An IDE for Software Development Using Tabular Expressions [*]

Dennis K. Peters[1], Mark Lawford[2], Baltasar Trancón y Widemann[3]

[1] Electrical and Computer Engineering, Memorial University, St. John's, NL
[2] Dept. of Computing and Software, McMaster University, Hamilton, ON
[3] Software Quality Research Lab, University of Limerick, Limerick, Ireland

Abstract

We present preliminary work on an IDE for formal software development using tabular expressions as the basis for precise specifications and descriptions of software behaviour.

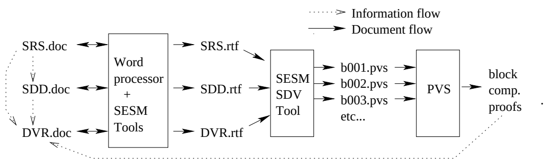
1 Motivation

Formal software specification and design techniques, when combined with analysis and verification tools, can significantly increase the confidence that software will behave as required. Such techniques, however, usually require documents to be produced that include fairly complex mathematical expressions, which can often be effectively presented as tabular expressions. Traditional documentation tools have proven to be insufficient for this purpose. We hope to improve this situation by developing tools that support both the production and maintenance of good documentation and the application of this documentation to such tasks as design analysis, verification and testing.

Although tabular expressions could be useful in many forms of documentation, our emphasis will be on documents that either specify or describe behaviour of software entities using relations on the quantities that are input and output from the component [10]. Rather than define a specification language, *per se*, we use standard mathematics together with some special functions or notations that are particular to the kind of document and are defined using standard mathematics [9].

The experience of Ontario Power Generation (OPG)in developing and using their tools serves as a motivating example. For safety-critical projects, OPG used a software methodology, described in [15], which is based primarily on Parnas' "Rational Design Process" [12] and uses tabular expressions [3]. In this process, in addition to code, the following types of documents are required: system or software requirements [13], module interface [14], module internal design [11], and reports.

As an example of the tool support in the OPG process, the following diagram provides a graphic overview of the relationship between the documents and tools employed in the systematic design verification (SDV) process.



A word processor with additional tools, was used to create and check, the software requirements, software design and design verification report. The exchange medium between the word processor and the SDV tool was Rich Text Format (RTF), which at the time ws the closest thing to a "standard" format available. The shortcomings of this approach included the need for verifiers to use PVS's different, and somewhat esoteric user interface; the manual steps required to integrate the results of the

[*]This work was carried out while all 3 authors were at the Software Quality Research Laboratory, University of Limerick.

verification back into the verification report; and the significant effort required to develop the tools, in part because they had to parse the RTF output and infer the expression semantics from it. We refer the reader to [5, 15] for a further discussion of this process.

2 Preliminaries

2.1 Tabular Expressions

Computer systems often must react to their environment and behave differently under different circumstances. The result is that the mathematics describing system behaviour consists of a large number of conditions and cases. It has been recognized for some time that tables can be used to effectively present such mathematics [2].We view tabular representations of relations and functions as an important factor in making the documentation more readable, and so we have specialized our tools to support them.

A full discussion of tabular expressions is beyond the scope of this paper, so interested readers are referred to the cited publications. In their most basic form, tabular expressions represent conditional expressions. For example, the function definition (1), could be represented by the tabular expression (2).

$$\mathrm{f}(x,y) \overset{\mathrm{df}}{=} \begin{cases} x+y & \text{if } x>1 \wedge y<0 \\ x-y & \text{if } x\le 1 \wedge y<0 \\ x & \text{if } x>1 \wedge y=0 \\ xy & \text{if } x\le 1 \wedge y=0 \\ y & \text{if } x>1 \wedge y>0 \\ x/y & \text{if } x\le 1 \wedge y>0 \end{cases} \qquad (1)$$

$$\mathrm{f}(x,y) \overset{\mathrm{df}}{=} \qquad (2)$$

	$x>1$	$x \le 1$
$y<0$	$x+y$	$x-y$
$y=0$	x	xy
$y>0$	y	x/y

Although (1) and (2) are an academic example, they are representative of the kind of conditional expression that occurs often in documentation of software. The tabular form of the expressions is not only easier to read, but also easier to *write* correctly. Tabular expressions make it very clear what the cases are, and that all cases are considered.

2.2 OMDoc Document Model

The mathematical content markup language OMDoc [4] addresses the problem of communicating the semantics of expressions in documentation and serves as a basis on which to build our tools. A review of the contents of the above document types leads us to propose a document model consisting of the following elements: theories, symbols, types, definitions, code and text. These elements of our documents fall within the OMDoc model. In OMDoc it is straightforward to add support for tabular expressions, simply by defining appropriate (OpenMath) symbols to denote them.[1]

2.3 The Eclipse Framework

Eclipseis an open development platform that supports extension through a plugin mechanism. The platform provides an advanced IDE for software development, and a wide range of available plugins to support such tasks as testing, modeling and documentation. The framework includes a mechanism for plugins to define extension points which facilitate interaction between plugins. Developing our tools in this framework provides several substantial advantages over developing a stand-alone tool set, including its familiarity to the user community, it facilitates tight integration of documents with other software artifacts, the existing plugin base and the built in support for typical software development tasks.

3 Tool Support

The set of tools that may be appropriate outcomes from this project is very large and includes powerful editors, document consistency checkers, verification systems, oracle generators, test case generators and model checkers, to name a few. Developing all of these from scratch would be a major undertaking. However, we are focusing our initial efforts on ways to leverage existing systems to our advantage. The OMDoc representation of a tabular specification with its embedded semantics is the

[1] An example of an OMDoc representation of a specification including tabular expressions is included with the CD or on-line version of this paper.

common glue that allows us to easily bind together components as diverse as an Eclipse plugin GUI, the PVS theorem prover and a prototype function based specification system that also acts as a Java code generator.

3.1 Eclipse Plugin GUI

By developing a prototype plugin to support production of software documents, we are able to build on the strengths of Eclipse to help integrate the documentation into the development process, for example by supporting navigation between a specification and the code that implements the specification or by generating oracles or test cases that integrate with automated testing using the JUnit plugin.

The initial version of this plugin provides a "multi-page editor" (which provides different views of the same source file) for ".tts" files, which are OMDoc files.[2] One page of the editor is a structured view of the document, while another shows the raw XML representation. The support libraries in Eclipse provide techniques to ensure that the views of the document are consistent. The plugin is built using several open source libraries including the RIACA OpenMath Library[3].

This plugin is seen as a primary means for the human users to interact with specification documents. Currently it supports basic verification and validation of tabular specifications via export to the Prototype Verification System (PVS) [8] using XSLT to translate the OMDoc into PVS, as described below.

3.2 Example V&V Environment

PVS is a "proof assistant" that can automatically check for completeness (coverage) and determinism (disjointness) of several types of tables [7], i.e. PVS checks that a table defines a total function. This is very important in safety critical environments since the engineers want to avoid any unspecified behaviour. While PVS has a steep learning curve for users with further development effort we can design the Eclipse plugin to "shield" the users from PVS. Further, new features in PVS such as the random test and execution of a subset of the PVS specification language via the ground evaluator can be easily translated into new table tool features.

PwrCond(Prev:bool, Power, Kin, Kout:posreal):

$$\text{bool} = \begin{array}{|c||c|} \hline Power \leq Kout & FALSE \\ \hline Kout < Power < Kin & Prev \\ \hline Power \geq Kin & TRUE \\ \hline \end{array}$$

Figure 1: Power conditioning specification

We illustrate these capabilities with an example, a "Power Conditioning" subsystem of a reactor Shutdown System (SDS) as illustrated in Figure 1 [5]. The PVS Specification of the *PwrCond* function can be generated from the OMDoc tabular specification by applying a modified version of `omdoc2pvs.xsl` that is available from the OMDoc subversion repository.[4] This PVS specification automatically produces proof obligations for coverage and disjointness. When type-checking the `PwrCond` table the coverage proof obligation is automatically proved by PVS but the Disjointness obligation fails, indicating that the rows might overlap. We can use random testing in PVS to determine if there is an error. This command generates inputs and then evaluates a "theorem" to look for counter examples. Using PVSio we evaluate a counter example on the table header and discover that rows 1 and 3 overlap. The above steps will be automated via the Eclipse plugin using PVS's batch processing mode and then overlapping rows can be highlighted.

3.3 Code Generation

We have constructed the prototype of a tool that provides basic support for function-based specification. It has a front end syntax similar to a functional programming or theorem prover language, and a semantic intermediate representation based on OpenMath objects for individual types and definitions, and OMDoc for theory-level structure. A typechecker supports the Calculus of Constructions [1], a subset of the proposed higher-order type system

[2]See screenshots available with the CD or on-line version of this paper.

[3]`http://www.mathdox.org/projects/openmath/lib/2.0/`

[4]Available at `https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2`.

for OpenMath [6]. Executable code can be generated from the typechecked intermediate representation. The tool is implemented in Java, and currently supports Java code generation.

Specifications represented in function-based style and processed using this tool have two important properties. Firstly, they are defined in a self-contained and unambiguous way in pure typed lambda-calculus. Together with the OMDoc-based format, this makes a good starting point for interaction with various theorem proving tools. Secondly, all properties that do not involve infinite quantification are directly computable. Hence the static restriction check for a table and the evaluation and dynamic restriction check for a table and a given variable assignment can be interpreted or compiled to executable code, whereas the dynamic restriction check for all possible values still requires the use of a theorem prover.

4 Conclusions

Early results show the promise in the chosen techniques – the model supports the needs of our documentation and the ability to interact with other tools such as PVS shows the potential to achieve significant leverage from these tools. Short term goals are to enhance the plugin editor editor, extend translation to PVS and to add oracle generation similar to [13].

References

[1] T. Coquand and G Huet. The calculus of constructions. *Inf. Comput.*, 76(2-3):95–120, 1988.

[2] K.L. Heninger, D.L. Parnas, J.E. Shore, and J. Kallander. Software requirements for the A-7E aircraft. Tech Report MR 3876, Naval Research Laboratory, 1978.

[3] R. Janicki, D.L. Parnas, and J. Zucker. Tabular representations in relational documents. In *Relational Methods in Computer Science*, Ch. 12, pages 184–196. Springer Wien New York, 1997.

[4] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. LNAI 4180. Springer Verlag, 2006.

[5] M. Lawford, P. Froebel, and G. Moum. Application of tabular methods to the specification and verification of a nuclear reactor shutdown system. Accepted for publication in in FMSD Oct 2004.[5]

[6] The OpenMath Society. *A Type System for OpenMath*, 1.0 edition, February 1999.

[7] S. Owre, J. Rushby, and N. Shankar. Integration in PVS: Tables, types, and model checking. In *TACAS '97*, LNCS 1217, pages 366–383, 1997. Springer-Verlag.

[8] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Soft. Eng.*, 21(2):107–125, Feb. 1995.

[9] D.L. Parnas. A family of mathematical methods for professional software documentation. In *IFM'05*, LNCS 3771, pages 1–4. Springer-Verlag, Nov. 2005.

[10] D.L. Parnas and J. Madey. Functional documentation for computer systems. *Science of Computer Programming*, 25(1):41–61, Oct. 1995.

[11] D.L. Parnas, J. Madey, and M. Iglewski. Precise documentation of well-structured programs. *IEEE Trans. Soft. Eng.*, 20(12):948–976, Dec. 1994.

[12] D.L. Parnas and P. Clements. A rational design process: How and why to fake it. *IEEE Trans. Soft. Eng.*, 12(2):251–257, Feb. 1986.

[13] D.K. Peters and D.L. Parnas. Requirements-based monitors for real-time systems. *IEEE Trans. Soft. Eng.*, 28(2):146–158, Feb. 2002.

[14] C. Quinn, S. Vilkomir, D.L. Parnas, and S. Kostic. Specification of software component requirements using the trace function method. In *Int'l Conf. on Software Engineering Advances*, page 50, 2006.

[15] A. Wassyng and M. Lawford. Software tools for safety-critical software development. *STTT*, 8(4–5):337–354, Aug. 2006.

[5] `http://www.cas.mcmaster.ca/~lawford/papers/`