

# TOOL SUPPORT FOR PRODUCTION USE OF FORMAL TECHNIQUES

John C. Knight, Kimberly S. Hanks, and Sean R. Travis

Department of Computer Science, University of Virginia  
151 Engineer's Way  
PO Box 400740  
Charlottesville, VA 22904-4740  
USA

(knight | ksh4q | srt3k)@cs.virginia.edu

+1 804 982 2216 (Voice)

+1 804 982 2214 (FAX)

**Abstract.** The relatively scant use of formal techniques in software development is the result, in part, of a lack of suitable support tools. Many tools have been developed that provide novel analysis capabilities but lack basic yet commonplace facilities which are essential in production software development. More importantly, many existing tools for the development of formal specifications fail to provide mechanisms for the manipulation of natural language despite the fact that natural language is essential to give meaning to the terms in the formal specification. In this paper, we describe a toolset that has been developed with the specific intent of providing comprehensive facilities for creating formal specifications in production software development. The toolset supports a powerful formal notation, Z, but also provides comprehensive and fully integrated support for natural language. As well as describing the toolset we present a preliminary evaluation of its use on a commercial specification.

**Keywords:** Formal specification, natural language, tool support.

# TOOL SUPPORT FOR PRODUCTION USE OF FORMAL TECHNIQUES

## 1 Introduction

The application of formal techniques offers many benefits, and there are certainly examples of the use of formal techniques in production development [2, 9, 11, 15, 21]. Yet, despite their popularity in academia and the claimed benefits, formal techniques are still not as widely used as they might be in commercial software development. Industrial authors have expressed frustration when trying to incorporate formal technologies into practical software development for many reasons including the perceptions that formal techniques increase development time, that they require extensive personnel training, or that they are incompatible with other software packages. Experts in formal methods have analyzed this situation and provided useful insights into the reasons for this low level of acceptance [1, 4, 7, 10, 16, 17].

Although there are several reasons for this low level of acceptance, in this paper we address two important and related reasons in the area of formal specification. The first is the relative lack of effective tool support for the preparation and manipulation of formal specifications [13], and the second is the absence of an integrated view of a specification that includes both natural language text and formal language text. There are plenty of tools for working with formal specifications but, for the most part, they do not provide the level of functionality that is essential in modern industrial development. Of the tools that are available, none of which we are aware views natural language as an integral and essential part of a specification.

We are engaged in a research project to determine what needs to be done to improve this situation, and one of the results of the present research is a comprehensive toolset designed to support the development and use of Z specifications in production software projects. A unique feature that

the toolset introduces is preliminary support for the *combined* analysis of the formal and natural language elements of a specification. In this paper we discuss the requirements for tools used in the development of formal specification, the architecture of the toolset that we have developed to meet those requirements, and the services that the toolset provides.

In section 2 we review the criteria that tools have to meet, and in section 3 we summarize existing tools. The architecture of the toolset that we are developing is discussed in section 4 and the facilities that it provides are described in section 5. In section 6 we present the results of a preliminary evaluation of the toolset, and in section 7 we present our conclusions.

## **2 Issues in Toolset Support**

The disparity between research and production use of formal techniques was examined in earlier research [14]. In that work, a detailed evaluation framework for formal methods was developed based on the requirement that any software technology, including formal methods, has to contribute to one overriding goal—the cost-effective development of high-quality software.

Clearly this means that formal methods must be fully integrated into the software lifecycle. It is unlikely that any formal technique will be adopted for production software development if its use is not fully merged with all the other activities that are required during the lifecycle. Tool support is an essential element of this since tools are the mechanism by which formal artifacts are manipulated. Thus important requirements for tools derive from the need to work with all the other activities of software development, and in an environment in which many software engineers are working together. Meeting these requirements is one of the primary goals of this research. It is with this goal that the research we describe differs from the work of most other researchers.

In this section we outline the various requirements that arise from the need to support the software lifecycle fully. We also introduce the notion of specification analysis in which the formal and informal parts are examined together.

## **2.1 Requirements for the Support of Formal Notations**

Tools to support formal notations must be at least as powerful as those that support informal notations. In addition, they must not impede other existing tools or techniques.

Though true, these are imprecise statements that provide little guidance for tool developers and so we turn to the evaluation framework mentioned above [3, 14]. The criteria in the evaluation framework were derived from the software lifecycle, and the roles of specific formal techniques were studied in each lifecycle phase. The result of this process was an extensive list of criteria deemed necessary for the successful application of formal techniques, and many of the criteria apply to support tools. There are too many criteria to repeat here, but the following are a representative subset of those that apply to support tools for formal specification<sup>1</sup>:

### *General:*

- The toolset must be capable of manipulating commercial specifications written in more than one notation using reasonable resources.
- The toolset must provide state-of-the-art analysis of the formal notations that are used and either support analysis of other necessary notations or interface with other tools used in specification documentation.

### *Usability:*

- The toolset must support routine development requirements such as editing, printing and file manipulation, and have the same “look and feel” as other tools being used.

---

1. Some of the criteria list here have been reworded from their original form for clarification and brevity.

- The toolset must tolerate incomplete specifications and support navigation through the specification.

*Software Lifecycle:*

- The toolset must support specification development by a team, and the manipulation and analysis of specifications in parts (in the sense of separate compilation of source programs).
- The toolset must support the other phases of software development (design, implementation, verification, etc.) by providing necessary information to and interfacing with support tools in these phases.

The complete list of requirements is quite formidable, and many of them are difficult to meet because of the extensive support that is implied. For example, the last one listed is open ended—no specific tools are noted. However, this requirement is essential and quite reasonable since the intent is to ensure that any analysis that might be possible in later phases using information about a specification should not be precluded.

## **2.2 Requirements for the Support of Natural Language**

By definition, any formal notation that is used in software development has no inherent meaning. The names of variables used in any formal specification are merely symbols which can be manipulated according to a rule set; whether a variable is called *x*, *index*, or *capacity* does not affect the formal validity of a piece of notation. However, the real world entities that formal symbols are intended to represent may have real world constraints on their interaction which might render a real world relation among them absurd despite formal validity of its representation. Since the user of a specification will often not possess, for example, some critical piece of domain knowledge on which real world validity of the system depends, it is absolutely necessary that he be provided

with the precise semantics intended to be associated with the formal symbols.

The link between a formal specification and the real world function that the specified system is to perform is forged for a user in a number of ways. It can be done explicitly through annotation, implicitly through the choice of mnemonic symbol names, or in the dangerously uncontrolled fashion of allowing the user to assume what he will based on his idiosyncratic experiences with interpreting specifications. This link between form and meaning is a critical part of the specification. To allow the user to construct that mapping as accurately as possible, considered use of natural language is highly preferred over methods that leave much interpretation to the user. As has been pointed out by other authors[20], natural language is a necessary component of any formal specification. Thus to be able to manipulate a formal specification comprehensively, appropriate tool support for natural language needs to be present.

The preparation and manipulation of documents written *exclusively* in natural language are supported by extensive and powerful tools, many of which are integrated into sophisticated desktop publishing systems. These systems permit natural language documents to be prepared with highly structured and visibly appealing presentation, to be checked for trivial errors, and to be manipulated in a variety of ways including WYSIWYG editing, printing and storage in file systems.

Such tools are also an integral part of the software development process. The ease with which they can be used together with the quality of the documents they produce are expected by industrial developers and so have to be present in any toolset that is proposed for formal specifications.

### **2.3 Requirements for the Support of Combinations of Formal & Natural Languages**

A formal specification that contains material in both formal and natural languages is organized

typically as one document with the two notations interspersed. Integration of the two notations into a single presentation is extremely convenient for the reader because the elements of each are related to each other. For example, a piece of natural language text is present to support a piece of formal notation, and it makes sense for the two pieces to be collocated and interspersed. This presentation format suggests two simple but important requirements:

- Manipulation and analysis tools need to know which language is which so as to be able to undertake appropriate analyses.
- The tools that are provided for the two different languages need to use interfaces that are either the same or closely related so that the system is consistent for the user.

Since at least two notations are integrated in a formal specification, the opportunity for mechanical analysis that is based on the content of *both* is presented. Both notations play an essential role and independent checking of either allow certain properties to be assured. But it is clearly desirable to establish properties of both if possible. Examples of the analysis that might be performed include:

- *Completeness.*  
It is reasonable to expect that certain symbols used in the formal-language part of a specification have definitions in the natural-language part. This is, after all, how meaning is associated with the formal part. A definition of completeness could be defined that required all the symbols in a certain category in the formal-language part to have corresponding definitions in the natural-language part. Similarly, it is reasonable to expect that everything is defined in a prescribed style, and that nothing is defined but never used.
- *Consistency.*  
The incorrect use of a variable in the formal part of a specification can be detected in many

cases by type checking. The same cannot be done in the natural language part. However, the uses of a variable that appears in both parts of a specification can be flagged for visual checks by a human to ensure that the uses are consistent.

This type of analysis has many possible benefits, and it has the potential to ensure a more complete integration of the two notations.

## **2.4 Toolset Development Resources**

In practice, developing production-quality tools to support formal notations, i.e., that meet the requirements summarized above, is unusually challenging. Part of the reason derives from the rapid pace of development in formal methods. Because of this rapid pace, there tend to be numerous innovative concepts for new tools and tool enhancements from both researchers and developers.

A major part of the challenge in developing tools in the formal methods area arises, however, from the requirement for new tools to provide a wide range of customarily available features. To become available for regular use in production engineering development, innovative tool ideas have to be combined with a multitude of routine facilities that, though mundane, are essential. Formal specifications are complex documents, and developers of such specifications appreciate the value of innovative analysis but must be able to perform routine manipulations conveniently.

It is at this point that the greatest difficulties arise. Implementing these routine facilities, facilities that have been referred to as the “superstructure” of a tool [19], requires extensive resources—frequently *many* more than are needed to implement the innovation that is at the heart of the tool. This has led to many tools being developed with only a limited superstructure thereby limiting the exploitation of their innovation.

An obvious manifestation of this problem can be seen in the continued widespread use of the ASCII representation of the Z character set. Using many of the existing tools, engineers developing Z specifications are required to work with the ASCII representation and to edit versions of their specifications that are substantially different from the printed forms. The primary text-manipulation tools at their disposal are an ASCII editor and LaTeX. This is the case at a time when elaborate word-processing systems are in routine use that provide convenient access to operating system services and WYSIWYG editing for natural-language processing.

### **3 Existing Toolsets**

Many toolsets have been developed by both academic and by industrial organizations to support the use of formal languages including Z. For an excellent survey, see the Web site at the University of Oxford[22].

The majority of existing Z toolsets have been developed to explore a specific research direction such as animation or proof support. These toolsets often use the ASCII representation of Z and make use of the facilities available for printing Z using LaTeX. A few (such as Zola [23] and Formalizer [24]) have incorporated graphic editing capabilities and analysis features.

Two toolsets of particular note are CADiZ [25] for Z and IFAD's VDMTools [26] for VDM. Both provide extensive analysis capabilities for formal notations including substantial proof support. Both support the integration of natural and formal languages but neither provides high-quality facilities for the natural language component of the specification. The emphasis in both systems is on formal analysis, and they both rely on LaTeX for manipulation of natural language. Neither supports the kind of powerful facilities found in modern tools for preparing natural-language documents.

## 4 Toolset Architecture

The architecture of the toolset that we have developed is designed primarily to address our major goal of supporting industrial use of formal specification. All of the requirements discussed in section 2 are addressed, and there are four important characteristics that help meet this goal:

- A production desktop publishing system is reused (FrameMaker) and extended substantially to provide many of the required features.
- Convenient access to a high-performance analysis system for Z (Z/EVES) is provided.
- Natural-language text and Z text are processed separately or together as necessary. In the latter case, combined analysis is performed.
- An open central data structure maintains information about a specification so as to ensure that integration with other lifecycle tools is as straightforward as possible.

The high-level architecture of the toolset (named Zeus) together with its connections to both FrameMaker and Z/EVES [18] are shown in Fig. 1. We discuss the four major characteristics of the architecture in turn.

### 4.1 FrameMaker

The implementation of Zeus is based on FrameMaker, a commercial desktop publishing system. Previous research has shown that the requisite superstructure for many tools can be provided by reusing and adapting package programs [19]. By a package program, we mean a very large and powerful application program that implements substantial functionality but which is designed to be adapted for new applications. Package programs include the components of Microsoft's Office suite including the Visio drawing package, and Adobe Systems' FrameMaker. As well as the familiar graphic user interface, all of these packages provide extensive *application-programmer*

*interfaces* (APIs) designed to be used under program control. Almost all of the facilities available in the graphic user interface are also available via the API. By suitable use of the API through special-purpose software, a package program can be tailored to the unique needs of a variety of specialized applications.

Given the difficulties surrounding the development of tools in the formal-methods area and the preliminary success in other domains using package programs, we hypothesized that the use of package programs might help solve the problem outlined above—developing superstructure—for tools in the formal-methods domain.

The FrameMaker API is organized in such a way that functionality is added by creating software that is invoked by FrameMaker when specified events occur. Thus, the actions associated with a Zeus menu item or dialog box is performed by a program that is executed when the action is requested. The major functions that the Zeus software provides are:

- Creation and management of the windows that the user sees and uses.

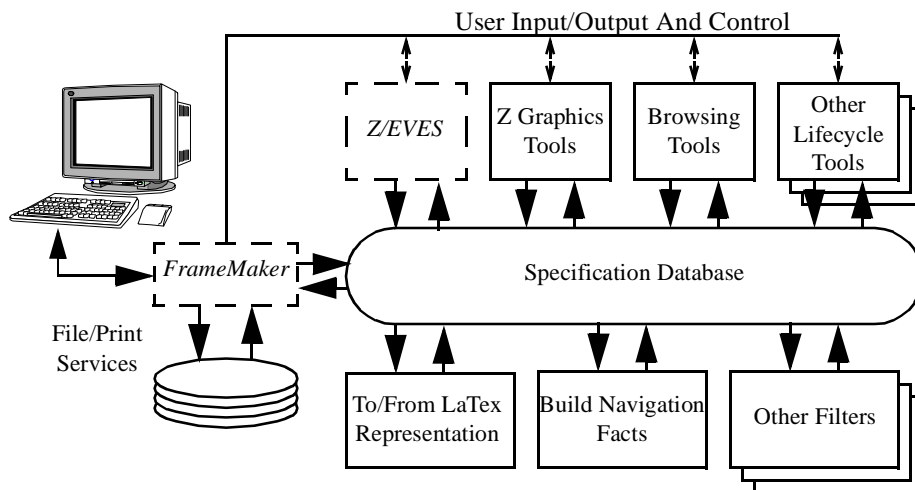


Fig. 1. The Zeus architecture and its interfaces with FrameMaker and Z/EVES.

- Creation of the Z graphic structures and manipulation of the document to cause them to be inserted and displayed.
- Document scanning and parsing to locate and select the formal elements.
- Translation of the internal representation of Z text to the appropriate ASCII form.
- Creation of the Z/EVES process and communication with Z/EVES (see below).
- Parsing of the Z/EVES output to permit translation of the output to the requisite internal form.

## 4.2 Z/EVES

Z/EVES is a widely-used, high-performance system that provides comprehensive facilities for the analysis of Z specifications. It uses state-of-the-art formal techniques to support the analysis of Z specifications in several ways including syntax and type checking, schema expansion, precondition calculation, domain checking, and general theorem proving.

Z/EVES does not, however, meet some of the requirements identified earlier such as convenient specification editing. For example, it operates with a graphic user interface that displays Z text using the Z character set and permits simple editing. But this interface is primarily designed to support the use of the theorem-proving facilities in Z/EVES. Z/EVES can also be used with the ASCII representation of the Z character set in which case it relies upon typical system facilities for editing and file manipulation.

In the context of Zeus, Z/EVES operates as a separate process that communicates with Zeus using sockets. All of the files that Z/EVES needs for analysis of a specification (in XML form) are prepared by filters that are part of Zeus through analysis of the specification and translation of the Z graphics to ASCII characters. The communication is via hidden files so the user is unaware of the translation or the specifics of the Z/EVES interface. The user does have to understand and

appreciate the analysis capabilities of Z/EVES, and when using the advanced theorem-proving facilities in Z/EVES, the user switches to a window providing the Z/EVES interface.

### **4.3 Combined Analysis Of Formal And Informal Notations**

By providing integrated support for natural language, additional opportunities are presented for the combined analysis of relationships between the formal- and natural-language content of a specification. Zeus is designed to take advantage of these opportunities.

As presently implemented, Zeus extracts all of the symbols from the formal part of a specification and stores them with information on their location (which schema, where) and their formal type. It also extracts all the symbols that appear in prescribed locations (particular locations in specified paragraph types) in the English text and stores them along with their location. Checks between the two lists are then carried to: (1) ensure that all the symbols in certain categories extracted from the formal part appear in the English text and in the right form; and (2) to ensure that all the symbols extracted from the English text appear in the formal part and in the right locations.

Planned extensions to this type of analysis include checking on the syntactic structure of the natural language text and checking on the overall structure of the document to ensure appropriate interleaving of formal and informal text.

### **4.4 Central Data Structure**

The entire Zeus system is organized with a central data structure that maintains all essential information about an open specification, and that is saved between sessions working with a particular specification. This provides access to any aspect of the specification that might be needed for interfacing with other tools.

The data structure stores mapping information about the Z text contained within a specification (a set of documents) rather than the Z text itself. Thus, the internal names of paragraphs of Z text and their location within a document (page and line) are stored along with their syntax- and type-checking status. Access to this material is provided through a set of class definitions that can be thought of as an API for Zeus. FrameMaker supplies all the display and user interface services and they are accessed through the FrameMaker API. With these interfaces in place, it is a simple matter to add additional tools to manipulate a combined natural- and formal-language specification.

## **5 Toolset Facilities**

The current version of Zeus runs on the IBM PC/Microsoft Windows platform. It is configured as a tool that can be started from within FrameMaker once FrameMaker is running. Zeus starts Z/EVES as necessary. Z/EVES can be executed on the same machine as Zeus or on a high-performance remote computer if extensive computing associated with proofs is expected.

### **5.1 Manipulation of Natural Language Text**

Since Zeus is based on FrameMaker, it provides all the document-processing features that FrameMaker provides, including:

- WYSIWYG editing using the correct graphics for the Z character set. Editing includes cutting, pasting, and inserting both natural language and Z text, as well as find and replace. The Z character set is available either through a palette or keyboard sequences.
- Access to the host operating system's file system thereby allowing specifications to be stored conveniently. FrameMaker provides a "book" mechanism that allows a document to be stored in separate files that are linked to preserve page numbering and other parameters. Thus a spec-

ification can be developed and manipulated as a book.

- A wealth of formatting features permitting page layouts to be controlled, any compatible font to be used, paragraph formats to be controlled, and so on.
- The standard look and feel of Microsoft Windows applications including all window functions and standard keyboard shortcuts.

In summary, Zeus provides the user with all the capabilities of a powerful desktop publishing system. Clearly if such features had to be developed from scratch for a tool designed to support manipulation of Z specifications, the requisite effort would be tremendous as would the development time.

## **5.2 Manipulation of Formal Text**

Zeus provides many Z-specific facilities that are accessed either by additional menus or by dialog boxes that are not part of the standard FrameMaker interface. These facilities include capabilities for Z graphics control, Z/EVES control, and user preference selection. Fig's 2 and 3 show screens that include a variety of the Zeus interface facilities.

All of the Z graphic structures can be used with Zeus (see Fig's 2 and 3) and are created using a simple menu selection. To insert a schema into a specification, for example, the user merely places the insertion point at the desired location and selects the appropriate menu item. Graphic structures are inserted automatically below the insertion point as anchored frames that move along with the text to which they are anchored. Thus, as text is added to or deleted from a specification, graphic structures remain with their associated natural language text (or with whatever the specifier wishes them to be associated). Graphic structures can, of course, be cut, copied and pasted as desired to move them or copy them as needed.

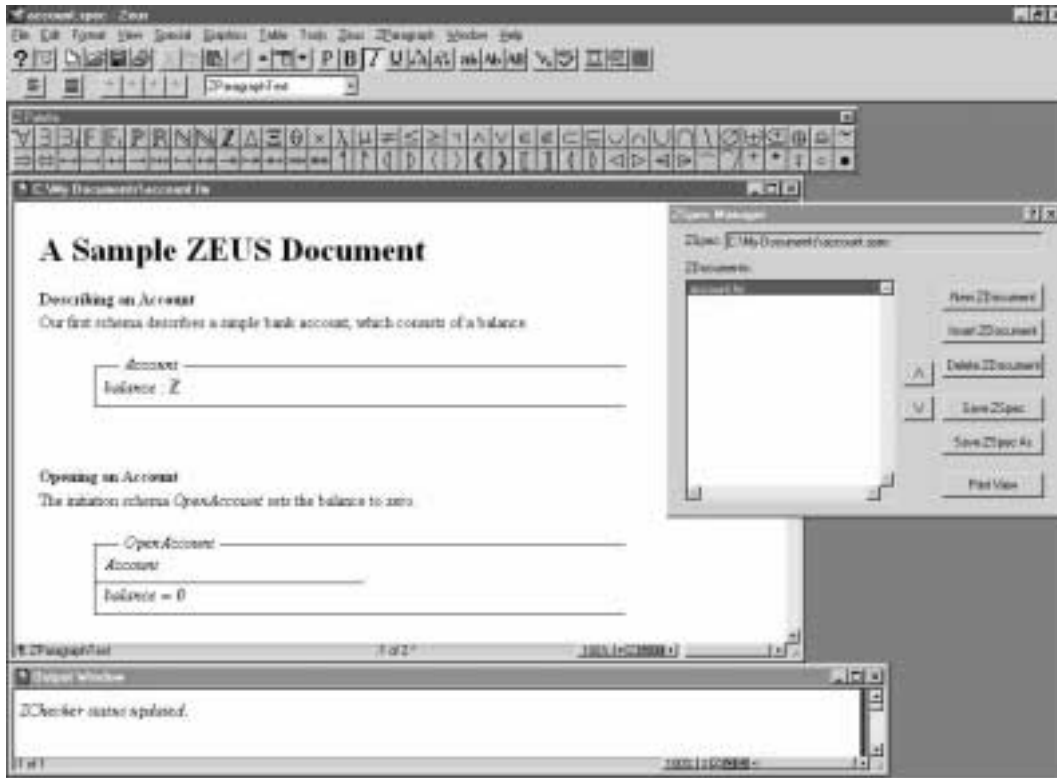


Fig. 2. The character set palette, the Z specification manager and the Z/EVES output window.

The graphics that Zeus synthesizes are inserted with standard spacing and line widths, with default-sized text blocks in the appropriate locations, and with paragraph types and font styles set appropriately. The user can then merely place the insertion point into the text block of, say, a schema predicate to enter Z text. This text can be modified with all the available editing facilities. In Fig. 3 the optional borders that FrameMaker supports are shown. They indicate the extent of the anchored frame and the text blocks that the user can access.

The sizes of graphic structures are adjusted automatically as the user types or deletes text. Graphic structures are not split on page boundaries but always maintained whole by insertion of white space into the document if needed. This is the standard mechanism that FrameMaker uses with anchored frames and is quite convenient for Z specifications.

Frequently the order of material in a Z specification that is syntactically correct differs from the order that a user would like to see in the displayed version of the specification (on screen or on paper). For example, many low-level schemas might be used in a specification and they need to appear before they are used. While reading a specification, it is often more convenient to relegate such schemas to a clear but less prominent location, such as an appendix. To accommodate this, Zeus provides two views of a multi-file specification. The first, the *print* view, provides a view of the specification suitable for the user and presents the files in the order that they are contained in a FrameMaker book. Using this view, Z text can be reordered to suit the convenience of the user and the order specified and changed as needed using the FrameMaker book mechanism.

The second view is the *Z/EVES* view and it is defined using a Zeus dialog box (see Fig.3). This view defines the order in which files will be processed by Z/EVES. Any order can be specified and the order is controlled with a mechanism that is very similar to that used to control file names and order in a FrameMaker book<sup>1</sup>.

### **5.3 Z/EVES Communication And Control**

Zeus has been tailored to provide a convenient interface to Z/EVES. The interface consists of a FrameMaker window and menu items that invoke relevant actions. The status of the checking of individual elements of a specification is maintained and displayed for the user using a color-coded bar that appears to the right of the associated Z entity (dashed bars in Fig. 3). Zeus monitors user actions so as to invalidate the checked status of text that is modified. In addition, Zeus maintains dependency information and invalidates the checked status of Z entities that depend on entities

---

1. We are extremely grateful to Anthony Hall of Praxis Critical Systems for suggesting this two-view strategy for assisting the user in reading Z specifications.

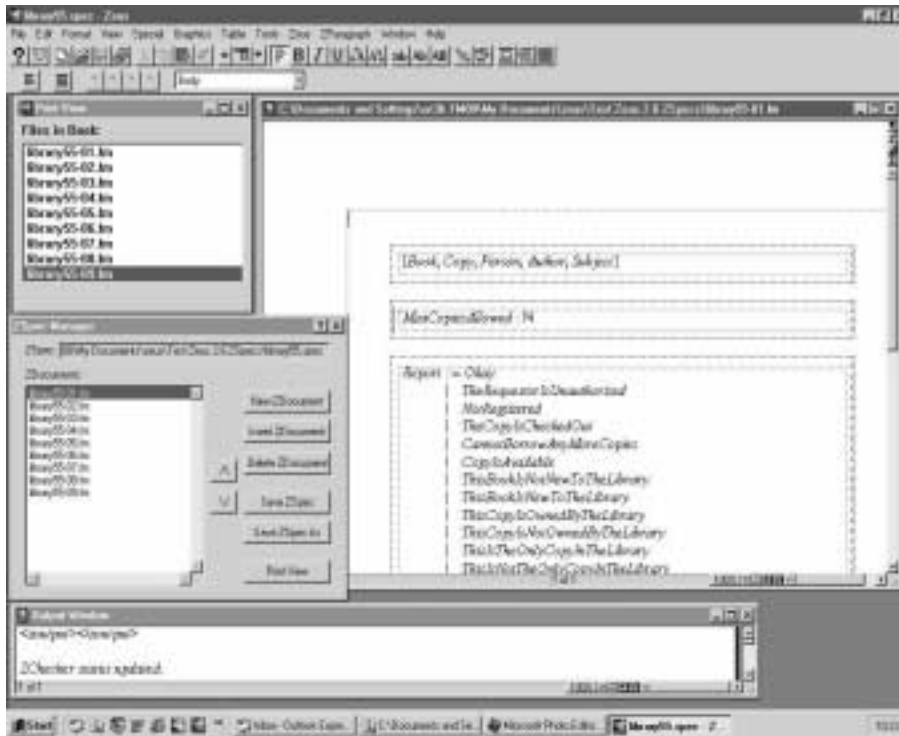


Fig. 3. The two views of a specification along with part of a specification.

that are edited. It also adjusts the state of Z/EVES as necessary if entities are submitted for checking more than once (as is usually the case). The output window displays Z/EVES diagnostic messages and output. The material is displayed as it is received to maintain compatibility with the Z/EVES documentation and to ensure user familiarity.

## 6 Preliminary Evaluation

Zeus has been used on a variety of specifications, mostly small, throughout its development. However, in order to perform a preliminary evaluation of Zeus for the type of industrial project for which it is intended, it has been used to develop a relatively large specification of an international maritime software standard[12]. The primary goal of the development was to evaluate the utility of all the different features of the toolset for editing and analyzing a large formal specification.

The subject of study was a complete standard document, and, as such, it was not a specification of a particular software system but a specification of a standard to which software products have to comply. It contains all of the challenges that are present in typical software specifications.

The standard defines the basic international requirements for automatic track-control systems that are used on ocean-going vessels and a set of tests that are used to determine compliance. The standard was developed by International Electrotechnical Commission (IEC) and the official version is in English. We have developed a formal version that uses Z to specify the formal elements of the standard. The important elements of the standard that were formalized include: (1) the computations associated with distance, course direction, and speed; (2) the data management associated with course planning; and (3) the logic concerned with potentially dangerous situations requiring alarms.

The official standard is written entirely in English, and it is about 83 pages long including appendices. A few pages are devoted to routine context statements but the majority of the English text describes the rules for maritime track control. The formal specification is currently 68 pages long (with an additional 49 pages in test case descriptions) and is stored in several files; the Z text takes approximately half of the space.

The process we followed was to use the overall structure of the original document and to translate the functional statements into Z systematically. Whilst doing so the associated English text was refined, systematized, and properly integrated with the Z text. Syntax and type checking were carried out regularly. Presentation of the formal specification was improved by using FrameMaker's facilities to: (1) add a comprehensive table of contents; (2) create consistent and useful page layouts; and (3) structure the text in a manner that was easy to read using a variety of different fonts, font sizes, and spacings.

Numerous passes were made through the document over a protracted period of time to complete the specification. The development was undertaken by researchers, but periodic reviews were held with industrial experts on maritime electronic systems and large-vessel navigation. Opinions on the formal specification and the process used in its development were solicited from these experts also.

From our experience with this activity we have determined that Zeus either meets or has the potential to meet all of the requirements identified earlier. The vast majority are met because Zeus is integrated with FrameMaker and because it provides a simple and easy-to-use interface to Z/EVES. FrameMaker provides all of the text manipulation facilities and the accessibility of Z/EVES means that all the necessary analysis features are available. We note that, as both products are enhanced over time, Zeus will be able to benefit from these enhancements.

Other requirements are met because Zeus operates on a commonly-used software-development platform. Thus, important software lifecycle capabilities such as version control and configuration management can be provided by existing tools on that platform. In a similar way, the commonly-used software-development platform allows the development of specifications by teams of engineers in exactly the same way as they would develop any other artifact. Integration with lifecycle tools can be achieved as needed by interfacing with the Zeus specification database.

In terms of performance, Zeus operates satisfactorily on ordinary workstations. The most time-consuming activity that the user will conduct with Zeus is the translation of the Z text in a specification into XML for submission to Z/EVES. For our track-control specification, the time to perform syntax and type checking is minimal. Tests of this activity with synthetic specifications that were intentionally built to contain several hundred schemas indicated that the processing resources Zeus uses are small enough to be considered essentially trivial.

Finally, in terms of the development effort required to build Zeus, our experience is entirely positive. Clearly, for the features that Zeus provides or to which it provides access, the amount of source code that had to be written is remarkably small. The source code for Zeus is only a few thousand lines. It is hard to estimate the amount of software that we have *not* had to write thanks to the facilities that we get from FrameMaker and Z/EVES but it is certainly hundreds of thousands of source lines<sup>1</sup>.

## 7 Conclusions

Tool support of formal techniques is essential if these techniques are to achieve wide acceptance in industrial software development. Successful tools, however, must address the fundamental issue of integration into the software lifecycle. Formal specifications are documents that are used in every lifecycle phase and so must be convenient to develop and use. The support provided by other tools, though powerful in many respects, does not address this critical requirement in a comprehensive way. By making available all the facilities of FrameMaker and Z/EVES as well as adding to their capabilities, Zeus does address this critical requirement

Zeus is a prototype but by prototype we mean merely that it lacks production features like detailed manuals, a “help” system, user support, and so on. Zeus is quite capable of supporting the development of large specifications, and we are using it for this purpose. By introducing the concept of combined analysis, Zeus permits new properties to be established for formal specifications that increase the users’ confidence that the formal and informal parts are consistent.

---

1. For more information about Zeus, see <http://www.cs.virginia.edu/zeus>

## Acknowledgments

It is a pleasure to thank Dan Craigen and his colleagues at ORA Canada for their extensive technical support in this project. We also thank John Yancey and Jonathan Michel of Litton Marine Systems for providing extensive technical help on marine navigation and systems, Anthony Hall of Praxis Critical Systems for his comments about Zeus, and Tom Fletcher, Brian Hicks, Zac Kohn, Steve Ziegler, and Roy Bodayla for implementation support on earlier versions of Zeus. This work was supported in part by the National Science Foundation under grant number CCR-9213427, and in part by NASA under grant number NAG1-1123-FDP.

## References

1. Mark A. Ardis, John A. Chaves, Lalita J. Jagadeesan, Peter Mataga, Carlos Puchol, Mark G. Staskauskas, and James Von Olnhausen. A Framework for Evaluating Specification Methods for Reactive Systems: Experience Report. *IEEE Transactions on Software Engineering*, 22(6):378–389, June 1996.
2. Dan Craigen, Susan Gerhart, Ted Ralston. *An International Survey of Industrial Applications of Formal Methods*. U.S. Department of Commerce, March 1993.
3. Colleen L. DeJong, Matthew S. Gible, John C. Knight, and Luís G. Nakano. Formal Specification: A Systematic Evaluation. Technical Report CS-97-09, Department of Computer Science, University of Virginia, Charlottesville, VA, June 1997.
4. David Dill and John Rushby. Acceptance of Formal Methods: Lessons from Hardware Design. *IEEE Computer*, 29(4):23-24, April 1996.
5. Antoni Diller: *Z: An Introduction to Formal Methods*, (2nd ed) John Wiley and Sons, Inc., New York (1994).
6. Stuart Faulk. Software Requirements: A Tutorial. *Technical Report NRL/MR/5546—95-7775*, Naval Research Laboratories, November 14, 1995.
7. Anthony Hall. What is the Formal Methods Debate About? *IEEE Computer*, 29(4):22-23, April 1996.
8. Constance Heitmeyer and John McLean. Abstract Requirements Specification: A New Approach and Its Application. *IEEE Transactions on Software Engineering*, 9(5), Sept. 1983.
9. Kathryn L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering*, 6(1):2–13, January, 1980.

10. C. Michael Holloway and Ricky W. Butler. Impediments to Industrial Use of Formal Methods. *IEEE Computer*, 29(4):25-26, April 1996.
11. I. Houston and S. King. CICS Project Report: Experiences and Results from The Use Of Z In IBM. *VDM '91. Formal Software Development Methods, Vol. 1: Conference Contribution*. Lecture Notes in Computer Science, Volume 552, Springer Verlag, 588–596.
12. International Electrotechnical Commission, Maritime navigation and radiocommunication equipment and systems - Track control systems - Operational and performance requirements, methods of testing and required test results. Project number: 62065/Ed. 1 (2000-08-11)
13. John C. Knight: Challenges in the Utilization of Formal Methods. *Proceedings of the workshop on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Technical University of Denmark, Lyngby, Denmark, September 1998.
14. John C. Knight, Colleen DeJong, Matthew Gible, and Luis Nakano: Why Are Formal Methods Not Used More Widely? NASA Workshop on Formal Methods, Hampton VA (1997).
15. Robyn Lutz and Yoko Ampo. Experience Report: Using Formal Methods For Requirements Analysis Of Critical Spacecraft Software. In *Proceedings of the 19th Annual Software Engineering Workshop*, pp. 231–248, Greenbelt, MD, December 1994. NASA Goddard Space Flight Center.
16. C. R. Nobe and W. E. Warner. Lessons Learned from a Trial Application of Requirements Modeling using Statecharts. In *Proceedings the Second International Conference on Requirements Engineering*, pp. 86–93, April 15–18, 1996.
17. John Rushby. Formal Methods and the Certification of Critical Systems. *Technical Report CSL-93-7, SRI International*, December 1993
18. Mark Saaltink and Irwin Meisels: The Z/EVES Reference Manual. ORA Canada, TR-97-5493-03 (1997)
19. Kevin Sullivan and John C. Knight: Experience Assessing an Architectural Approach to Large-scale Systematic Reuse, *Proceedings of the 18th International Conference on Software Engineering*, Berlin Germany, March 1996.
20. Pamela Zave and Michael Jackson, *ACM Transactions on Software Engineering and Methodology*, Volume 6, Issue 1, pp. 1-30, (January 1997).
21. Some industrial uses of iLogix tools can be found on-line at: <http://www.ilogix.com/company/success.htm>, 1997.
22. Oxford Computing Laboratory, Z archive, <http://www.comlab.ox.ac.uk/archive/z.html#tools>
23. Zola, Imperial Software Technology, <http://www.ist.co.uk/products/zola.html>
24. Formalizer, Logica Corp., <http://public.logica.com/~formaliser/formlsr/formlsr.htm>
25. CADiZ, York Software Engineering, <http://www-users.cs.york.ac.uk/~ian/cadiz/home.html>
26. VDMTools, IFAD, <http://www.ifad.dk/Products/VDMTools>