A Formal Modeling Tool for Exploratory Modeling in Software Development

Tomohiro ODA†††, Member, Keijiro ARAKI††, and Peter GORM LARSEN††††, Nonmembers

SUMMARY The software development process is front-loaded when formal specification is deployed and as a consequence more problems are identified and solved at an earlier point of time. This places extra importance on the quality and efficiency of the different formal specification tasks. We use the term “exploratory modeling” to denote the modeling that is conducted during the early stages of software development before the requirements are clearly understood. We believe tools that support not only rigorous but also flexible construction of the specification at the same time are helpful in such exploratory modeling phases. This paper presents a web-based IDE named VDMPad to demonstrate the concept of exploratory modeling. VDMPad has been evaluated by experienced professional VDM engineers from industry. The positive evaluation resulting from such industrial users are presented. It is believed that flexible and rigorous tools for exploratory modeling will help to improve the productivity of the industrial software developments by making the formal specification phase more efficient.

**key words:** lightweight formal methods, formal specification, VDM, integrated development environment

1. Introduction

The lightweight use of formal methods has been proposed quite some time ago as a cost-effective way to enter industrial use [1]–[4]. The Vienna Development Method (VDM) [5] contains different specification language dialects, known to support lightweight formal methods. In this paper, we will concentrate on the VDM Specification Language (VDM-SL) which is an ISO standard [6].

The front loading effect, putting more efforts in earlier phases of software development to reduce the total costs and efforts, is often observed in industrial case studies. A better abstraction of the problem domain and the system in the specification phase reduces cost of fixing errors caused by the earlier phases. Improved clarification of requirements and rapid feedback from customers and domain experts raise confidence in specification artefacts, shorten time to market and reduce maintenance costs. Development tools for the formal specification phase are the key to successful development. Integrated Development Environments (IDEs) for formal specifications, such as VDMTools [1], [7] and the Overture tool [8], provide specification editors, type checkers, proof obligation generators, proof assistants, specification animators and diagram generators. They are designed to produce and analyse rigorous specifications throughout the formal specification phase and offer many different features.

In order to produce a good specification, a clear understanding of the problem domain is essential. The specification engineers need to learn the domain first when they work on a software development task in a new problem domain. Learning efforts could take a long time, and the quality of their understanding might be insufficient at the beginning. The understanding of the problem domain is gradually developed during the software development by reflecting on the consequences of the specification artefact [5]. We are interested in the earlier stage of the specification phase where learning efforts with trial-and-error in modeling take place, which we call the exploratory modeling phase. The exploratory modeling involves face-to-face pair working with domain experts. We developed an easy-to-use web-based IDE named VDMPad as a proof-of-concept of exploratory modeling.

In this paper, the formal specification language VDM-SL is briefly introduced in Sect. 2. The concept of exploratory modeling and its requirements to support tools are explained in Sect. 3, and a tool, named VDMPad, that demonstrates the exploratory modeling is described in Sect. 4. Afterwards, VDMPad is evaluated in Sect. 5. Section 6 covers related work including Lively Walk-Through, PVSio and Overture tool. Finally, Sect. 7 completes this article with concluding remarks.

2. VDM Specification Language

VDM-SL is a model-oriented formal notation based on discrete mathematics and logic that supports the description of both data and functionality. Data are defined by means of types built using constructors that define structured data with records and tuples and collections such as sets, sequences and mappings from basic values such as Booleans and numbers. These types are very abstract, allowing the user to add any relevant constraints as data type invariants. Constant values can also be defined. Functionality is defined in terms of functions and operations over these data types. These can be defined implicitly by preconditions and postconditions that characterise their behaviour, or explicitly by describing how the input is transformed into the desired output. The distinction between functions and opera-
types
Item = <BEER> | <WINE>;
Bag = map Item to nat1;
Order = seq1 of (Item * nat1);
values
empty :Bag = { |-> };
state Inventory of
stock : Bag
init s == s = mk_Inventory(empty)
end
operations
Buy : Order ==> ()
Buy(order) == for mk_(item, num) in order do
  let current_num =
  if item in set dom stock
  then stock(item)
  else 0
  in stock := stock++{item|->current_num+num};
Sell : Order ==> ()
Sell(order) == for mk_(item, num) in order do
  let new_num = stock(item) - num
  in stock :=
  if new_num > 0
  then stock ++ { item |-> new_num }
  else {item} <-: stock;

Fig. 1 An example VDM-SL specification of liquor shop

3. Exploratory Modeling

Software development starts with incomplete and informal requirements. There is no rigorous formal specification at the beginning. The front-loading effect of formal methods increases importance of the specification phase. The quality and productivity of the specification phase greatly impact on the later phases because many decisions are made in the specification phase. This section explains what exploratory modeling is in the specification phase and requirements for its support tools.

3.1 Exploratory Modeling in Earlier Formal Specification Phases

Developers often need to gain domain knowledge required to model the desired system. Indeed, the formal specification phase contains a pair of parallel processes; building and learning. Developers build a tentative model based on their understanding of the domain, and learn the domain more precisely through the model by analyzing and getting feedback from domain experts and other stakeholders to build a more precise model. We named the earlier stage of the formal specification phase exploratory modeling characterised by the exploration over the target domain driven by the tentative models. Figure 2 shows the exploratory modeling phase in the software development process.

The specification shown in Fig. 1 is the final result of an exploration. The requirement is simple: “We are running a liquor shop. We want to manage our stock.” Many things are uncertain. The VDM model below shows the very initial specification from the requirement which could be made as the first step of an exploration.

Note that the Sell operation may not behave as a customer would expect but this is exactly the kind of error that you would like to gradually explore at earlier stages of the formal specification phase.
This specification is incomplete and syntactically incorrect. However, it conveys the vague understanding of the specification engineer at the beginning. The specification engineer explains the specification to the client and the client responds, “We do not operate the stock management by single items. We manage by order sheets with items and quantities.” Then, the specification engineer makes the following modification to the specification as a new step in the exploration.

```
R1: The tool shall enable direct interaction with smaller
models
```

The new specification is explained to the client, but the client is not satisfied yet. The client says, “Buy is OK, but Sell is too naive. We sell bottles listed in an order until we hit an empty item in the stock.” The specification shown in Fig. 1 is the result of this pair work session in the exploratory phase. The exploratory modeling is the process to find these possible alternatives and make choices referring to feedback from domain experts.

At some point, refining the formal model to make it more rigorous and more precise takes over from the exploration (See Fig. 2). We call this phase rigorous modeling. In the liquor shop example, we will strictly define preconditions, postconditions of each functions and operations, and declare invariants to types and state if needed, and prove that all operations and function calls are type-safe and satisfy preconditions and postconditions. Most formal methods research pays attention to the rigorous specification phase. We shed light on the exploratory modeling phase. We consider that the productivity and quality of the exploratory modeling phase greatly impacts on the later phases because the quality of the resulting formal specification relies on precise knowledge about the domain and the tentative model as the seed of the rigorous formal specification.

### 3.2 Requirements to Exploratory Support Tools

Tools supporting such an exploratory phase do not have exactly the same requirements as those for rigorous specifications. Below we explain the key requirements we find most important for tool support for such exploratory activities.

```
R1: The tool shall enable direct interaction with smaller

The model is typically small and monolithic at the beginning, and then grows larger and possibly spreads over multiple modules. The exploratory modeling is not only to write a formal model from dialogues with domain experts, but to learn from and improve tentative models. This process can be seen as an interaction between the formal engineer and the model at hand. The quality of interaction affects the efficiency of the exploratory modeling task.

What constitutes a good interaction between formal specifiers and tentative models? A good editor is not enough. A user interface that gives direct “touch” of the model is required.

```
R2: The tool shall be understandable by stakeholders with no formal methods background

```
R3: The tool shall enable choosing permissive checking by choice

```
Strict and powerful checking on the model is definitely a strength of formal methods. However, exploratory modeling handles tentative, immature, often inaccurate models. Permissive checking rather than fully strict checking is sometimes useful in the exploratory modeling phase. It should be also possible to enforce strict checking because models passed to the rigorous modeling phase should not be error prone.

```
R4: The tool shall provide continuous analysis

```
### 4. VDMPad

VDMPad[10], [11] is a simple Web IDE for the formal specification language VDM-SL and is designed to support exploratory modeling. A VDMPad server provides a VDM animation service to web clients for an executable subset of VDM[12]. It is designed to supply a lightweight, handy-to-use functionality comparable to a calculator in an engineer’s

† A public server of VDMPad is available at http://vdmpad.csce.kyushu-u.ac.jp/.
In this section, we will explain the architecture of VDMPad and introduces the features of VDMPad for friendliness to new VDM users and for exploratory development.

4.1 Architecture

Roughly speaking, VDMPad adds a flexible and easy-to-use interaction to a command line VDM interpreter, as well as intuitive presentation of the model by syntax highlighting, pretty printing and diagrammatic representation of internal state. A VDMPad server provides a Web API called eval and an interactive user interface. An eval API call is dispatched with a VDM specification, bindings of state variables, and an expression to evaluate, and returns the specification, new bindings of state variables, the evaluated expression, the return value and an error message from the interpreter if relevant. The eval API is stateless and sessionless. The current animation state, which consists of the VDM specification, the bindings of state variables, the return value and the error message, is not stored in the VDMPad server, but in the web browser’s persistent memory called localStorage. Each evaluation is thus isolated. The isolation of the evaluation at the server side enables flexible user interactions at the client side.

The user interface of VDMPad stores the animation status returned from the eval API, renders the animation status, accepts modification on the animation status from the user, and dispatches the next eval API. This cycle is similar to the Read-Eval-Print Loop (REPL) based on console-based interfaces. VDMPad in addition provides more flexible input from a user. The user can edit the value of each state variable, and a modification to the specification does not require re-initialisation of the state. VDMPad also displays graphical diagram representations of VDM expressions for intuitive readability. The details of flexible user interaction and diagram representations will be described below.

4.2 Simple UI Designed for Exploration Tasks

All interactions with VDMPad are performed via the same simple page. Figure 3 shows the major UI components of VDMPad. The user interface of VDMPad is very simple and minimal to interact with the small-scale specifications. The simple UI of VDMPad is designed to satisfy R1 discussed in Sect. 3.2.

The user interface of VDMPad consists of five parts; a specification editor, an area for information about the state, a workspace, a return value display and finally a message area. In addition, a retractable menu pane is provided.

The specification editor has a syntax highlighting feature using the open-source code editor, CodeMirror, to help the user see the structure of the specification. It also has a pretty printing feature to reduce the burden of writing the specification. The user does not have to save the specification to animate it. Evaluating an expression in the workspace will send the specification to the server to animate it, and the specification will be saved on the local storage in the browser automatically along with the content of the workspace and other information about the status of the animation.

VDMPad displays the value of each of the state variables grouped by modules in the state area. The value of each variable is printed in an input field so that the user can edit the value. VDMPad can optionally display a diagram representation of the value of each state variable for intuitive understanding of the data structure of the value. For example, the state variable stock is a mapping from symbols to natural numbers. Details will be explained in Sect. 4.4.

The workspace is a text editor named after its counterpart the Smalltalk environment [13]. The main purpose of the workspace is to write and organise a collection of expressions to evaluate, but the user can also write notes inside this area. The user can evaluate such an expression by selecting it on the workspace and clicking on the evaluate button (See Fig. 4). The return value of the expression is displayed below the evaluate button. The return value can also be displayed in a diagram representation. If the expression or the specification has a run-time error, the error message is displayed in the message area at the bottom of the window.

REPL interpreters require users to type and/or edit one or more lines of expressions to perform animation steps. The Overture tool provides REPL interpreter and also debugging interface for rich debugging features such as break points and step-wise execution. VDMPad is oriented to simplicity and flexibility in contrast to rich functionality. The
workspace of VDMPad enables the user to maintain multiple expressions and select a fragment to be evaluated in the workspace. The workspace is not only used to save typing effort but also to give more freedom to organise expressions to be evaluated than REPL on a console.

4.3 Liveness

The basic design of the VDMPad user interface aims at frequent interactions to benefit lightweight analyses such as syntax checking, type checking, dynamic assertion checking and unit testing. Liveness is a property of programming environments that a program code can be modified while running and the program can continue running with the modified code. VDMPad adopts liveness to support R1 while other tools such as VDMTools and the Overture tool do not.

VDMPad does not abort its animation when the user edits the specification. Most other conventional specification animators initialise the state when a specification is modified; the old animation session is closed and then a new animation starts with the new initial state defined in the modified specification. VDMPad tries to continue the current state with a modified formal specification unless the state violates any state invariant in the modified specification (see Fig. 5).

Another live feature implemented in VDMPad is direct modification of the state by the user during an animation session. The user can edit the values of state variables without executing an operation. This feature enables the user to explore behaviour of the system based on an arbitrary hypothetical state without having to define and call different operations for setting such state components. This is one of the major differences from other IDEs for VDM such as the Overture tool. The Overture tool shows values of state components during animation, but does not allow the user to directly modify them.

4.4 Diagram Representations of VDM Values

One complex expression in VDM can be hard to read especially for those who are not familiar with expressions in formal grammars. Graphical expressions in the form of diagrams can be helpful for stakeholders with little VDM background. VDMPad displays VDM-SL expressions in diagram representation using the “visual presentation” option. Diagram presentations appear in the state area and return value in VDMPad. In Fig. 3, a diagram presentation of stock is displayed in the state area. Diagram representation helps the user to read complex values. This feature is introduced to support R1 and R2 and is dedicated to VDMPad. Legends of diagram representation is shown in Fig. 6.

4.5 Unit Testing

Unit testing is a common practice in systematic specification and implementation phases that contributes to R4. The VDM-family of specification languages has a framework for unit testing named VDM-unit [14]. VDM-unit is good for testing a rigorous specification with large number of test cases. However, in the exploratory specification phase, lighter testing framework for smaller test cases can make quicker feedback to developers.

Unit testing on VDMPad is performed at every click of the evaluate button when the unit testing feature is turned on. Because every animation step, including modification to the specification and the state, is triggered by the evaluate button, unit test is performed at every modification to the specification and animation step. A test case is a quadruple of pre-state, expression, return value and post-state. Continuous performing of unit testing helps the user to watch over unexpected ripple-effects caused by modification during incremental and exploratory development. The
results of tests are displayed below the message area. Clicking on a test result expands the test case to its quadruple of pre-state, expression, return value and post-state. Failed test cases are printed in red (see Fig. 7).

It is simple to define and perform a test case on VDMPad. The user can create a test case by clicking the make it a test case button that appears under the evaluate button. VDMPad composes a test case of pre-state, expression, return value and post-state from the previous evaluation.

4.6 Enable/Disable Runtime Checking

VDMPad provides an option to the user to determine whether assertions and type checking at runtime shall be performed or not, as described in R3. The user can experiment with a potential behaviour of the system without guards when the user is gaining confidence in the validity of neither the explicit definitions nor assertions. This feature should be used with the greatest caution and must be turned off as soon as the user understands the essential behaviour of the system.

5. Evaluation

We conducted a user evaluation by using questionnaires for three industrial VDM experts who have used VDMPad for two years or longer, and have even longer and deeper experience on VDMTools and the Overture tool. The questionnaire consists of four groups of questions: (a) usability, (b) comparison between smaller-scaled and larger-scaled specification, (c) comparison between the initial stage and the final stage of the specification phase and (d) any comments in free format. The expression “earlier stages” corresponds to the exploratory modeling, and the expression “later stage” corresponds to the rigorous modeling. By the expression “smaller specs” we mean at most two pages of VDM specification which we expect in exploratory modeling.

Each question in groups from (a) through (c) asks whether VDMPad is better than other IDEs in each specific aspect, and answer was chosen from 1. yes, 2. weakly yes, 3. equal, 4. weakly no, 5. no, or 6. other. Please note that the questions are comparison between VDMPad and other IDEs. The questions and average scores of the results are summarised in Table 1.

VDMPad is designed for exploratory modeling at earlier stages of the specification phase with smaller models. Questions a-1 through a-5 are evaluations to design and implementation as a modeling tool in the aspects of R-1 through R-4 described in Sect. 3.2.

VDMPad is evaluated better than other tools in questions a-1, a-2, a-3, a-4, b-1, c-1, c-3, c-5 and c-7. Under the condition that the subjects were more used to other tools, the results are quite positive although none of the questions got “1. yes” from all subjects.†

The average results from questions b-2, c-2, c-4, c-6 and c-8 show that existing tools perform better then VDMPad. It is understandable because these questions are not for evaluating tools for exploratory modeling, but for rigorous modeling. The average score from these questions illustrates that VDMPad is specialised to exploratory modeling rather than for general purpose. Our data mentions that VDMPad works well in exploratory modeling (b-1, c-1, c-3, c-5 and c-7) sacrificing its general purpose usability.

As a summary of the user evaluation, in general VDMPad was positively accepted by experts of VDM as a tool dedicated to exploratory modeling.

6. Related Work

The exploratory modeling have similarity with prototype-based approach when the specification language or its subset is executable. The resulting specification by the exploratory modeling phase is written in a formal notation and should be rigorously analysed, reviewed and brushed up.

Intensive changes are often made in the exploratory modeling phase. Flexibility is a key to successful exploration. Agile methods similarly give importance to flexibility. VDMPad borrowed many concepts and techniques, such as liveness and workspace, from the Smalltalk environment, which is one of the most successful platforms for Agile developments. Some practices encouraged in Agile methods are also considered effective in exploratory modeling. However, the exploratory modeling is only a part of the formal specification phase while most Agile methods cover the entire development. Three tools that are related to or compared with VDMPad are described below.

<table>
<thead>
<tr>
<th>question</th>
<th>score</th>
<th>associated requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-1: overall usability</td>
<td>2.7</td>
<td>R1, R2, R3, R4</td>
</tr>
<tr>
<td>a-2: UI usability</td>
<td>2.7</td>
<td>R1</td>
</tr>
<tr>
<td>a-3: live usability</td>
<td>2.0</td>
<td>R1</td>
</tr>
<tr>
<td>a-4: understandable by non-experts</td>
<td>1.7</td>
<td>R2</td>
</tr>
<tr>
<td>a-5: validation</td>
<td>3.0</td>
<td>R2, R3, R4</td>
</tr>
<tr>
<td>b-1: for smaller specs</td>
<td>1.7</td>
<td>R1</td>
</tr>
<tr>
<td>b-2: for larger specs</td>
<td>4.3</td>
<td></td>
</tr>
<tr>
<td>c-1: at earlier stages</td>
<td>1.7</td>
<td>R1, R2, R3, R4</td>
</tr>
<tr>
<td>c-2: at later stages</td>
<td>3.7</td>
<td></td>
</tr>
<tr>
<td>c-3: animation at earlier stages</td>
<td>1.3</td>
<td>R1</td>
</tr>
<tr>
<td>c-4: animation at later stages</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>c-5: analysis at earlier stages</td>
<td>1.3</td>
<td>R3</td>
</tr>
<tr>
<td>c-6: analysis at later stages</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>c-7: unit testing at earlier stages</td>
<td>2.0</td>
<td>R4</td>
</tr>
<tr>
<td>c-8: unit testing at later stages</td>
<td>4.0</td>
<td></td>
</tr>
</tbody>
</table>

†We estimate that this is caused by the fact that when one is very used to use a tool everything feels easier with that.
6.1 Lively Walk-Through

Lively Walk-Through is another tool to support exploratory modeling. Lively Walk-Through is a UI prototyping environment where formal engineers and UI designers collaboratively develop common understandings and agreements by creating, operating and discussing a UI prototype.

Lively Walk-Through is a live environment where the user can modify the specification while the UI prototype is running. The user of the UI prototype does not need to know the syntax of VDM-SL, but can operate on GUI widgets such as labels, text fields, graphics image, graphs and dialogs with natural language texts. Lively Walk-Through thus support R1 and R2 described in Sect. 3.2.

VDMPad can be used in conjunction with Lively Walk-Through. Lively Walk-Through assumes that the user has a specification at hand. One can create a tentative specification on VDMPad and then give it to Lively Walk-Through to check what functionality is needed by creating a prototypical user interface. Although VDMPad can show diagram representation of VDM values, GUI prototypes built on Lively Walk-Through can be more expressive and may be closer to the user interface of a deliverable product.

6.2 PVSio-Web

PVSio-web [15] is a high fidelity UI prototyping tool to animate a formal specification generated from a graphical diagram notation of Emucharts. A VDM-SL generator from Emucharts was also developed [16]. Like Lively Walk-Through, PVSio-web supports validating formal specification by domain experts. PVSio-web is not explicitly designed in the context of exploratory modeling. PVSio is based on state machines which is a different technique from VDM. It has graphical presentation of the model. PVSio has technical elements useful for validation in exploratory modeling when a state machine is used as a model of the system but thus it is substantially different from VDMPad since the VDM model is entirely generated.

6.3 The Overture Tool

The Overture tool [8] is a fully-fledged IDE with many different features supporting development, translation and validation and verification (see Fig. 8). It is developed on top of the generic Eclipse platform. The Overture tool has practical functionalities for professional work with large VDM specifications, including code generation, test automation and test coverage as well as clarification of potential places where run-time errors could occur with a proof obligation generator. The Overture tool can be considered a reference of conventional IDEs to be compared with support tools specialised for exploratory specification.

A prototype WebIDE for the Overture tool has been developed [17]. The Overture’s WebIDE and VDMPad have different orientations inherited from their base environment. Overture’s WebIDE has a file-based user interface based on the Eclipse-based standalone Overture tool, and provides functionality using components of the Overture tool. VDMPad has an image-based user interface which does not employ the concept of files except importing/exporting snapshots from/to local file systems. VDMPad is designed to provide personal modeling environments for exploratory modeling tasks while Overture and its WebIDE pursue rigorous modeling in either personal or team-based developments.

Since the rigorous modeling phase follows the exploratory modeling phase, tools for exploratory modeling should be able to export resulting models to IDEs for rigorous modeling such as the Overture tool.

7. Conclusion

We presented the concept of exploratory modeling and proposed a support tool that satisfies four requirements of exploratory modeling. The limited user evaluation carried out was, in general, supportive. The formal specification does not make a good software development by itself, but needs other phases to follow and feedback to the formal specification. A tool chain that covers the entire development is desired. Exploratory modeling is not an exception. Support tools for exploratory modeling with rich and intensive interactivity should fit in a tool chain.

We expect the interactivity explored in this study to be adopted by other formal methods tools such as IDEs and proof assistants.
Acknowledgements

We would like to thank Nick Battle and the anonymous referees for valuable input on this work. This work is partially supported by Grant-in-Aid for Scientific Research (S) 24220001 and Grant-in-Aid for Scientific Research (C) 26330099.

References


Tomohiro Oda works as a senior researcher at Key Technology Laboratory, Software Research Associates, Inc. After receiving his M.Sc. at Kyushu University in 1994, he has been working as a researcher and also an industrial engineer bridging research institutes and industrial developments. His research interests are in Formal Methods, Human-Computer Interaction and Personal Computing Media.

Keijiro Araki received B.S. Degree, M.S. Degree and Ph.D. Degree in Computer Science and Communication Engineering from Kyushu University in 1976, 1978 and 1982 respectively. Currently he is a Distinguished Professor of Kyushu University. He has also been serving as Dean of Graduate School of Information Science and Electrical Engineering, Director of Ito Library, and Director of Research Center for Architecture Oriented Formal Methods, Kyushu University. His research interests include formal approaches to software development, formal modeling and analysis of safe and secure systems.

Peter Gorm Larsen works as a professor at the Department of Engineering at Aarhus University in Denmark. After receiving his M.Sc. at the Technical University of Denmark in 1988, he worked in industry before returning to complete an industrial Ph.D. in 1995. In his industry career he gave industrial support on VDM all over the world. He returned to academia in 2005. His research interests are in Formal Methods, Tool Support, Cyber-Physical Systems and System of Systems.