SOFTWARE VERIFICATION RESEARCH CENTRE

DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF QUEENSLAND

Queensland 4072
Australia

TECHNICAL REPORT

No. 96-01

Specification Models

Ian J. Hayes

January 1996

Phone: +61 7 3365 1003
Fax: +61 7 3365 1533

Specification Models*

Ian J. Hayes

Abstract

There is a wide range of specification languages, but one can group specification languages according to the class of problems they can specify. The aim of this paper is to examine a hierarchy of specification models, where each model corresponds to a group of specification languages that cover a similar area.

1 Introduction

The practical reason for making use of a specification is that it allows a separation of concerns between the use of a component and the implementation of the component. In writing a specification we are trying to capture the essence of what a component does, without the clutter of detail concerned with how it does it.

As a specification is primarily used to understand the function of a component, it should be as clear and as concise as possible. A specification should be easier to understand than a corresponding implementation, while containing all the relevant information necessary to be able to use the component. The challenge to writing a good specification is to achieve both the goals of comprehensibility and completeness. An informal natural language specification is more likely to run into problems with a lack of completeness, including ambiguities. Use of a formal specification helps avoid these problems, but the challenge is to write a comprehensible formal specification. Achieving this goal is aided by choosing appropriate abstractions to specify the component.

A formal specification represents an abstract model of the behaviour of the system. In devising a specification of a system, one can identify a number of levels at which one must make appropriate decisions in order to produce a suitable specification:

- one needs to choose a specification model (or language) that has the ability to specify the behaviour of the desired system;
- using that specification model, one needs to choose appropriate abstractions that capture the particular system simply and accurately; and
- one needs to use appropriate notation to describe the abstractions.

The remainder of the paper examines alternative specification models in more detail. In the process we touch on the need for suitable abstractions and notations to describe particular systems. We can identify a hierarchy of specification models that are progressively more sophisticated in their ability to model systems. The levels can describe, respectively, procedures, data, concurrency, and distributed systems. Each of these levels can be further extended to cover timing and reliability aspects. We proceed by highlighting the main features of each of these specification models.

*This report appeared as an invited paper in *Proc. 7th International Conference on Putting into Practice Methods and Tools for Information Systems Design, 10–12 October, 1995, Nantes, p.1–10.*

2 Procedure

The simplest specification model treats a component as a black box which takes an input and produces an output. Such a model is suitable for specifying individual procedures and batch programs. Already at this level a number of interesting points about specifications can be discussed.

Assumptions and effects A common method of specifying procedures is by stating the assumptions a procedure can make about its input (a precondition) and the effect it is required to achieve on termination (a postcondition) [4, 11]. In VDM [13], a specification is of this form. A procedure is required to terminate whenever its precondition holds initially. A postcondition is a predicate on both the input and output variables that is required to hold on completion of the procedure provided that the precondition held on the input variables.

The refinement calculus [2, 3, 19, 21] decouples the use of pre/post-condition pairs from procedures by introducing a specification statement [20] that allows a specification to appear in a program anywhere a normal statement can occur. Of course programs containing specification statements are not executable code.

In Z [22], a specification consists of a single predicate on both the input and output variables which encodes both a precondition and a postcondition. The precondition can be extracted from such a specification.

Frames When specifying a procedure, it is important to distinguish between variables that the procedure may modify (they are in its frame) and variables it may only read. In VDM, variables that may be modified by a procedure are indicated by the annotation `ext wr` whereas read-only variables are annotated by `ext rd`. In the refinement calculus, a specification statement has a component that indicates the set of variables that may be modified.

Z takes a different approach: if the value of a variable (x') after a procedure call is to be the same as it was before the call (x), then an explicit predicate equating x' and x has to be included in the specification of the operation. As such constraints are common, Z has an abbreviation to ease writing them.

One subtle point worth mentioning here is that in the presence of concurrency, the VDM and Z approaches differ significantly. In VDM a variable not annotated as `ext wr` cannot be modified, but in Z there is no requirement that a variable constrained by a predicate of the form $x' = x$ not be modified during the execution of the procedure, provided its value on termination of the procedure is the same as its value initially. With concurrent access to shared variables, such differences become significant.

Nondeterminism A specification of a procedure may be nondeterministic: for a given input it does not *uniquely* determine the value of the output. For example, consider the specification of a sorting procedure [10] where a sequence of records is to be sorted according to some key field. If two or more records have the same key value, we do not care in which order they appear in the output.

Sort

$in, out : \mathrm{seq}\ Record$

$$(\forall\, i, j : \mathrm{dom}(out) \bullet i \leq j \Rightarrow$$
$$\quad out(i).key \leq out(j).key) \land$$
$$items(in) = items(out)$$

The above specification requires that the output sequence is in nondecreasing order according to key values, and that the bag (multiset) of items in the output is the same as the bag of items in the input. This specification has no precondition other than that the input is a sequence of records. It is nondeterministic because, if there are multiple records with the same key, it does not determine in what order those records should appear. Note that the use of an appropriate mathematical abstraction, in this case bags [5], helps to simplify the specification.

Executability The goal of a specification is to provide a clear and precise description of a component. With this goal in mind, there is no requirement that a specification language be executable. Hence when designing a specification language, one can concentrate on making the language as simple and expressive as possible without concern for implementing the language.

Logic and set theory have been used in the specification languages of VDM [13] and Z [22]. These languages allow for simple abstract specifications of components, but are powerful enough to allow the specification of problems which are not computable, such as the halting problem.

The lack of a requirement for executability of specification languages is dealt with in more detail in [8], which also has a more detailed discussion of nondeterministic specifications.

3 Data

The next level of sophistication introduces local data or state as a factor in the specification model. The data model is suitable for modules with local (hidden) state and for programs that interact sequentially with their environment. For example, modules in Modula-2 have local variables (state) and a set of operations (procedures) which act on that state.

To specify systems with local data, one needs a richer specification model than that used to specify procedures, because the results of a call to an operation within a module depend not only on the inputs to the call, but also on the value of the local data. The latter is determined by the complete sequence of previous calls on all the operations of the module. Instead of modelling each procedure in complete isolation, one needs to consider the set of operations together. Such systems can be specified either by giving an abstract state and a set of operations on that state (so called model-based specification), or by giving a set of operations and a set of equations relating the interactions of the operations (algebraic specification). With the model-based approach, each operation is specified in a form similar to that used for procedures, except that the local data acts as both an additional input and output. The allowed initial values of the local data also need to be specified.

The most significant difference from specifying procedures is that the local data is not directly accessible from outside the module. This allows the abstract local data used in the specification to be replaced by a more efficient representation of that data, provided the external behaviour of the module corresponds to that of the specification. The technique of data refinement is widely used for this purpose.

Because the local data used in the specification may be replaced by a concrete representation that allows for a more efficient implementation of the module, the specification can use an abstract description of the local data that allows a clear and concise specification of the operations. There is no need to concern oneself with implementation considerations: the goal is an understandable specification. To achieve this goal, one may use abstractions that may not be available in the chosen implementation language. For example, in specifying a bill-of-materials system [7], a suitable abstraction for describing the parts explosion relationship (including frequencies) is a two-dimensional, natural-number matrix. The use of matrices allows the operations on the bill-of-materials system to be specified clearly and consisely using matrix operations, although more efficient techniques are likely to be used in the implementation of such a system.

The most common form of data abstraction allows only a single instance of the data. It is called an abstract machine and is the form used with AMN [1], VDM [13] and Z [22]. More sophisticated schemes allow multiple instances of data, so that a module acts like a data type, or exports a data type.

4 Concurrency

The data model can describe a single sequential stream of interaction, but cannot adequately describe systems involving concurrency, such as a system with multiple users interacting with it. Again a richer specification model is required to adequately capture the behaviour of such systems. Such systems can be specified using process algebras such as CSP [12] and CCS [18], or traces of events.

To model such systems, we need to introduce the concept of a process and communication channels of some form between processes (within these we include shared memory with some form of synchronisation primitives). The model needs to be able to define the evolution of a single process over time in relation to the interactions it has with other processes, including input/output interactions with the environment of the system.

As with the earlier models, the assumptions a process can make about its environment can be separated from the effect it is to achieve, and its frame (alphabet of communication channels) needs to be specified. Again specifications can be nondeterministic, and there is no requirement at the specification level that it be directly executable.

5 Distributed systems

Our final level of specification model includes the physical distribution of components of the system. This aspect is necessary for distributed applications and the specification of components such as communication protocols. A distributed system may be as simple as a local area network where communication is cheap, or it may involve more widely distributed components on a global scale.

The typical approach to modelling distributed systems assumes that the system consists of concurrent processes and that each process has a location (either physical or logical). In addition, there are (asynchronous) communication channels between the processes.

6 Timing

To a large extent timing is an orthogonal concern to those listed above. It can be added to each of the above specification models:

- the time available for a procedure call may be limited according to the size of the input to the procedure;
- for the data model one can put time limitations on not only the individual operation calls but also sequences of calls;
- for concurrency one can extend the time limits of the data models to include time limits on interactions between multiple processes;
- for distributed systems the physical location of components of the system can be

taken into account when specifying timing constraints, for example, the timing constraints may allow for the 'distances' between components.

The time may be either real-time, or some abstraction of time (logical time) suitable for describing the properties of the system. Real-time may be measured in either discrete units such as clock cycles or using real numbers.

7 Continuous systems

To give a top-level specification of a real-time control application, one needs to be able to specify the behaviour of the whole system including components which may be implemented as a combination of software and hardware, digital and analog.

Classical control systems are typically modelled using continous, real-valued functions of (real-valued) time. This should be contrasted with software development and digital hardware design which typically make use of discrete-valued variables and which are commonly modelled as state machines with discrete state-to-state transitions. The first step in providing an integrated framework is to devise a common model for control systems and state machines.

One approach is to model all state variables, whether they are implemented in software or hardware (digital or analog), as functions of (real-valued) time [6, 16]. For analog variables this is the natural model commonly used in science and engineering, but for discrete variables this represents a departure from the norm. For discrete variables, a function of time corresponds to a trace of the history over time of the value of the variable. There will be contiguous intervals of time during which the variable has a fixed value, interleaved with intervals representing transition times, when the value of the variable is not defined. To allow for the undefined periods, we make use of a partial function of time: the domain of the function corresponds to those intervals of time during which it is defined.

As a further aid to understanding (inherited from science and engineering), we tag variables with their dimensions of measurement [9]. The inclusion of such information as the units of measurement not only provides a more complete specification but it also provides a further validity check on specifications annotated with dimensions of measurement.

For a *process*, we specify both the *assumptions* that it makes about its environment, and the *effect* that it achieves. Provided the environment satisfies the assumptions, the process is required to achieve the desired effect. Both the assumptions and effect can be specified via predicates. The assumption predicate selects those traces of the environmental variables for which the process is required to operate correctly, and the effect predicate defines the allowable behaviours of the process [15, 14]. There is an analogy here with the use of preconditions (assumptions) and postconditions (effects) to specify components of sequential programs. However, the assumptions and effects of timed refinement are quite different to preconditions and postconditions: the former are predicates over all time, while the latter are predicates on the state of the system at two different instants of time.

The assumptions may either correspond to assumptions about the physics of the environment in which the system is to be placed — in essence these are assumptions about the laws of nature — or they may correspond to conditions under which the component may be safely used. These latter assumptions must either be guaranteed by the user of the system or, if the process in question is a component of a larger system, they need to be guaranteed by the operation of the remainder of the system.

Typically there is a range of behaviour of a control system that satisfies the requirements of the user of the system. To allow for this, the effect predicate specifies a set of allowable behaviours for a given environment — the specification is nondeterministic. This allows a degree of latitude in the implementation: provided it always achieves one of the allowable behaviours, it is a valid implementation.

The primary method of implementation is to decompose a process into a set of parallel processes that work in concert to achieve the desired behaviour of the system [17]. The component processes in such a decomposition can represent either software or hardware, or a combination of both that will require further decomposition.

8 Reliability

Reliability can also be considered orthogonal to the various specification models. If we have hardware components with a known probability of failure (or more generally failure modes), we would like to deduce the probability of a system (software plus hardware) failing.

For some applications the reliability of a component varies over time. For example, as a component approaches its typical lifetime it becomes less dependable. For such systems the probability of failure of a component varies with time. The reliability of a component may also depend on the amount of use the component gets over its lifetime, so that its probability of failure is not just a function of time, but a function of the complete history over time of the use of the component.

9 Conclusions

Different specification models are appropriate for different applications. We have presented an hierarchy of specification models: procedures, data, concurrency and distributed systems. Both timing and reliability are orthogonal concerns which can be added, either independently or together, to the basic hierarchy. This gives three new hierarchies that have the obvious relationships with the original hierarchy and each other.

There are some facets of specification that occur in all the specification models we have considered:

- use of the assumption/effect paradigm for specifying a component within an environment;
- use of a frame (or alphabet) limiting the variables (channels) that a component can modify;
- the use of nondeterminism in specification to allow a wider range of implementations; and
- executability of the specification language is not a requirement.

References

[1] J. R. Abrial. A formal approach to large software construction. In J. L. A. van de Snepscheut, editor, *Proceedings of Mathematics of Program Construction, The Netherlands*, volume 375 of *Lecture Notes in Computer Science*, pages 1–20. Springer-Verlag, June 1989.

[2] R.-J. Back. Correctness preserving program refinements: Proof theory and applications. Tract 131, Mathematisch Centrum, Amsterdam, 1980.

[3] R.-J. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[4] R. W. Floyd. Assigning meaning to programs. *Math. Aspects of Comput. Sci.*, 19:19–32, 1967.

[5] I. J. Hayes. A generalisation of bags in Z. In J. E. Nicholls, editor, *Z User Workshop: Proceedings of the Fourth Annual Z User Meeting, Oxford, December 1989*, Workshops in Computing, pages 113–127. Springer-Verlag, 1990.

[6] I. J. Hayes. Specifying physical limitations: A case study of an oscilloscope. Technical report 167, Department of Computer Science, University of Queensland, July 1990.

[7] I. J. Hayes. Multi-relations in Z: A cross between multi-sets and binary relations. *Acta Informatica*, 29(1):33–62, February 1992.

[8] I. J. Hayes and C. B. Jones. Specifications are not (necessarily) executable. *IEE/BCS Software Engineering Journal*, 4(6):330–338, November 1989.

[9] I. J. Hayes and B. P. Mahony. Using units of measurement in formal specifications. *Formal Aspects of Computing*, 7(3):329–347, 1995. Also available as Technical Report UQ-SVRC-94-31, Software Verification Research Centre, University of Queensland.

[10] Ian Hayes, editor. *Specification Case Studies.* Prentice Hall International, 1987. Second edition 1993.

[11] C.A.R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580, 583, October 1969.

[12] C.A.R Hoare. *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[13] C. B. Jones. *Systematic Software Development Using VDM.* Prentice Hall International, second edition, 1990.

[14] B. P. Mahony. *The Specification and Refinement of Timed Processes.* PhD thesis, Department of Computer Science, The University of Queensland, 1991.

[15] B. P. Mahony and I. J. Hayes. A case study in timed refinement: A central heater. In *Proc. BCS/FACS Fourth Refinement Workshop*, Workshops in Computing, pages 138–149. Springer-Verlag, January 1991.

[16] B. P. Mahony and I. J. Hayes. Using continuous real functions to model timed histories. In P. A. Bailes, editor, *Proc. of the 6th Australian Software Engineering Conference (ASWEC91)*, pages 257–270. Australian Computer Society, July 1991.

[17] B. P. Mahony and I. J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Transactions on Software Engineering*, 18(9):817–826, September 1992.

[18] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[19] Carroll Morgan. *Programming from Specifications.* Prentice Hall, second edition, 1994.

[20] C.C. Morgan. The specification statement. *ACM Trans. Prog. Lang. and Sys.*, 10(3), July 1988.

[21] J.M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, December 1987.

[22] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International, second edition, 1992.