and internal tools. In the work on embedded systems in Crescendo, we also demonstrated a semantically deep integration of quite heterogeneous tools to facilitate co-simulation. The goal of our current work[1] in the INTO-CPS project [102] is to develop a well-founded toolchain rather than a

Aside from semantic heterogeneity, we see two other significant issues. First, the design space for a CPS is large; we require exploration of trade-offs between physical components, hardware and software, rapidly modifying and reevaluating designs. Second, support is needed to maintain traceability over the complex collections of artefacts produced in a CPS development, allowing the provenance of all elements to be recorded, and the final system linked to the requirements.

Using foundations defined using UTP (inspired by the work in the COMPASS project), we aim to create a family of interlinked tools supporting CPS development from requirements and architectural modelling formalised using SysML, via the Function Mockup Interface (FMI) interface definitions to co-models. The toolchain is intended to permit static analysis of co-models, as well as DE/CT co-simulation and co-simulation of models with implementations. We aim to allow these co-simulations to be exploited with DSE (inspired by the work carried out in the DESTECS project) and test automation (using RT Tester as in the COMPASS project). The conceptual toolchain can be seen in Figure 7. The baseline technologies are Modelio\textsuperscript{12} for SysML, co-modelling and co-simulation using VDM Overture, 20-sim\textsuperscript{13} and OpenModelica\textsuperscript{14} Co-simulation will build on Crescendo and the TWT co-simulation engine\textsuperscript{15} and test automation builds on RT-Tester\textsuperscript{16} We plan to evaluate the framework using industrial applications in railways, agriculture, automotive systems and building automation. These are provided and developed by four of the industrial partners of the consortium: ClearSy (France), Agro Intelligence (Denmark), TWT (Germany) and United Technologies (Ireland).

7. VDM in the Wider Formal Methods Context

VDM is not alone in aiming for ever more capable and robust support tools with the goal of industrial application. It is worthwhile briefly considering how approaches and motivations have differed, and how this has affected the tools and the record of industry application. VDM belongs to the group of model-based

\textsuperscript{12}http://www.modelio.org/
\textsuperscript{13}http://overturetool.org/
\textsuperscript{14}http://www.20sim.com/
\textsuperscript{15}https://www.openmodelica.org/
\textsuperscript{16}http://www.twt-gmbh.de/produkte/co-simulationen/co-simulation-framework.html/
\textsuperscript{17}http://www.verified.de/products/rt-tester/
formal methods that is typically said to include formalisms such as B and Abstract State Machines. B \cite{114} and Event-B \cite{115} have benefited from considerable research investment in recent years, including the high-profile project Deploy \cite{116}, which evaluated the industrial deployment of Event-B, based on the Eclipse-based Rodin toolset\cite{17} and the Pro-B animator and model-checker\cite{18}. The formalisms focus strongly on refinement to implementation, with tools development being driven to a large degree by the desire to deliver maximum automation in discharging refinement obligations associated with design decision, in contrast with the simulation focus of VDM. Building on the Event-B/Rodin foundations, the project ADVANCE\cite{19} has sought to provide a tools environment that supports cyber-physical systems design. Again, in contrast to the VDM-based work that we have described here, ADVANCE has sought to provide integrated support co-simulation \cite{117} as well as proof.

The community working in Abstract State Machines (ASM) \cite{118,119} has also been successful in industrial applications \cite{120}. Several ASM tools have been developed since 2000 but unfortunately these are not 100% compatible and

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{toolchain.png}
\caption{The envisaged toolchain for INTO-CPS.}
\end{figure}
the different tool initiatives are not so well coordinated. Here CoreASM is the main tool environment for the development and execution of ASM. ASM is similar to VDM in the sense that it is a multi-purpose notation suitable for modelling algorithms, protocols, systems, etc. Just like VDM it was originally developed to describe the semantics of programming languages. The CoreASM engine executes CoreASM specifications and is designed to with an extensible plugin-based architecture. Compared to the VDM tools focus there has been more focus on verification support in the ASM tool support.

The Temporal Logic of Actions (TLA) was introduced in the early 1990s by Lamport. The TLA Toolbox is an IDE for TLA+ which is a specification language for modelling and verifying concurrent systems. Recently the TLA Proof System has been extended by connection with SMT solvers via translation of the TLA specification syntax into the input language of the SMT solvers. Compared to the VDM, developments in TLA have been more focused on raising model checking capabilities. Impressive applications of TLA+ have been reported. Both ASM and TLA+ are substantially supported by different teams at Microsoft Research.

8. Concluding Remarks

VDM tools development has been guided by the desire to achieve industrial impact and deployment. How far have we succeeded against the challenges expressed by Oliveira in 1997? From an early stage we sought simplicity by focusing on concepts that would be familiar to developers in industry. We prioritised support for models that use data and functional abstraction, and validation through simulation and test, rather than pushing users prematurely towards the less robust technology of proof. Some of the VDM variants, including VDM++ and VDM-RT, have features that are semantically complex (object-orientation and concurrency features, for example) but which are superficially familiar to developers in certain application domains. We suspect that future work on the language will start to emphasise axiomatic semantics to support proof automation, and this may lead to re-simplification in some areas.

The feature of compatibility has been a major focus, realised through the development of methods that yield improvements in existing development processes. We have sought to pay increasing attention to the role of models in the design.
flow [130], and this has influenced the tools in that we have provided support for development activities rather than mandating a specific process as a prerequisite for their use. Although it was an important point to reiterate in 1997, and something emphasised by the CAMILA/SETS work, the need for good tools is almost always taken as read today. In line with Oliveira’s suggestion to seek flexibility, our industry experience is increasingly leading us to the need for formal methods researcher to address the development of toolchains at a deep semantic level.

Although we believe progress is being made against the majority of Oliveira’s goals, support both foundational and practical for modularity and reusability remain weak. Support for model management and traceability will assume increasing priority for VDM.

Finally, regarding good engineering habits, Oliveira’s point in 1997 was that the production and exploitation of models at all would be a step forward. The need for strong engineer education has not diminished in recent years, and Oliveira’s own contributions as convener of the Formal Methods Europe (FME) Subgroup on Education bear this out [131]. Formal methods teachers have a responsibility to deliver training that meets the needs of the majority of computing graduates and does not alienate them from the use of formal model-based design. We have found that there is some potential in delivering courses that integrate model-based design with familiar development activities, and these approaches are supported by the tools framework that we have developed [17].

The decisions characterising the development of VDM support technology in the last 15+ years have been influenced by goals that we have shared with Oliveira, and others in the field. Achievements to date have relied as much upon persistence in the face of scepticism as they have depended upon technical advances. The challenges facing us in the next 15 years concern a widening of scope to the design of complex systems – not only software. This requires us to address a range of stimulating but substantial challenges at the level of compatible semantic foundations, pragmatic methods and integrated toolchains. More than ever, formal methods have a great deal to offer the engineering professions. However, as hopefully demonstrated by this article a significant amount of work is required in order to achieve industrially applicable tool support for formal methods.

Acknowledgments.

We are grateful to José Nuno Oliveira for the inspiration and wisdom he has provided over decades, and in particular for the open and welcoming research group that he has created at Minho.

VDM and its tools have benefitted from the contributions of many research
scientists and engineers over the decades. Although they are far too many to list, we record our gratitude to them all. For their inputs on this paper, we are particularly grateful to Nick Battle and Victor Bandur, as well as to the editors and reviewers of this volume.

It is difficult to understate the contribution that the European Union and its research programmes have made by enabling collaboration across national boundaries, between industry and academia, and between disciplines. VDM++ was initially developed in Afrodite (FP3, project 1798). Our collaboration on industry deployment was initiated in project ConForm (ESSI, 10670). Crescendo and Symphony were developed in DESTECS (FP7, 248134), and COMPASS (FP7, 287829). Our current work is partially supported by the INTO-CPS project (Horizon 2020, 664047). Many of our collaborative activities have been facilitated by Formal Methods Europe (FME), which began as VDM Europe – an EC initiative to which Oliveira contributed on many occasions as a symposium chair [132] and as a leader of its educational initiatives.

References


tion – an Industrial Experiment using the CAMILA/SETS Approach, Sem-
inar, UNU/IIST, Macau (May 1997).

(Eds.), 2nd BCS-FACS Northern Formal Methods Workshop, BCS-FACS,


in: Proceedings of the International Workshop on Current Trends in Ap-

trial Uptake of Formal Methods, in: C. B. Jones, Z. Liu, J. Woodcock
(Eds.), Formal Methods and Hybrid Real-Time Systems, Essays in Honour
of Dines Bjørner and Chaochen Zhou on the Occasion of Their 70th Birthdays,
237–254, iSBN 978-3-540-75220-2.

[12] I. Hayes, C. Jones, Specifications are not (Necessarily) Executable,
URL http://www.cs.man.ac.uk/csonly/cstechrep/
Abstracts/UMCS-89-12-1.html

[13] N. E. Fuchs, Specifications are (preferably) executable, Software Engineering

[14] M. Andersen, R. Elmstrøm, P. B. Lassen, P. G. Larsen, Making Specifica-
tions Executable – Using IPTES Meta-IV, Microprocessing and Micropro-

[15] J. Kramer, Is Abstraction the Key to Computing?, Communica-
tions of the ACM 50 (4) (2007) 37–42. doi:http://doi.acm.org/ez-always.statsbiblioteket.dk:
2048/10.1145/1232743.1232745


URL http://overturetool.org/publications/books/vdoos/


URL http://doi.acm.org/10.1145/1668862.1668864


URLs:
- http://dx.doi.org/10.1007/978-3-642-05089-3_36
- http://dx.doi.org/10.1007/978-3-642-24559-6_14
- http://dx.doi.org/10.1109/SEFM.2010.32


URL http://dx.doi.org/10.1007/s00165-015-0342-2


[103] M. P. Christiansen, K. Bjerge, G. Edwards, P. G. Larsen, Towards a Methodology for Modelling and Validation of an Agricultural Vehicle’s


