The IFAD VDM-SL Toolbox: A Practical Approach to Formal Specifications

René Elmstrøm, Peter Gorm Larsen and Poul Bøgh Lassen
IFAD
The Institute of Applied Computer Science
Forskerparken 10, DK-5230 Odense, Denmark
{rene|peter|poul}@ifad.dk

Abstract

The IFAD VDM-SL Toolbox is a collection of tools for formal specifications development using the latest version of the VDM-SL standard. In addition to the full language it also supports a module-based structuring mechanism for large specifications. The Toolbox features extensive semantics checking, documentation support, test coverage analysis and debugging support. We have focused on supporting real-life specifications development in industrial settings. This paper presents the Toolbox and also reports on our own experience using it for the development of large specifications.

1 Background

The Vienna Development Method (VDM) is one of the most mature formal methods, primarily intended for the formal specification and the subsequent development of functional aspects of software systems. A central element of VDM is its specification language: VDM-SL.

VDM-SL is used during the specification and design phases of a software development project and supports the production of correct and high quality software. Compared to traditional natural language specifications, a formal specification is unambiguous and can be automatically checked for many inconsistencies and errors.

VDM-SL is being standardised under the auspices of the International Standards Institution (ISO) and the British Standards Institution (BSI). It is currently a Committee Draft standard under ISO and it is expected that this draft will be accepted as a final standard more or less in its current form.

The use of VDM-SL in the development of software systems has been approached in various ways in the past. A very pragmatic approach reported is based on prototyping VDM-SL specifications using a programming language. E.g. [3] reports on the use of ABC+ to prototype VDM-SL specifications for educational purposes and [1] reports on (a prototype) semi-automatic translation system producing Lazy ML code from VDM-SL specifications. These approaches however have the drawback that another formalism (ABC+ or Lazy ML) must be introduced to cope with the semantics of the VDM-SL specifications. Furthermore the support provided for the prototyping process in these tools is not specially tailored for VDM-SL but rather dependent on the programming language in use. A more formal approach to the use of VDM-SL is through proof of essential properties. Many text books cover this area e.g. [2] provides a sound constructive approach to proof in VDM.

A problem with using a fully formal development (discharging all proof obligations) is that the resources needed in most cases are impossible to provide for a typical development project. Carrying out proofs on industrial scale applications even supported through state-of-the-art tools is very expensive.

The IFAD VDM-SL Toolbox is based on a practical approach to using VDM-SL in the system development process. The tool supports the full standard VDM-SL language through syntax checking, extensive static semantics checking (extended type checking) and documentation support. Furthermore it supports the validation of specifications written in a (large) executable subset of VDM-SL using testing and debugging techniques. Our experience with the use of the tool is that these techniques work well for specifications

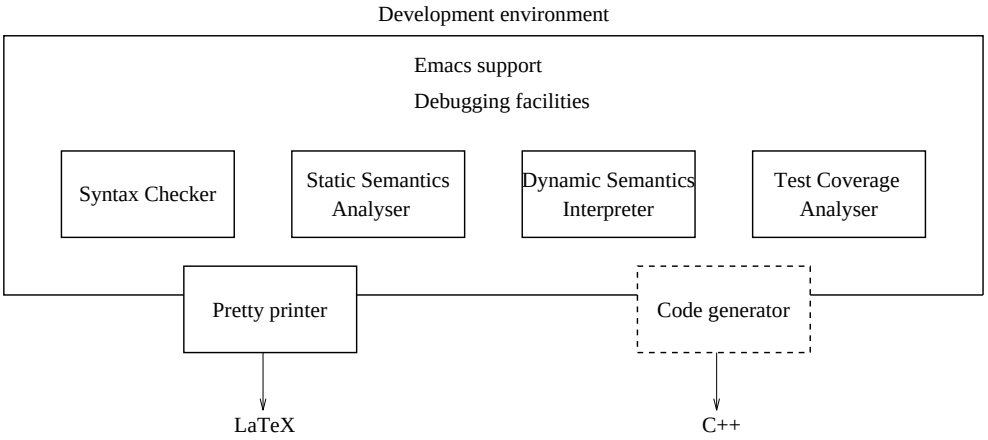

Figure 1: The Structure of the IFAD VDM-SL Toolbox

up to a size of 5000 – 10000 lines of VDM-SL.

2 Working with the Toolbox

An overview of the structure of the Toolbox is presented in Figure 1.

We will present the use of the different tool components by describing the way in which we typically use the tool in the development of specifications normally produced in a development team of 2 - 4 members.

Developing specifications in parallel, calls for supporting specifications physically stored in a number of files. The Toolbox supports the development and validation of module structured[1] specifications distributed in any number of files. Furthermore since VDM-SL specifications are written in their ASCII representation, a standard version control system can be applied to the specification files (as e. g. RCS).

2.1 Producing Documents

The documentation value of a VDM-SL specification is large. Since the specification provides an unambiguous description of a system the specification is helpful both as a basis for the implementation and later as technical documentation of the implemented system. This implies that keeping a high quality of the documentation produced for VDM-SL specifications is very important.

We see VDM-SL specifications as an integrated part of a system's documentation including e.g. informal specifications, graphical structure information and specification annotations. The Toolbox supports this idea. VDM-SL specifications are not written in isolation but as parts of LaTeX documents (like in literate programming) which may also include any other type of graphical or textual documentation. This way a VDM-SL specification may be distributed throughout a (number of) LaTeX document(s). VDM-SL specifications are written in the standard ASCII syntax and the pretty printing facility of the Toolbox produces LaTeX macros to support type-setting specifications in the mathematical syntax. Furthermore the pretty printer supports the use of the LaTeX indexing facility when generating documentation output. This way an index of the definition and use of all functions, operations, types and constants in the specification is automatically generated. We have found these facilities to be essential when handling large specification documents.

[1] Modules are not supported in the current version of the VDM standard; however work is in progress to include them.

```
  ...                                  vdm> rtinfo DFD TransClosure
* 561  DFD`LowerChar
  403  DFD`ToLower                        Callers :
  245  DFD`FlowIdVarConf                    76  DFD  ExecutionOrders
  196  DFD`StateVarIntConf                  30  DFD  NeedsQuant
  150  DFD`StateVarConf                     37  DFD  TransClosure
* 143  DFD`TransClosure
* 143  DFD`UpperChar                      Descendants :
  121  DFD`ToUpper                          37  DFD  TransClosure
* 117  DFD`OpIdConf
  99   DFD`FlowIdTypeConf                 At line: 607 construct not executed at column 29
  ...                                  vdm>
```

Figure 2: Example test coverage information generated by the Toolbox.

2.2 Syntax Checking

The Toolbox supports syntax checking of VDM-SL specifications. The syntax checker can be called from within Emacs (the Toolbox comes with an interface supporting the Emacs editor) which will allow you to step through the syntax errors found. The Toolbox will indicate the error position directly in the specification source files. If you prefer to use another editor the Toolbox allows you to activate the syntax checker from the command line and will produce a list of error messages including the line and column number of the errors found.

2.3 Static Semantics Checking

Having established a VDM-SL specification as syntactically correct, the next step is to check its static semantics (type errors, scope errors, etc.). VDM-SL has a very powerful type system supporting complex (from a type checking point of view) type constructs like union types and recursively defined types. This makes thorough type checking of VDM-SL specifications a challenge. The Toolbox includes a static semantics analyser which is able to check for a large number of static semantics errors well-known from normal programming language type checkers e.g. correct values applied to function calls, assignments, use of undefined variables and module imports/exports. Furthermore it provides specific support for formal specification languages with extended checks giving warnings corresponding to proof obligations in cases like e.g. potential division by zero or potential violation of type invariants. In all these cases the types involved can be arbitrary complex compositions of any of the standard VDM-SL types.

For large specifications it is important to be able to check smaller parts of the specification in isolation before they are integrated. The static semantics analyser can check VDM-SL modules in isolation but when checking several modules will also check the consistency of the module interfaces.

2.4 Prototyping

To aid the understanding of complex specifications the Toolbox supports execution and debugging. The debugger supports many of the facilities known from debuggers of programming languages such as setting breakpoints (at functions and operations), stepping (performed at expression and statement level) and inspecting the current calling stack. Furthermore the interpreter supports VDM-SL specific facilities such as dynamic checking of type invariants and checking pre and post conditions for functions and operations.

One of the advantages of the Toolbox compared to other approaches reported to prototype VDM-

SL specifications is the large subset of VDM-SL supported for execution. All the VDM-SL constructs supported for execution by e.g. [1] and [3] are executable directly in the Toolbox. Furthermore a number of more advanced constructs are supported by the Toolbox including: higher order functions, polymorphic functions, complex (loose) pattern matching, comprehensions (map, set, sequence), lambda expressions and exception handling. Since the full VDM-SL language is not executable in general, the Toolbox does not support execution of type bindings and purely implicitly defined (pre/post style) functions and operations.

2.5 Systematic Testing

Systematic test of the specification is for us the most important way to get confidence in the correctness of a large specification. In addition to working interactively it is possible to use the interpreter in a test mode. We use this facility to set up (using standard shell scripts) a test suite for the specification. This enables us to perform a thorough test of reasonably large specifications and run it as a batch process.

To get an overview of how well a test suite has covered a specification, the Toolbox is able to collect statistics of the testing activity.

Figure 2 shows an example of the statistical information collected by the test coverage facility. All functions and operations of the specification are listed on the left hand side and the number of calls to them are given. Furthermore a * indicates that not all constructs of this function/operation have been evaluated during the test suite. On the right hand side more detailed information about the `DFD‘TransClosure` function is provided. The callers and descendants of the function are listed and the position of constructs not executed are given. This information is used to document the level of testing performed and to plan further testing activities.

2.6 Implementation

Once the specification has been suitably validated (using reviews and testing) implementation is the next step to be covered. In our own work we have found that implementing a formal specification is an almost mechanical (but manual) effort when one has first decided on an implementation strategy. We ourselves use a strategy for C++ where the data handled by an application are represented by a number of C++ classes representing the common VDM-SL data types (sets, maps, sequences etc.).

Furthermore we re-use the same test cases developed for the specification to test the implementation as well. This gives us a certain degree of confidence that no more errors are introduced during the implementation phase.

3 Availability and Future Work

Automated C++ code generation of VDM-SL specifications is currently being developed for the Toolbox and is expected to be available in 4th quarter of 1994. In this development work we use (again) the development strategy outlined here.

A commercial supported version of the IFAD VDM-SL Toolbox is now available for Sun SPARC (SunOS) and supports all facilities described here.

A free evaluation version of the tool is available through ftp at `hermes.ifad.dk (130.225.136.3)` in the directory `/pub/toolbox`. For further information contact `toolbox@ifad.dk`.

References

[1] P. Borba and S. Meira. From VDM Specifications to Functional Prototypes. *Journal of Systems Software*, 21:267–278, 1993.

[2] J. Bicarregui, J. Fitzgerald, P. Lindsay, R. Moore and B. Ritchie. *Proof in VDM: A Practitioner's Guide*. FACIT. Springer-Verlag, 1994.

[3] A. Kans and C. Hayton. Using ABC To Prototype VDM Specifications. *ACM SIGPLAN Notices*, pages 27–37, January 1994.