VALIDATE OBJECT-ORIENTED MODELS USING VDM++

Anding Anak Nyuak

Master of Science
2010
VALIDATE OBJECT-ORIENTED MODELS USING VDM++

ANDING ANAK NYUAK

A dissertation submitted
In partial fulfillment of the requirements for the degree of
Master of Advanced Information Technology

Faculty of Computer Science and Information Technology
UNIVERSITI MALAYSIA SARAWAK
2010
**Declaration**

I certify that all works in this thesis have not been submitted for any academic awards at other colleges, institutes or universities. The work presented here is carried out under the supervision of Dr. Edwin Mit. All other works in the thesis are my own except those where noted.

Signed,

Anding anak Nyuak

July, 2010

---

**Acknowledgment**

I would like to thank Dr. Edwin Mit for his inspiring comments and proofreading. Many thanks for his valuable ideas, helpful suggestions and comments and proofreading the thesis. Special thanks to my wife and families for their love and support, which gave me courage to complete this dissertation. Last but not least, I would like to thank all my colleagues for their cooperation in many ways.
# Table of Contents

Declaration ........................................................................................................................ i
Acknowledgment ............................................................................................................... i
Table of Contents .............................................................................................................. ii
List of Tables ..................................................................................................................... vi
List of Figures .................................................................................................................... vii
Abstract ........................................................................................................................... ix

## Chapter 1: Overview ........................................................................................................ 1

1.1 Background ................................................................................................................... 1
1.2 Objectives ...................................................................................................................... 2
1.3 Problem statement ........................................................................................................ 2
1.4 Purpose of study ............................................................................................................ 3
1.5 Research Contribution ................................................................................................. 4
1.6 Scope of Work ............................................................................................................... 4
1.7 Chapter overview .......................................................................................................... 4

## Chapter 2: Literature Review .......................................................................................... 6

2.1 Introduction ................................................................................................................... 6
2.2 Object-Oriented (OO) Tools ......................................................................................... 6
  2.2.1 Rational Rose 2002 ............................................................................................... 7
  2.2.2 Unified Modeling Language Tool: ArgoUML ...................................................... 8
  2.2.3 Object-Oriented Tools Comparison ..................................................................... 9
2.3 Formal Methods and Tool Support ............................................................................ 11
  2.3.1 Formal Method – Vienna Development Method (VDM) language ..................... 11
    2.3.1.1 VDM Tools .................................................................................................. 12
  2.3.2 Formal Method - Z Language .............................................................................. 13
    2.3.2.1 Strengths of Z language .............................................................................. 13
    2.3.2.2 Weaknesses of Z language ........................................................................... 13
  2.3.3 Event-B language ................................................................................................. 14
  2.3.4 Formal Methods Tools Comparison ................................................................... 14
  2.3.5 OO and FM Integration Tools ............................................................................. 15
    2.3.5.1 Overture ..................................................................................................... 15
    2.3.5.2 Object Z ..................................................................................................... 16


Chapter 3: An Overview of UML and VDM++

3.1 Introduction .................................................................................................................. 20
3.2 The Unified Modeling Language
   3.2.1 UML Static Models ............................................................................................... 21
   3.2.2 Use Case Diagrams .............................................................................................. 21
   3.2.3 Class Diagrams ...................................................................................................... 23
   3.2.4 UML Class ............................................................................................................. 23
   3.2.5 UML Attributes .................................................................................................... 24
   3.2.6 UML Operations ................................................................................................... 25
3.3 Introduction to VDM and VDM++ .............................................................................. 26
   3.3.1 Object-Oriented Concepts in VDM++ ................................................................. 26
   3.3.2 VDM++ Class ....................................................................................................... 27
   3.3.3 Type Definitions ................................................................................................. 29
      3.3.3.1 Basic Data Types ......................................................................................... 29
      3.3.3.2 Compound Types ..................................................................................... 30
   3.3.4 Value Definitions ............................................................................................... 31
   3.3.5 Operation Definitions ......................................................................................... 31
      3.3.5.1 Explicit operation definition ....................................................................... 32
   3.3.6 Instance Variables ............................................................................................. 33
   3.3.7 Multiplicity and Ordering in VDM++ ................................................................. 34
3.4 Specifications using VDM++ ...................................................................................... 34
3.5 Proposed Prototype ..................................................................................................... 35
   3.5.1 Proposed Prototype Architecture ....................................................................... 36
   3.5.2 The UML Interface Components ....................................................................... 37
   3.5.3 The VDM++ Interface Components .................................................................. 37
   3.5.4 The XML Repository ......................................................................................... 38
3.6 Conclusion .................................................................................................................... 38

Chapter 4: Rules for Translating UML to VDM++

4.1 Introduction .................................................................................................................. 39
4.2 Translating UML Class Diagrams to VDM++ .......................................................... 39
   4.2.1 Generating a VDM++ Class Name ..................................................................... 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5.1 Capturing UML Static Model</td>
<td>75</td>
</tr>
<tr>
<td>5.5.2 Prototype Testing and Evaluation</td>
<td>76</td>
</tr>
<tr>
<td>5.6 Conclusions</td>
<td>77</td>
</tr>
<tr>
<td>Chapter 6: System Implementation and Testing</td>
<td>78</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>78</td>
</tr>
<tr>
<td>6.2 Implementation</td>
<td>78</td>
</tr>
<tr>
<td>6.3 Testing Approach</td>
<td>79</td>
</tr>
<tr>
<td>6.4 Case study: Campaign Management</td>
<td>81</td>
</tr>
<tr>
<td>6.4.1 Requirements Specification</td>
<td>83</td>
</tr>
<tr>
<td>6.4.2 Capture of UML specifications of Add a new Client</td>
<td>84</td>
</tr>
<tr>
<td>6.4.3 The XML Representation of the UML specification</td>
<td>87</td>
</tr>
<tr>
<td>6.4.4 Generation of VDM++</td>
<td>89</td>
</tr>
<tr>
<td>6.4.5 User Defined Data Type</td>
<td>90</td>
</tr>
<tr>
<td>6.5 Evaluations of Translation Rules</td>
<td>95</td>
</tr>
<tr>
<td>6.6 Conclusion</td>
<td>97</td>
</tr>
<tr>
<td>Chapter 7: Conclusion and Future Work</td>
<td>98</td>
</tr>
<tr>
<td>7.1 Introduction</td>
<td>98</td>
</tr>
<tr>
<td>7.2 Discussion</td>
<td>98</td>
</tr>
<tr>
<td>7.3 Future work</td>
<td>103</td>
</tr>
<tr>
<td>7.4 Conclusion</td>
<td>104</td>
</tr>
<tr>
<td>References</td>
<td>105</td>
</tr>
</tbody>
</table>
# List of Tables

2.1: UML and Rational Rose 2002 ......................................................... 8
2.2: Object-Oriented Tools Comparison ............................................. 10
2.3: Formal Method Tools Comparison ............................................. 15
3.1: Backus-Naur Form ................................................................. 26
3.2: Multiplicity and Ordering ....................................................... 33
4.1: Mapping VB data types to VDM++ ............................................. 41
4.2: Mapping UML Attributes to VDM++ instances variables ............ 43
4.3: Multiplicity ......................................................................... 44
4.4: Mapping Values to VDM++ ..................................................... 47
4.5: Mapping UML operation signatures to VDM++ operation signatures .... 50
5.1: The operations of a class ....................................................... 61
5.2: The operations of class VDM++Class ..................................... 61
6.1: Campaign management Use Case .......................................... 81
6.2: Requirement Specification .................................................. 83
6.3: Operation to add new client ................................................. 84
7.1: UML specifications ............................................................. 101
List of Figures

3.1: Campaign System with Extends Association ........................................... 21
3.2: Class Diagram ...................................................................................... 22
3.3: Prototype architecture ........................................................................ 35
4.1: Mapping UML class name to VDM++ class name................................. 39
4.2: Type Definition and Invariant ............................................................... 40
4.3: Attribute Syntax .................................................................................. 41
4.4: Instance variable definition ................................................................. 42
4.5: Instance variables ............................................................................... 42
4.6: Window to specify multiplicity and ordering ......................................... 48
4.7: Parameters list specification ................................................................. 50
4.8: Generated input parameters corresponding to parameters list ............. 51
5.1: The process flow of the prototype ....................................................... 54
5.2: Use case models of the prototype system ........................................... 55
5.3: The sequence diagram for Create Static OO Model............................. 56
5.4: The Use Case Generate XML Internal Representation ....................... 57
5.5: The Use Case Generate VDM++ Specifications .................................... 58
5.6: Static UML OO model .................................................................... 59
5.7: Static UML model ............................................................................... 60
5.8: Generated XML for UML Attributes .................................................. 66
5.9: Generated XML for UML Operations ................................................. 67
5.10: Main User Interface ........................................................................ 69
5.11: Entry and Exit Interface ................................................................. 72
5.12: Top-level states ............................................................................... 73
5.13: The class specification in Tree View ................................................. 75
5.14: Multiplicity and Ordering from ListBox ............................................ 76
6.1: Proposed prototype architecture ....................................................... 77
6.2: Campaign Management .................................................................. 81
6.3: Class Diagram with Composite data type ......................................... 82
6.4: Interface Window to capture UML specification ................................ 85
6.5: Interface Window with Treeview sub window .................................. 86
6.6: Interface Window for UML Diagram ................................................ 87
6.7: Interface Window for XML .............................................................. 88
6.8: Interface Window for VDM++ Code
6.9: Step to Create Composite data type
6.10: User Define Data Type
6.11: User Define data type testing
6.12: Value definitions based on Property string frozen
6.13: The combination of Multiplicity and Ordering to produce set or seq
6.14: VDM++ Toolbox interface
7.1: Transformation of UML specifications to VDM++
7.2: User Define data type interface
7.3: Verify of Type definitions using VDM++ toolbox Lite v8.1
7.4: VDM++ specifications Generated from UML specifications in Table 7.1
Abstract

The goal of this thesis is to generate Formal Method (FM) specifications using the Unified Modeling Language (UML), class diagram models. In this context, we use the Vienna Development Method for modeling object-oriented models (VDM++) as a formal specification language. We studied the syntax and semantic of UML and VDM++ models and then defined the mapping rules that were used to transform UML models into VDM++ specifications. To achieve this purpose, we present a framework of the prototype that automatically generates UML models into VDM++ specifications. The prototype derives UML properties of a class diagram and uses the translation rules to construct VDM++ class, types, values, instance variables and operation signatures. We proposed a rule-based to construct Set and Sequence in VDM++ based on Multiplicity and Ordered/Unordered properties of UML model. We also developed an interactive interface to construct composite data type to model structured data such as record. A transformed model from UML into VDM++ specification then validated and checked using VDMTool Lite v8.1.
Abstrak

Chapter 1: Overview

1.1 Background

The software development is typically challenging in the early stage because there are too many complex requirements to be taken into account. Dealing with these uncertainties from various perspectives and at the same time to increase the reliability in the design of the system makes system development a very difficult task [4]. One of the possible solutions to minimize this complexity in software development is by using formal methods with the combination of UML integration [2]. The combination of the two methods may provide quality improvements through reduce software errors, reduce software development costs, and time [7].

The Unified Modeling Language (UML) [8] is described as a “general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of a software system”. Formal Methods (FM) specify requirements using mathematical notations which can be analysed to prove correctness and consistency [10]. FM can also be used to verify that an implementation is consistent with its specification. However, the development of formal model is currently time consuming and expensive. It requires expertise and extensive training [5].

A large number of tools that support formal method have been developed. For example the Vienna Development Method (VDM and VDM++) is supported by VDM++ Toolbox [9], Event-B is supported by Rodin platform [11] and Z specification language is supported by Z language Type Checker (ZTC) [4]. So far, FM is being used for safety critical systems such as aircraft engine control, nuclear power plant and railroad interlocking [13]. Therefore the availability of high quality FM tools is essential to reduce human error [12] and to promote their usefulness in industry [13].
1.2 Objectives

Past literatures have shown that VDM++ operation signatures can be transformed from UML diagram [5]. This thesis presents an approach to translate UML static model, class diagram into formal VDM++ specification. A prototype system was then developed to translate and generate UML model into VDM++ specification language. Particularly, we explored the following objectives.

- To capture the UML static model specifications, this includes class, attributes and operation signatures.
- To define VDM++ static model including class definitions, type definitions, value definitions, instance variable definitions and operation signature definitions.
- To validate and verify correctness of VDM++ static model definitions using VDM++ Toolbox Lite v8.1.

We identified rules to transform UML properties into VDM++ models. Finally, VDM++ Toolbox Lite v8.1 was used to verify and validate the models.

1.3 Problem statement

In recent years, we have seen the emergence and development of methodologies and CASE tools that attempt to support, facilitate and manage the software development process. UML is commonly regarded as one of the most popular graphical notations for software system modeling [8]. However, UML does not emphasis on verification and correctness of the software. UML based development starts with a specification of required system functions. Without rigorous specification technology, it is difficult to devote time and effort to the specification process [10]. In UML, specifications are normally written in natural language, with inevitable ambiguities.
In response to this problem, mathematically based approaches known as formal methods (FM) have been developed for the specification, development and verification of software and hardware systems [7]. The purpose of a FM is to provide an unambiguous notation that can be validated [2]. FM uses a mathematical and logical formalization to prove that the key properties of the system satisfy the expected behavior of the software system. It is important to develop a linking tool from UML to FM. The benefits of integrating UML with FM are [14]:

- Logic is able to define statements that consider all possible input values. This is significantly better than unit tests, which are usually able to test just a small subset of input data.
- Design choices can be formally verified before any implementation.
- Correctness verification can be done automatically through theorem provers and model checkers.
- Changes in software specifications can be handled more easily.

It is clear that software development community should not ignore the advantages it may gain by integrating FM and UML.

1.4 Purpose of study

The main purpose of this thesis is to develop a tool that should able be to capture and manipulate the UML details and store it as a XML internal representation. The tool should be able to manipulate inadequate details of UML to make it closer to VDM++ semantics, therefore the VDM++ specifications can be generated from the XML internal representation of UML models. The tool will be developed using Visual Basic 2005 platform and the internal representations are represented by using Extensible Markup Language (XML).
1.5 Research Contribution

The contribution of this thesis is to define mapping rules from UML models to VDM++ language specifications. Particularly, to define mapping rules for translating instance variable definitions of type set and sequence based on the UML attribute properties, multiplicity and ordered/unordered. It also proposed a mapping rule to define value definitions using frozen property of UML model. Finally, we developed a simple and good interface, User Define Data Type to create composite data type such as record. We also developed an interface that automatically generates parameters list for operation signature. This function is able to minimize the difficulties to recall what parameters have inputs into the system.

1.6 Scope of Work

This thesis mainly concentrates on transformation of UML model to VDM++ language specification and then using VDMTools to validate and verify the correctness of implementation rules. As this is an extended work to Edwin (2008) who proposed "Developing VDM++ Operations from UML Activity Diagrams", we also use the same mapping rules for some of the UML model properties to VDM++ language specification. As the previous work mainly focused on dynamic behavior of UML models and translates the models into VDM++ language specification, in this study we expand this by exploring the static model of UML class diagram to VDM++ specification.

1.7 Chapter overview

Chapter 1 gives a general overview of the study carried out for this dissertation. Chapter 2 describes the literature review in the areas of UML and Formal Method integration. Chapter 3
presents the features of UML and VDM++ intended for use in the prototype. Chapter 4 presents the translation process from UML to VDM++ specification. Chapter 5 describes the methodology to be used in developing the system prototype. Chapter 6 describes the testing process and evaluation of the prototype based on translation rules defined in chapter 4. And finally, Chapter 7 concludes the contributions of the study and provides ideas for further works.
Chapter 2: Literature Review

2.1 Introduction

This chapter describes relevant literature reviews and concepts in the integration of object-oriented approach and formal method tools. It also provides a brief comparison of object-oriented approaches and features of formal method tools. The chapter ends with a brief evaluation and conclusion of relevant works in integrating both methods.

2.2 Object-Oriented (OO) Tools

Due to the popularity of object-oriented programming in the early nineties, a growing number of software methodologies have appeared to suit object-oriented software development [39]. Object-oriented (OO) approaches have already been applied to programming languages, information systems, simulation and artificial intelligence. At the same time, an OO approaches are supported by many CASE tools and has gained popularity among software developers. Some of important features of object-oriented approach are their support of abstraction to deal with complexity, friendliness, maintenance, support for reuse, and easy transition from analysis to design [15]. The beauty of OO tool is that it captures requirements using a combination of text and graphical notation making them easy to understand and to use [2].

During the eighties, a number of OO analysis and design methodologies and notations were developed by different research teams [6]. Computer Aided Software Engineering (CASE) tools such as Rational Rose [16] and ArgoUML [17] were developed to support many existing notations used such as the Unified Modeling Language (UML). To which, UML become the standard notation for OO methods [19].
Programming language support for objects began with Simula 67, and then progressed through purely OO languages such as C++, Object Pascal, Java, Visual Basic, and Smalltalk. Most of these programming languages support OO features such as classes and objects, inheritance, and polymorphism which can reduce software complexity. However, the major shortcoming of OO approaches is their lack of formal semantics [41].

This section describes briefly the usage of UML tools and its features.

2.2.1 Rational Rose 2002

Rational Rose can be used to make visual models of software using UML diagrams to capture various aspects of software development. It supports the UML 1.x diagram types, use case diagrams, class diagrams, activity diagrams, sequence diagrams, collaboration diagrams, state chart diagrams, component diagrams and deployment diagrams. It supports a variety of implementation languages, including C++, Java, PowerBuilder, Smalltalk, and Visual Basic. Rational Rose can be used to design and create various features in UML models. Table 2.1 lists some of the significant UML modeling features allowed by the Rational Rose 2002: [20].

<table>
<thead>
<tr>
<th>UML feature</th>
<th>Rose 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML 2.0</td>
<td>No</td>
</tr>
<tr>
<td>Hyperlink from UML Diagram and Elements to URLs</td>
<td>Limited, a URL may be visited from the explorer view rather than from the diagram editor.</td>
</tr>
<tr>
<td>Hyperlink to other UML diagrams and elements</td>
<td>Yes, through shortcuts on attached comments.</td>
</tr>
<tr>
<td>Convert sequence diagrams to collaboration diagrams and vice-versa</td>
<td>Yes</td>
</tr>
<tr>
<td>Create diagram elements from explorer tree</td>
<td>Yes</td>
</tr>
<tr>
<td>UML tagged values</td>
<td>No intelligent support</td>
</tr>
<tr>
<td>UML constraints</td>
<td>No intelligent support, extendable through</td>
</tr>
</tbody>
</table>
Table 2.1: UML and Rational Rose 2002 [40]

<table>
<thead>
<tr>
<th>Feature</th>
<th>Extensibility features and partner products</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML standard stereotypes</td>
<td>Yes</td>
</tr>
<tr>
<td>User defined stereotypes</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom Icons for stereotypes</td>
<td>Yes, some pre-defined, additional set in INI file</td>
</tr>
<tr>
<td>Additional Diagram Types shipped with product</td>
<td>No</td>
</tr>
<tr>
<td>Add custom diagram types</td>
<td>Limited, by creating and editing configuration files</td>
</tr>
<tr>
<td>Business Modeling extensions</td>
<td>Yes, through use of predefined stereotype and in conjunction with the RUP which also explains the business modeling discipline workflow</td>
</tr>
<tr>
<td>Data Modeling</td>
<td>Yes, through stereotypes and data modeler extension</td>
</tr>
<tr>
<td>Activation (Programmable extension of functionality)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

2.2.2 Unified Modeling Language Tool: ArgoUML

ArgoUML [18] is conceived as a tool and environment for use in the analysis and design of object-oriented software system. It draws on research in cognitive psychology to provide novel features that increase productivity by supporting the cognitive needs of object-oriented software designers and architects.

ArgoUML is an open source CASE tool that supports most UML diagrams [17]. It supports open standards extensively (e.g., OMG standard for UML, XMI), provides full Object Constraints Language (OCL) syntax and type checking, supports code generation for Java, C++, C), PHP4 and PHP5, can be run on any platform that support Java5 and Java6, and provides cognitive support.
2.2.3 Object-Oriented Tools Comparison

Object-orientation has reached a state of maturity and some useful techniques have been standardized which has resulted in the specification of the UML and CASE tools that support UML. However, there are some limitations to current existing OO tools. Based on Table 2.2, you can see that all these tools are able to generate source from UML models. However, there some limitations and additional works are required to complete the development process of software system.
<table>
<thead>
<tr>
<th>Name</th>
<th>Creator</th>
<th>Platform / OS</th>
<th>Open source</th>
<th>Programming language used</th>
<th>Approach</th>
<th>Languages generated</th>
<th>Reverse engineered languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgoUML</td>
<td>Tigris.org</td>
<td>Java (cross-platform)</td>
<td>Yes</td>
<td>Java</td>
<td>None</td>
<td>C++, C#, PHP4, PHP5</td>
<td>None</td>
</tr>
<tr>
<td>BoUML</td>
<td>Bruno Pagès</td>
<td>C++/Qt (cross-platform)</td>
<td>Yes</td>
<td>C++</td>
<td>MDA, template</td>
<td>Java, C++, PHP, Python, IDL</td>
<td>Java, C++, PHP.</td>
</tr>
<tr>
<td>Dia</td>
<td>Alexander Larsson/GNOME Office</td>
<td>GTK+ (cross-platform)</td>
<td>Yes</td>
<td>None</td>
<td>Java, C++, ADA (using dia2code)</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Eclipse UML Tools</td>
<td>Eclipse Foundation</td>
<td>Java (cross-platform)</td>
<td>Yes</td>
<td>Java</td>
<td>None</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>StarUML</td>
<td>Plastic Software</td>
<td>Windows</td>
<td>Yes</td>
<td>Delphi</td>
<td>Plug-in architecture: C++, Delphi, C#, VB,</td>
<td>None</td>
<td>C#</td>
</tr>
<tr>
<td>Umbrello UML Modeller</td>
<td>Umbrello Team</td>
<td>Linux</td>
<td>Yes</td>
<td>C++, KDE</td>
<td>None</td>
<td>C++, Java, Perl, PHP, Python... 16</td>
<td>C++, IDL, Pascal/Delphi, Ada, Python, Java; import XMI, RoseMDL</td>
</tr>
<tr>
<td>Frame UML</td>
<td>Frame</td>
<td>Windows</td>
<td>Yes</td>
<td>C++</td>
<td>A UML tool, support UML2.xx, and embed JavaScript, so you can generate source code from model by JS</td>
<td>Almost any language you want if you can write JavaScript to generate it.</td>
<td>Java (partial), but you can use JavaScript to reverse other languages to model</td>
</tr>
</tbody>
</table>

Table 2.2: Object-Oriented Tools Comparison [42].
2.3 Formal Methods and Tool Support

FM is mainly used for the development of safety critical systems such as flight control and medical systems [43]. They are not widely used as they are expensive and difficult to use because they rely heavily on mathematics and logic. To encourage the use of FM, many CASE tools have been developed to support the various phases of a formal software development process. For example the Vienna Development Method (VDM and VDM++) is supported by VDMTools [9], B language is supported by B-Toolkit and Z language is supported by Z Tool Checker (ZTC) [4].

Most of the tools have the capability to generate source code from formal specifications. This section describes the application of FM tools, the advantages and weaknesses. It also discusses the integration of different formal methods with UML for different aspect of software design.

2.3.1 Formal Method – Vienna Development Method (VDM) language

There are three platforms in the Vienna Development Method [9].

- Vienna Development Method-Specification Language (VDM-SL)
  - Provides facilities for the functional specification of sequential systems with basic support for modular structuring.

- VDM++
  - Object-oriented modeling and concurrency.

- VICE (VDM++ In Constrained Environment)
  - Real-time computations and distributed systems.
2.3.1.1 VDM Tools

VDMTools supports software development based on the specification written by formal specification language, VDM-SL or VDM++. VDMTools Lite v8.1 provides the various features described below [9].

- Syntax checker
- Type checker
- Integrity Examiner
- Interpreter and debugger
- Test coverage statistics tool
- Rose - VDM++ link
- Pretty Printer
- VDM++ to C++ code generator (Optional)
- VDM++ to Java code generator (Optional)
- Java to VDM++ generator (Optional)
- CORBA Compliant API (Optional)

For example, in syntax checking, it will produce a list of error messages and indicate the position of the error in the specifications source files. This allows for stepping through the syntax of error found.

VDMTools have been applied in a various application systems;

- T. Kurita et al. developed an operating system for an integrated circuit of cellular telephone applications [38].
- E. Mit developed a FOTool application to generate specifically a VDM++ body operation from a UML model and then used the Toolbox to validate and verify the translation rules [5].