Specifying a Safety-Critical Control System in Z

Jonathan Jacky*
Department of Radiation Oncology RC-08
University of Washington,
Seattle WA 98195, USA

Prepared for submission to:
FME '93 Symposium: Industrial Strength Formal Methods
Odense, Denmark, 19 to 23 April, 1993

Abstract

This report presents a formal specification in the Z notation for a safety-critical control system. It describes a particular medical device but is quite generic and should be widely applicable. The specification emphasizes safety interlocking and other discontinous features that are not considered in classical control theory. A method for calculating interlock conditions for particular operations from system safety assertions is proposed; it is similar to ordinary Z precondition calculation, but usually results in stronger preconditions. The specification is presented as a partially complete framework that can be edited and filled in with the specific features of a particular control system. Our system is large but the specification is concise. It is built up from components, subsystems, conditions and modes that are developed separately, but also accounts for behaviors that emerge at the system level. The specification illustrates several useful idioms of the Z notation, and demonstrates that an object-oriented specification style can be expressed in ordinary Z.

*email `jon@radonc.washington.edu`, telephone 206.548.4117

1 Introduction

Safety-critical control systems are often advocated as ideal applications for formal software development methods [1]. However, there are very few published examples of formal specifications for real safety-critical systems that have been built and used. Those few are expressed in notations that are not in wide use (for example, [5]).

The complexity of a real control system confronts the specification writer with problems of style and organization whose solutions are not apparent from most small case studies found in the literature. Examples of formal specifications for realistic control systems might serve as models, or *reusable frameworks* [3], that could be adapted to other projects.

The large literature on control theory (for example, [2]) emphasizes continuous, closed-loop controls. It provides little guidance regarding discontinuous, essentially "open-loop" operations such as turning subsystems on and off, and safety interlocking. Such features dominate the requirements for many safety-critical systems, including our own.

Researchers concerned with safety issues have proposed abstract formal models of process control systems that provide criteria for evaluating specifications for desirable properties such as completeness and safety [10, 13]. This work challenges builders of real systems to provide specifications that are sufficiently formal to support such evaluation.

This report describes a framework for formal specifications of safety-critical control systems, and demonstrates its application to a real medical device. Some of our preliminary work was reported in [6].

2 A case study

The Clinical Neutron Therapy System at the University of Washington is a cyclotron and treatment facility that provides particle beams for cancer treatments with fast neutrons, production of medical isotopes, and physics experiments. The facility was installed in 1984, and includes a computer control system provided by the cyclotron vendor [15]. Devices under computer control include a 900 amp electromagnet and a 30 ton rotating gantry, as well as four terminals at three operator consoles. The control system handles over one thousand input and output signals, and includes six programmable processors as well as some nonprogrammable (hard-wired) controls.

The University is now developing a new, successor control system. This development project is motivated by requirements to make the system easier and quicker to use, easier to maintain, and able to accomodate future hardware and software modifications.

We have mostly completed an informal specification, which is being produced with the participation of the therapists, physicists and engineers who use and maintain the facility. It will comprise about 500 pages of prose and diagrams [8, 9], and documents the requirements expressed in the formal specification. We hope that the formal specification will be much shorter and will serve as the primary guidance for software development.

3 A framework for safety-critical systems

A *framework* is formal model that abstracts the central features of a family of applications, which can be adapted or extended to fit the needs of particular projects [3]. Our specification is presented here as a partially complete framework that can be edited and filled in with the specific features of different control systems.

3.1 State variables, control laws and safety assertions

Reviewing our prose specifications [8, 9], we find that most of our requirements can be expressed by a quite simple framework: a system is a collection of *state variables* that must obey certain *control laws* and *safety assertions.* This can be modeled by a Z state schema.

The state variables are named in the schema declaration and can be discrete indicators or numeric quantities. The control laws and safety assertions are system invariants which appear as schema predicates. Control laws are formulae that relate state variables in a way that produces the intended system behaviors. In classical control theory [2], control laws are usually differential equations that relate continuous variables, but our control laws also include discrete variables and logical connectives. Safety assertions are formulae that place additional constraints on the state variables, as required by considerations of human safety and equipment protection.

As an example of this framework, here are some definitions and a (much simplified) state schema for our cyclotron. The schema shows a few of the state variables and laws concerned with the radio-frequency (RF) amplifiers that accelerate the particles, the magnet that confines them, and the shielding door that protects staff and visitors from scattered radiation.

$$STATUS ::= disabled \mid off \mid on \mid error$$

$$CURRENT == -100.00 \,..\, 900.00$$

Many more definitions ...

Cyclotron

$mainfld : STATUS$
$mainfld\_setpoint, mainfld\_preset, mainfld\_current : CURRENT$
$rf : STATUS$
$door : DOOR$

Many other state variables ...

$mainfld \in \{disabled, off\} \Rightarrow mainfld\_setpoint = 0.00$
$mainfld \in \{on, error\} \Rightarrow mainfld\_setpoint = mainfld\_preset$
$mainfld = on \Rightarrow \mid mainfld\_setpoint - mainfld\_current \mid \leq \epsilon$

Many other control laws ...

$rf = on \Rightarrow door = closed$
$rf = on \Rightarrow mainfld = on$

Many other safety assertions ...

The particle beam is considered to be on whenever the RF drive amplifiers are on. When the main magnet field is off, its current is zero; when it is on, its current is held at a nominal preset value (this magnet also has a disabled state from which it cannot be turned on, and an error state where it has been turned on but is not running correctly). The safety assertions say that the beam can only be on when the vault door is closed and the main field is running within its nominal range.

This report does not describe how the state variables are input, output, or transformed between their values in meaningful engineering units and their low-level representation as bit patterns in device registers. Those vital activities are the subject of another report [7].

3.2 Operations

The control system provides a repertoire of *operations* that can change the values of some state variables. These are modeled by Z operation schemas.

For example, this operation turns on the main field power supply, unless it has been disabled.

$$
\begin{array}{|l}
\hline
\textit{TurnOnMainfld} \\
\Delta \textit{Cyclotron} \\
\hline
\textit{mainfld} \neq \textit{disabled} \\
\textit{mainfld\_setpoint}' = \textit{mainfld\_preset} \\
\hline
\end{array}
$$

Changing $\textit{mainfld\_setpoint}$ usually causes $\textit{mainfld\_current}$ to follow (the control law for this is rather complicated and is not shown). The control laws require that *mainfld* must change as well; it either becomes *on* or *error*, depending on whether $\textit{mainfld\_current}$ approaches $\textit{mainfld\_setpoint}$.

This illustrates a common technique for writing concise operation definitions: the variables explicitly changed in the operation schema drive other variables, as dictated by the control laws. Therefore, operation definitions usually do not include predicates that fix the values of variables that are not explicitly changed.

3.3 Interlocks

A distinguishing feature of safety-critical control systems is that many operations are *interlocked*; they are not allowed to proceed if certain potentially hazardous conditions exist. In our framework, interlocks are preconditions for operation schemas. If a precondition is not satisfied, the interlock is *set* or *active*, and the operation must not proceed; otherwise the interlock is *clear*.

Consider the operation invoked by pressing the BEAM ON button. Here is a naive specification.

$$
\begin{array}{|l}
\hline
\textit{TurnOnBeam} \\
\Delta \textit{Cyclotron} \\
\hline
\textit{rf} \neq \textit{disabled} \\
\textit{rf}' \in \{\textit{on}, \textit{error}\} \\
\hline
\end{array}
$$

This schema says that pressing the BEAM ON button when the RF system has not been disabled will attempt turn on the RF drive amplifiers (it cannot be guaranteed that they will turn on; they may indicate an error).

This definition is not consistent with the intent of the system safety assertions. Additional interlocks should prevent the beam from turning on if the vault door is not closed, or the main field current is outside its nominal range.

It seems that it should be possible to calculate the interlock conditions from the system safety assertions. However, the ordinary Z precondition [16, 14] is too weak; for example, it does not include $door = closed$. We cannot calculate any ordinary Z precondition involving *door* because we cannot require that the value of *door* remain the same in the "before" and "after" states; *door* is an input sensor whose value may change at any time.

To achieve the intended effect, the interlock predicates should be chosen to ensure that the "after" state of the operation schema (the state formed by the primed schema variables) will be sure to satisfy the system state invariant *even when the values of all the sensor variables remain the same in the "before" and "after" states.*

We can state this formally, by making a stronger version of the usual Z precondition expression: from the state schema S, extract the schema *Sensor* that consists only of the declarations of the state variables that represent sensors whose values cannot be directly controlled. This is necessary because safety assertions typically involve these sensors. Then the interlock precondition for operation *Op* is given by the schema expression $PreSafeOp \mathrel{\widehat=} \exists\, S' \bullet Op \wedge \Xi Sensors$. In our example,

$$
\begin{array}{|l}
\hline
\textit{CyclotronSensors} \\
\hline
mainfld\_current : CURRENT \\
door : DOOR \\
\hline
\end{array}
$$

$$PreSafeTurnOnBeam \mathrel{\widehat=} \exists\, Cyclotron' \bullet TurnOnBeam \land \Xi CyclotronSensors$$

We obtain

$$
\begin{array}{|l}
\hline
\textit{PreSafeTurnOnBeam} \\
\hline
Cyclotron \\
\hline
rf \neq disabled \\
door = closed \\
mainfld = on \\
\hline
\end{array}
$$

These preconditions can be conjoined with the naive operation definition to obtain the intended definition: $SafeTurnOnBeam \mathrel{\widehat=} TurnOnBeam \land PreSafeTurnOnBeam$

It is useful to compare the interlock conditions computed from the state schema by this method to the interlocks recommended by the designers, based on their understanding of the system. Disagreement may indicate that the system safety assertions are not complete (or are too restrictive), or the operation is not fully described.

There are legitimate reasons why the computed interlock conditions might not agree with designers' recommendations. It is sometimes necessary to add interlocks beyond those entailed by the system safety assertions, in order to prevent certain transitions between states, even though the states themselves are sometimes permitted.

4 Limitations of the basic framework

The basic framework presented in section 3 can describe most of our requirements, but it is not very useful as a practical specification style. Its disadvantages arise because all the system state variables appear in a single state schema. Real process control systems have hundreds or thousands of state variables. Moreover, the number of operations and the specification for each would have to be very large because there are so many variables and conditions to consider.

5 A framework based on components

Most of our system's size derives from repetition of similar components. We can make our specification much shorter and easier to grasp by identifying the components, describing them separately, and then combining them. Each kind of component is specified using the basic framework presented in section 3, with its own state, operations, and interlocks.

Each kind of component can be considered an *abstract data type* or, to use the terminology of the popular object-oriented programming movement, a *class*. Several notations based on Z add constructs intended to support object-oriented programming [17]. We find that ordinary Z [16] serves well as a notation for specifying object-oriented programs.

The following sections describe some components we have found useful for our application. Subsequent sections show how the component specifications are combined into a system specification.

5.1 Analog control parameters

The three state variables $mainfld\_setpoint$, $mainfld\_preset$, $mainfld\_current$ that appeared in the *Cyclotron* schema in section 3 reveal a pattern that appears in many other contexts. We define a schema for this recurring pattern, which we call a *control parameter* or simply a *parameter* (in this report we use the word "parameter" in this sense, not the programming languages sense).

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Param}} \\
\hline
preset, setpoint, value : SIGNAL \\
\hline
\end{array}$$

It is useful to define a schema for the situation where the parameter's value is nearly equal to the setpoint.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ParamValid}} \\
\hline
Param \\
\hline
\mid setpoint - value \mid \leq \epsilon \\
\hline
\end{array}$$

5.2 Power supplies and servomotors

Many of the state variables in our system are devoted to about forty power supplies that provide current to the magnets that confine, focus and steer the beam. The main field supply discussed in section 3 is just one of these. Here is a slightly more realistic generalization; this model also includes the contactor that connects the supply to its power source, and represents the various faults that induce the *disabled* and *error* states. The control law says that current cannot flow when the contactor is open. The safety assertions say that we must not try to drive current when faults exist or the contactor is open.

$$SWITCH ::= open \mid closed$$

$$FAULT ::= overload \mid line\_voltage \mid overtemp \mid ground\_short$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{PS} \\
\hline
Param \\
contactor : SWITCH \\
faults : \mathbb{P}\, FAULT \\
\hline
contactor = open \Rightarrow value \leq \epsilon \\
\\
faults \neq \Rightarrow setpoint = 0 \\
contactor = open \Rightarrow setpoint = 0 \\
\hline
\end{array}
$$

Explicitly modelling the contactor and faults reveals that the status values of section 3 (*disabled*, *off* etc.) actually indicate different power supply states, so we no longer need an explicit *status* variable. The supply is *Off* when the contactor is open and there are no faults:

$$
\begin{array}{|l}
\multicolumn{1}{l}{Off} \\
\hline
PS \\
\hline
contactor = open \\
faults = \\
\hline
\end{array}
$$

The supply is *On* when the contactor is closed, there are no faults, and *setpoint* and *value* (nearly) equal *preset*. Note that power supplies can use ("inherit") any properties defined for parameters, such as *ParamValid*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{On}} \\
\hline
PS \\
\hline
\mathit{ParamValid} \\
\mathit{contactor} = \mathit{closed} \\
\mathit{setpoint} = \mathit{preset} \\
\mathit{faults} = \\
\hline
\end{array}
$$

It is easy to define operations in terms of these states.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{TurnOn}} \\
\hline
\Delta PS \\
\hline
\theta PS \in \mathit{Off} \\
\theta PS' \in \mathit{On} \cup \mathit{Error} \\
\hline
\end{array}
$$

Several other kinds of components besides power supplies include control parameters. For example, in servomotors the signals represent position, not current.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Servo}} \\
\hline
\mathit{Param} \\
\mathit{enable} : \mathit{MODE} \\
\text{Other state variables specific to servomotors} \ldots \\
\hline
\end{array}
$$

5.3 Discrete indicators

It is convenient if every state variable in the system is handled in a uniform way, as part of an instance of some class of components. Those few system-level state variables that don't belong to any obvious component can be handled by defining simple "components" with only one state variable.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Indicator}} \\
\hline
\mathit{status} : \mathit{INDICATOR} \\
\hline
\end{array}
$$

5.4 Combining the components

With several kinds components now in hand, we return to the system level. Every component has a name. Each class of components in the system is modelled as a function from names to instances of the state schema for that class. This example shows only three kinds of components; the real system has many more.

$[NAME]$

$$ps, s, i : \mathbb{P}\, NAME$$

Cyclotron

$$\begin{array}{l}
supply : NAME \rightarrowtail\!\!\!\!\!+ \; PS \\
servo : NAME \rightarrowtail\!\!\!\!\!+ \; Servo \\
indicator : NAME \rightarrowtail\!\!\!\!\!+ \; Indicator \\
\hline
\mathrm{dom}\, supply = ps \\
\mathrm{dom}\, servo = s \\
\mathrm{dom}\, indicator = i \\
\\
supply\ rf \in On \Rightarrow (indicator\ door).status = closed \\
supply\ rf \in On \Rightarrow supply\ mainfld \in On \\
\\
\text{Other system level laws} \ldots
\end{array}$$

For each class of component, there is a set that names all the components of that class. The first group of predicates says that the roster of components in the system is fixed. Therefore, each maplet of the form $name \mapsto \theta Component$ can be regarded as a persistent object. This is a central idea in our object-oriented specification style for Z.

All of the state variables and most of the predicates from the basic framework are now inside the various components, so the system state schema can be much smaller. However, laws that relate state variables in different components can only be expressed at the system level. These include the two safety assertions discussed in section 3.

6 Some useful idioms

Specifying the operations of a system described this way requires several constructions in the Z notation that are not obvious. We call them *idioms*. These idioms are not described in the reference manual [16] nor taught in textbooks [14]; they must be gleaned from case studies [12] or technical reports [11]. Here are two useful ones.

6.1 Promotion

Much useful behavior can be modeled at the component level. However, methods defined at the component level are not, by themselves, meaningful at the system level. For example, at the system level it makes no sense to merely turn on a power supply; it is necessary to say *which* supply. Component-level operations that must be made available at the system level can be adapted by applying a Z idiom called *promotion* [12, 11].

First, for each type of component we have to define a *framing schema*, where the identifier of the component of interest is an input parameter. For power supplies, the framing schema is $Cyclo\Phi PS$.

$$
\begin{array}{|l}
\multicolumn{1}{l}{Cyclo\Phi PS} \\
\hline
\Delta Cyclotron \\
\Delta PS \\
ps? : NAME \\
\hline
ps? \in ps \\
\theta PS = supply\ ps? \land \theta PS' = supply'\ ps? \\
\hline
\end{array}
$$

Then the operation to turn on the main field magnet is:

$$
\begin{array}{|l}
\multicolumn{1}{l}{TurnOnMainfld} \\
\hline
Cyclo\Phi PS \\
TurnOn \\
\hline
ps? = mainfld \\
\hline
\end{array}
$$

Other power supply operations can be promoted in the same way.

Sometimes, additional predicates must be added to promoted operations to account for requirements that emerge at the system level.

$$
\begin{array}{|l}
\textit{TurnOnBeam} \\
\hline
\textit{Cyclo}\Phi \textit{PS} \\
\textit{TurnOn} \\
\hline
\textit{ps}? = \textit{rf} \\
(\textit{indicator door}).\textit{status} = \textit{closed} \\
\textit{supply mainfld} \in \textit{On} \\
\hline
\end{array}
$$

The additional interlock preconditions here could have been calculated from the system safety assertions by the method described in section 3.3.

6.2 Operations on multiple components

Other system level operations are obtained by performing the same method on multiple components. For example, a common operation is turn on all the power supplies in some subsystem, say Beam Line A. This is provided at a single button, to save the operator the trouble of switching each supply on individually. It can be expressed by another Z idiom:

$$
\begin{array}{|l}
\textit{TurnOnBLA} \\
\hline
\Delta \textit{Cyclotron} \\
\hline
\forall \textit{ps} : \textit{blaps} \bullet \exists \textit{TurnOn} \bullet \theta \textit{PS} = \textit{supply ps} \land \theta \textit{PS}' = \textit{supply}' \textit{ ps} \\
\hline
\end{array}
$$

7 Subsystems, conditions, and modes

In addition to components, we use a few other ideas to organize the specification. Other authors have noted the usefulness of conditions and modes [5]. Subsystems are also helpful.

7.1 Subsystems

The various subsystems include the RF system, the cyclotron proper, the three beamlines, the two treatment rooms, etc. Each is simply a collection of components, identified by their names.

$$\mid\ rfsys, cyclo, bla, blb, blc, iso, fix : \mathbb{P}\ NAME$$

7.2 Conditions

It is useful to define schemas to abbreviate conditions that appear frequently in the specification. Some conditions are quite simple.

$$\begin{array}{|l}
\hline
\textit{BeamOn} \\
\textit{Cyclotron} \\
\hline
\textit{supply rf} \in \textit{On} \\
\hline
\end{array}$$

Others are more complex; subsystems often appear in these definitions. For example, the cyclotron is ready when the vault door is closed, all of its power supplies have been switched on and are free of faults, and all their currents are near their setpoints.

$$\begin{array}{|l}
\hline
\textit{CycloReady} \\
\textit{Cyclotron} \\
\hline
(\textit{indicator door}).\textit{status} = \textit{closed} \\
\textit{supply}(\!|\textit{cyclo} \cap \textit{ps}|\!) \subseteq \textit{On} \\
\hline
\end{array}$$

This is a precondition for many operations. Here is a more realistic specification for turning on the beam.

$$\begin{array}{|l}
\hline
\textit{TurnOnBeam} \\
\textit{Cyclo}\Phi \textit{PS} \\
\textit{TurnOn} \\
\hline
\textit{ps}? = \textit{rf} \\
\textit{CycloReady} \\
\textit{BeamOn}' \\
\hline
\end{array}$$

As this example shows, redundant conditions can be included to help make the intended effect clear to the reader.

7.3 Modes

Our cyclotron can be operated in different *modes*. Each mode is characterized by the destination and purpose of the beam. The beam can be delivered to two treatment rooms or an isotope production station. It can be used to treat patients, or for experiments and testing.

Modes are a kind of condition, for example:

IsoTest

Cyclotron

Isocentric room, test mode ...

Modes are important because the control laws and safety assertions depend on which mode is selected. In order to turn on the beam in a room, the beam line to that room must be ready, and different safety interlocks must be cleared to treat a patient than to run an experiment with no people in the room. This is expressed by using modes and other conditions to write the control laws and safety assertions.

SafeCyclotron

Cyclotron

$$IsoTest \wedge BeamOn \Rightarrow CycloReady \wedge BLAReady \wedge IsoReady$$
$$IsoTreat \wedge BeamOn \Rightarrow CycloReady \wedge BLAReady \wedge IsoSafe$$

Laws for other modes ...

These predicates concisely express many important properties. For example, if any of the conditions included in *IsoSafe* becomes false while the beam is on in *IsoTreat* mode, the beam must turn off.

Modes and conditions also appear in the operation schemas:

$$
\begin{array}{|l}
\hline
\textit{SafeTurnOnBeam} \\
\textit{TurnOnBeam} \\
\hline
\textit{IsoTest} \Rightarrow \textit{CycloReady} \land \textit{BLAReady} \land \textit{IsoReady} \\
\textit{IsoTreat} \Rightarrow \textit{CycloReady} \land \textit{BLAReady} \land \textit{IsoSafe} \\
\text{Preconditions for other modes} \ldots \\
\hline
\end{array}
$$

8 User interface

Our complete specification will include a schema for every operation that users can invoke which might change the values of any state variables. Therefore, we must write a schema for every control panel button and every on-screen menu selection.

Our complete specification will also include some schemas for operations that occur spontaneously when the values of certain sensor variables change. Turning off the beam at the end of a treatment, when integrating sensors indicate that the prescribed dose has been delivered, is one example.

When users attempt operations that are interlocked, the system state does not change. Pressing the BEAM ON button turns on the RF drive if all the interlocks relevant to the selected mode are clear; otherwise, nothing happens. Therefore, the full specification for this and every other operation must be *total*; they must cover both possibilities. This is expressed:

$$T\_TurnOnBeam \mathrel{\widehat{=}} SafeTurnOnBeam \lor \Xi Cyclotron$$

The active interlocks, conditions and modes are displayed at the control console so operators can see which operations are enabled.

Our specification implicitly determines that some sequences of operations are permitted and others are not possible, because in most states some variables act as interlocks to inhibit certain operations. Users may select operations in any sequence they wish, subject only to the sequencing constraints imposed by the preconditions. There is no other "flow of control."

Graphic notations such as state transition diagrams can help make sequencing constraints clear, and might be a useful complement to the Z texts.

The translations between internal state variable values and their representations in user interface devices such as analog meters or workstation displays are among the input/output operations that we have formally specified in another report [7]. Other details — whether a particular operation is invoked by pressing a button on a control panel, or selecting a menu option at a workstation — are described in prose and diagrams [9]. We do not believe it would be useful to formally specify the "look and feel" aspects of the user interface.

9 Progress report and preliminary evaluation

At this writing (September 1992) our formal specification is not complete, but we are confident that all functional requirements documented in the informal specification [8, 9] can be formalized using techniques described in this report (and a few more that we have omitted for brevity). All that remains is to finish filling in the framework.

We have only attempted to formalize the functional aspects of our system. The Z notation does not provide built-in facilities for representing time or concurrency. If we decide to formalize these features we will select a notation suited for them.

Much of the effort in developing large applications like ours is devoted to enumerating the system state variables and describing the operations that must be provided, taking care that nothing is omitted and no inconsistencies are introduced. The Z notation provides a discipline for organizing this work that is supported by a de-facto standard [16], several good textbooks, and robust tools for document preparation, syntax and type checking.

Z is particularly effective for systems whose size derives from repetition of components which are not identical but share many features in common. The Z schema calculus permits recurring features to be described with texts that apply to all, supplemented with brief texts that address the differences. As a result, definitions such as *SafeTurnOnBeam* in section 7.3 can be quite compact even though they actually describe hundreds of state variables.

We have already found the formal texts to be useful as descriptive documentation. We hope their brevity will help us build an economical implementation. The possibility that they might also support safety analyses and formal development of the implementation is an additional bonus.

10 Acknowledgements

The author thanks Norman Delisle, David Garlan and Mike Spivey for explaining some Z idioms and commenting on earlier versions of this work.

References

[1] Dan Craigen. FM89: Assessment of formal methods for trustworthy computer systems. In *12th International Conference on Software Engineering Proceedings*, pages 233–235. IEEE Computer Society, 1990.

[2] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley, second edition, 1991.

[3] David Garlan and Norman Delisle. Formal specifications as reusable frameworks. In D. Bjorner, C. A. R. Hoare, and H. Langmaack, editors, *VDM '90: VDM and Z — Formal Methods in Software Development*, pages 150–163, Kiel, FRG, April 1990. Third International Symposium of VDM Europe, Springer-Verlag. Lecture Notes in Computer Science number 428.

[4] Ian Hayes, editor. *Specification Case Studies*. Prentice Hall International, Englewood Ciffs, NJ, 1987.

[5] K.L. Heninger. Specifying software requirements for complex systems: new techniques and their application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, 1980.

[6] Jonathan Jacky. Formal specifications for a clinical cyclotron control system. In Mark Moriconi, editor, *Proceedings of the ACM SIGSOFT International Workshop on Formal Methods in Software Development*, pages 45–54, Napa, California, USA, May 9–11 1990. (Also in *ACM Software Engineering Notes*, 15(4), Sept. 1990).

[7] Jonathan Jacky. Formal specification and development of control system input/output. Technical Report 92-05-02, Radiation Oncology Department, University of Washington, Seattle, WA, May 1992.

[8] Jonathan Jacky, Ruedi Risler, Ira Kalet, and Peter Wootton. Clinical neutron therapy system, control system specification, Part I: System overview and hardware organization. Technical Report 90-12-01, Radiation Oncology Department, University of Washington, Seattle, WA, December 1990.

[9] Jonathan Jacky, Ruedi Risler, Ira Kalet, Peter Wootton, and Stan Brossard. Clinical neutron therapy system, control system specification, Part II: User operations. Technical Report 92-05-01, Radiation Oncology Department, University of Washington, Seattle, WA, May 1992.

[10] Matthew S. Jaffe, Nancy G. Leveson, Mats P. E. Heimdahl, and Bonnie E. Melhart. Software requirements analysis for real-time process control systems. *IEEE Transactions on Software Engineering*, 17(3):241–258, March 1991.

[11] Ruaridh Macdonald. Z usage and abusage. Technical Report 91003, Royal Signals and Radar Establishment, St. Andrews Road, Malvern, Worcestershire, WR14 3PS, February 1991.

[12] Carroll Morgan and Bernard Sufrin. Specification of the UNIX file system. *IEEE Transactions on Software Engineering*, SE-10(2):128–142, March 1984. (Also appears in [4]).

[13] David Lorge Parnas and Jan Madey. Functional documentation for computer systems engineering (version 2). Technical report, Telecommunications Research Institute of Ontario (TRIO), McMaster University, Hamilton, Ontario, L8S 4K1, September 1991. CRL Report No. 237.

[14] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z.* Prentice Hall International (UK) Ltd, Hemel Hempstead, Hertfordshire, 1991.

[15] Ruedi Risler, Jüri Eenmaa, Jonathan P. Jacky, Ira J. Kalet, Peter Wootton, and S. Lindbaeck. Installation of the cyclotron based clinical neutron therapy system in Seattle. In *Proceedings of the Tenth International Conference on Cyclotrons and their Applications*, pages 428–430, East Lansing, Michigan, May 1984. IEEE.

[16] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, New York, 1989.

[17] Susan Stepney, Rosalind Barden, and David Cooper. A survey of object orientation in Z. *Software Engineering Journal*, 7(2):150–160, March 1992.