Formal Specifications for a Clinical Cyclotron Control System

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

Abstract

This report describes preliminary experience writing formal specifications for the control system for a cyclotron and neutron radiation therapy apparatus. This effort is motivated by high reliability and safety requirements, and a need for concise, authoritative documentation to support coding, user instruction, and testing. Software development practices for therapy machines and physics reserach acclerators are reviewed. The operation of our machine from the point of view of the cyclotron operator is described. Many of the cyclotron operator's controls are well-matched to model-based notations such as Z and VDM. Sample specifications in Z are presented for representative operations of the cyclotron control programs. These notations provide no built-in way to represent the passage of time, and they cannot express some features of concurrent systems and event-driven systems. Alternative notations are discussed, including Petri Nets and Software Cost Reduction project (SCR) notation. We conclude that it is practical to attempt a comprehensive formal specification of our application, and anticipate that this will be a valuable supplement to traditional development practices.

1 Introduction

The Clinical Neutron Therapy System at the University of Washington is a cyclotron and radiation therapy facility that provides cancer treatments with fast neutrons, production of medical isotopes, and physics experiments [18]. The control system handles over one thousand input and output signals.

Devices under computer control include a 900 amp electromagnet and a 30 ton rotating gantry. The facility was installed in 1984 and includes a computer control system provided by the cyclotron vendor, which supports a clinically useful subset of radiation therapy treatment techniques. The University is developing a successor control system including all new computer hardware and software and some changes to the peripheral control hardware. The new system is motivated by requirements to support more elaborate treatments.

We have high reliability and safety requirements. There is growing recognition that the development of software which controls medical devices is not sufficiently systematic, that this is adversely affecting costs and safety, and there is much room for improvement [1]. The urgency of these concerns was illustrated by a series of incidents between 1985 and 1987 in which at least three serious injuries and two deaths were caused by failures of the computer control system in one model of a commercial radiation therapy machine [12].

We are attempting to achieve high reliability and safety by applying rigorous software development and quality assurance practices. We are intrigued by the possibility that formal software development techniques might result in additional improvements in reliability and safety beyond those achieved through traditional practices, namely, English-language specifications, subjective design and code reviews, and lots of testing. One aim of our control system project is to determine whether currently available formal development techniques will prove useful in our hands, or whether, as is sometimes claimed, they are "too difficult," or "not practical for large projects."

Other important design goals are improved maintainability and adaptability to future hardware and software modifications. Currently, we are limited by a paucity of documentation recording high-level design decisions. We often resort to inspecting wiring diagrams and over 60,000 lines of FORTRAN source code. This is very time-consuming and it is difficult to distinguish key requirements from non-essential features of the implementation (if the code says two events occur in a particular order, is that a requirement?). A major product of our effort will be concise and authoritative documentation.

This report describes initial experiences attempting to cast our informal specification [13] into formal notations. It is intended to acquaint the formal methods community with our application, and also to introduce the radiation therapy community to the formal approach.

2 Review of Software Development Practices for Accelerator Controls

Our project is apparently the first application of formal methods to an accelerator control system. The literature on radiation therapy machine control systems is very scant. A few reviews briefly discuss control system functionality [4, 14]; a few reports describe acceptance testing from the customer's point of view [17, 20]; none discuss design internals nor development practices. The commercial vendors do not publish their practices or experiences.

There is a larger literature is devoted to research accelerators, which share many features with therapy machines. A typical presentation (e.g., in [16]) emphasizes hardware organization. Discussion of software is limited to informal treatment of program structure, for example descriptions of important processes and subroutines. There is little interest in development methodology; the very concept of "specification" is practically absent from this literature.

3 A Brief Introduction to Cyclotron Controls

3.1 What the cyclotron operator does

Our facility is larger and more complex than a commercial accelerator; there are several operators, not just one. The cyclotron provides beam to two treatment rooms, each with its own therapy control console, as well as a third beamline devoted to isotope production. In commercial therapy machines beam production is largely automatic but our facility requires a cyclotron operator, who works at a control console similar to those found at research accelerators. The cyclotron operator is responsible for bringing up and shutting down the machine, switching the beam between the two treatment rooms and the isotope production line, and restoring normal machine operation after shutdowns or interruptions caused by faults.

To obtain satisfactory performance, the cyclotron operator must adjust, or *tune*, about eighty analog quantities which the operators call *cyclotron control parameters* or just *parameters* (in this paper we use the term "parameters" in this sense, not in the programming languages sense). The parameters are physical quantities such as ion source gas flow, deflector voltage, switching magnet current, etc. The cyclotron designers calculated sets of theoretically ideal parameter values, or *tuning tables*, for each combination of particle and energy. In practice these idealized values are only starting points; variations in beamline characteristics imposed by experimental or treatment configurations and the vagaries of nature require frequent manual tuning. A few parameters are partially controlled by automated servo loops in some operating modes, but the operator's participation is still essential.

3.2 What the cyclotron control programs do

The *cyclotron control programs* are the control subsystem dedicated to assisting the cyclotron operator. Inputs and outputs of this subsystem include a terminal keyboard, two video dispays, numerous buttons and lamps on the cyclotron operator's console, and the digital and analog interfaces to the controlled parameters.

The cyclotron control programs *scale* quantities between the engineering units needed by the operator (e.g. cc per minute, kilovolts, amps) and the digitally encoded bit patterns on the input and output converters. They *multiplex* subsets of parameters into a smaller number of physical dispays and controls. Early control consoles provided hard-wired meters, switches and dials for each parameter, but it is impractical to place over one hundred parameters within reach of the operator. Our console provides dedicated indicators and controls for only a few essential parameters; others are selected by the operator, singly or in groups, and assigned to a relatively small bank of meters, dials, and video displays. The control programs *save and restore* collections of parameter settings, so the operator need not repeatedly write down and individually set each one. The programs perform automatic *startup, shutdown, and sequencing*, so that collections of related parameters that are usually operated as a group (e.g. the three collections of magnets that focus the beam down each of the beamlines) can be operated as a unit, without requiring the operator to throw dozens of switches in the right order. The programs also perform a certain amount of *fault detection*, turning off the beam or shutting down subsystems when faults occur, and *interlocking*, preventing certain subsystems from being turned on when it might be unsafe to do so. They implement different *operating modes*, depending on which beamlines are in use, and whether treatments, isotope production, or physics experiments are in progress. Finally, the control programs *log* operating data and error reports to disk files for subsequent record keeping, performance analysis, and troubleshooting.

Patient and operator safety shutdown systems and interlocks at our facility are nonprogrammable hardwired controls, but these functions are backed up, and are displayed to the operators, through the computer. In addition, much internal equipment protection is provided solely by the computer. The cyclotron delivers up to 3.5 kilowatts in a beam about 5 millimeters wide; a poorly tuned beam may strike the beamline walls and can burn through tens of thousands of dollars worth of fragile components in a few seconds.

4 Model-based formal specifications: Z and VDM

Many functions of the cyclotron control programs appear well-suited to formal specifications in notations based on typed set theory and predicate calculus. These are called *model-based* notations because the developer defines data structures that are supposed to model the internal state of the system, as opposed to *property-oriented* or *algebraic* approaches which model only the input-output behavior. The model-based approach seems best for us because the internal states are the essential characteristics of the facility; many inputs and outputs may be changed to suit convenience.

The best-known model-based notations are Z [7, 19] and VDM [11, 5]. Both have similar expressive power, but as of this writing Z appears to be gathering more published tutorials and case studies. The distinguishing feature of Z is the *schema calculus*, which provides a convenient way to build up large specifications from textually separate components called *schemas*. On the other hand, publications from the VDM school provide more guidance for implementing their specifications in traditional imperative programming languages. We use Z in this manuscript because we admire the expository style it encourages.

In both notations, specifications consist of two principle components: descriptions of abstract data structures that model the internal state, and definitions of operations that manipulate the state. The data descriptions resemble data declarations in strongly-typed programming languages such as Pascal, but are usually more abstract; one may declare an ordered sequence without committing to an array, linked list, or some other implementation that is directly supported by an executable programming language. Each operation definition includes a section that resembles a procedure declaration in a programming languages. But, where a programmer would write an executable procedure body, the specifier writes logical assertions that describe the *effects* of each operation.

5 Model-based specification of cyclotron control programs

5.1 Scope

We present sample specifications for some essential operations that are required to run the beam inside the cyclotron vault. This is sufficient to produce isotopes but not to perform treatments. Running beam outside the vault into either treatment room requires that several nonprogrammable hardwired interlocks associated with treatments (including keys controlled by the therapy technologists) be clear. Therefore, our specifications here need not consider interaction with therapy operations and do not have critical human safety implications. However, they do have equipment protection implications. The operations specified here are simplified for purposes of exposition, but several important issues are considered.

5.2 Caveats

The author is not an experienced Z user. The fragments presented here may seem awkward and naïve to Z experts. Moreover, these samples have not been run through a type-checker, and may contain errors in Z usage. They are only intended to show how some of the control system requirements might be expressed in Z style, and to reveal some of the difficulties encountered by self-taught practitioners.

5.3 System state

There are about one hundred and twenty cyclotron control parameters, whose names were assigned by the cyclotron vendor: GASFLOW, DFLVOLT, SWTMAGN, etc. The names for these parameters constitute a Z set:

$$PNAME = \{GASFLOW, DFLVOLT, \dots\}$$

The system state is composed primarily of several quantities associated with each parameter. Most parameters have a quantity called (again following the vendor's nomenclature) *pset*, which is the control parameter output value most recently written to its digital-to-analog converter (DAC). The central problem in specifying the cyclotron controls is to define each *pset* as a function of time, the operators' activities, and potentially hundreds of other quantities, including the input values of other parameters, digital inputs such as interlocks, etc.

An important influence on the value of *pset* is *psetting*, which is the output value most recently requested by the operator; however, *pset* and *psetting* in general are *not* equal. The quantity *pread* is the input value most recently read from the parameter's analog-to-digital converter (ADC); *phigh* and *plow* define the upper and lower boundaries of a *window* of permissible values for *pread*, outside of which a fault is considered to exist.

All of these quantities are described by the Z schema:

```
┌─ CycloParams ──────────────────────
│ pset, psetting, pread, phigh, plow :
│              PNAME → FIX
└────────────────────────────────────
```

This schema says that the collection of cyclotron control parameters is a collection of functions that associates each parameter name with a named quantity. This usage of function differs from that encouraged by many programming languages, in which "functions" are methods for computing results, given some arguments. In contrast with that view, but consistent with ordinary mathematical usage, Z functions may be thought of as static objects pairing elements in a domain with elements in a range. The range of each function is declared as type FIX, the subrange of integers that can be represented by the hardware device registers in the ADC's and DAC's.

Clearly, much more is needed. We frequently display the value of a quantity in a form that is useful to the operator: a decimal number expressed in engineering units. We need to associate each parameter with its units and a conversion formula. These are also represented by Z sets:

$$UNIT = \{Amps, kVolts, cc/min, mbar, \ldots\}$$

$$FORM = \{Scale, Offset, Poly, Table, \ldots\}$$

Now we use *Schema inclusion* to indicate that the earlier definition of *CycloParams* is being extended with additional components:

$$
\begin{array}{|l}
\textit{CycloParams} \\
\hline
\textit{CycloParams} \\
units : PNAME \rightarrow UNIT \\
formula : PNAME \rightarrow FORM \\
a, b : PNAME \nrightarrow FLOAT \\
convert : PNAME \rightarrow (FIX \rightarrow FLOAT) \\
\hline
units = u_0 \wedge formula = f_0 \\
a = a_0 \wedge b = b_0 \wedge convert = c_0 \\
\\
\mathrm{dom}\, a = \mathrm{dom}\, formula \lhd \{Offset, Poly\} \\
\mathrm{dom}\, b = \mathrm{dom}\, formula \lhd \{Scale, Offset, Poly\} \\
\\
formula = Scale \Leftrightarrow \\
\quad convert = (\lambda\, x : FIX, b : FLOAT \bullet bx/M) \\
formula = Offset \Leftrightarrow \\
\quad convert = (\lambda\, x : FIX, a, b : FLOAT \bullet \\
\qquad a + bx/M) \\
formula = Poly \Leftrightarrow \\
\quad convert = (\lambda\, x : FIX, a, b : FLOAT \bullet \\
\qquad a + b^2 x/M) \\
\\
\quad \ldots \\
\quad (etc.) \\
\hline
\end{array}
$$

Each parameter now has conversion function *convert*. Note that the *value* of the function *convert* for any parameter is itself a function. The values of the functions *formula*, a, and b determine the shape of each parameter's *convert*. We declare a, b and the range of the conversion function to be *FLOAT* to indicate that they may assume values that may not be able to be represented in the DAC and ADC device registers.

This schema has a predicate section (below the horizontal line) which places constraints upon the values that may be assumed by the objects named in the declaration section, also called the *signature* (above the line). The first group of predicates indicates that the variables *units*, *formula*, a, b and *convert* are supposed to behave as constants. For example, the function u_0 would be defined in Z as follows:

$$
\begin{array}{|l}
u_0 : PNAME \rightarrow UNIT \\
\hline
u_0 = \{GASFLOW \mapsto cc/min, \\
\qquad DFLVOLT \mapsto kVolts, \ldots\}
\end{array}
$$

Table 1: Tabular representation of *CycloParams* schema

In an implementation, the functions u_0, f_0, a_0, b_0 and c_0 might be hard-coded into the program text or (better) read from a configuration file at system startup.

The functions a and b are declared with the symbol $\nrightarrow$ to indicate that they are *partial functions* which are not defined for some values of *PNAME*. The second group of predicates in *CycloParams* uses the range restriction symbol $\lhd$ to indicate that they are only defined for parameters associated with certain values of *FORM*.

The third group of predicates describe how the conversion functions are determined by the other components. They assume that mixed-mode arithmetic involving *FIX* and *FLOAT* is defined, and that the results can always be represented as a *FLOAT* (M is a global constant determined by the word size of the registers). It may seem that the *formula* component is redundant with *convert*; the intent is to convey that there are only a few kinds of conversion formulae (currently there are eight), even though each parameter has a different conversion function.

The *CycloParams* schema reveals that a large part of the task before us is to actually enumerate the elements of all the sets *PNAME*, *UNIT* and *FORM*, and assign all the values to u_0, f_0, a_0, b_0 and c_0. Rather than include these hundreds of items in the Z text (as in the definition of u_0 shown above) we maintain the information in a database so the cyclotron operating characteristics can be conveniently modified and examined independently of the Z text [9]. For example, the predicates in schema *CycloParams* describe Table 1.

Much of our use of Z so far can be viewed as a method for documenting the contents and meaning of our database. This alone is quite valuable; since the notation is described in the literature and supported by tools, it saves us from having to invent, explain and support one ourselves.

5.4 Some essential operations

Z models system behavior as a collection of discrete operations that are also written as schemas.

We need an operation that displays the value of a parameter to the user:

$$
\begin{array}{|l}
\hline
\textit{DisplayParam} \\
\Xi\, \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\textit{displayed\_pread}! : \textit{FLOAT} \\
\hline
\textit{displayed\_pread}! = \\
\qquad \textit{convert param}?(\textit{pread param}?) \\
\hline
\end{array}
$$

The Ξ indicates that this operation does not change the state of *CycloParams*; *param*? is the input to the operation (indicated in Z by ?); *displayed_pread*! is the output. We chose to make the displayed value an output, rather than a component of the system state, to indicate that the implementation is not required to maintain conversions of all quantities at all times, but is free to perform only those which are needed. The predicate says that the output is obtained by selecting the appropriate element from the range of the conversion function (in most implementations this would be accomplished by calculating the needed value from the conversion function). In Z, function application may be indicated by simple juxtaposition of function name and argument; in this example we use parentheses to help make it clear that first the values of *convert* and *pread* associated with *param*? are found, and then the resulting conversion function is applied to the value of *pread*.

There is a complementary input operation to set the *psetting* from an input value expressed in engineering units:

$$
\begin{array}{|l}
\hline
\textit{SetParam} \\
\Delta\, \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\textit{new\_psetting}? : \textit{FLOAT} \\
\hline
\textit{convert param}?(\textit{psetting}'\ \textit{param}?) = \\
\qquad \textit{new\_psetting}? \\
\hline
\end{array}
$$

The Δ indicates that this operation changes the state of *CycloParams*; the $'$ on $\textit{psetting}'$ indicates the value of that component of the state *after* the operation has completed. The predicate says that input conversion is the inverse of output conversion. It is not necessary to provide an explicit input conversion formula, in fact writing out a particular formula, e.g. $M(y-a)/b$ would not convey the essential property of inverse-ness. This example illustrates the economy of expression provided by non-executable notations such as Z.

The *DisplayParam* operation is usually not applied to just one quantity at a time; instead, it is usual to display groups of related quantities on one of the cyclotron operator's video screens. The *Status Display Screen* is repetitively updated in (almost) real time; it is the computer-age replacement for a bank of analog meters. This schema says that the control system provides a pre-defined set of named screen designs, each showing a collection of related parameters:

$$\textit{DNAME} = \{\textit{RFSystem}, \textit{VacuumSystem}, \dots\}$$

$$
\begin{array}{|l}
\hline
\textit{StatusDisplay} \\
\textit{screen} : \mathbb{P}\, \textit{PNAME} \\
\textit{designs} : \textit{DNAME} \rightarrow \mathbb{P}\, \textit{PNAME} \\
\hline
\textit{designs} = d_0 \\
\hline
\end{array}
$$

A typical operation is to select a named screen design and display it on the status display screen (replacing whatever was previously displayed there):

$$
\begin{array}{|l}
\hline
\textit{SelectStatus} \\
\Delta\, \textit{StatusDisplay} \\
\textit{design}? : \textit{DNAME} \\
\textit{background}! : \textit{PNAME} \nrightarrow \textit{UNIT} \\
\hline
\textit{screen}' = \textit{designs design}? \\
\textit{background}! = \mathrm{dom}\, \textit{screen}' \rhd u_0 \\
\hline
\end{array}
$$

Here *design*? is the input from the operator requesting a particular named collection of parameters. This operation also sends output *background*! to the display, painting the fixed background that persists through successive display updates. This background usually includes text depicting the names and engineering units of each parameter. We represent this as a function pairing parameter names with units. The function is partial because not all parameters are present in any one design. The first expression in the predicate says that the requested design is stored in the status display's state. The latter predicate says that the parameters whose names and units are depicted on the display background are the very ones in the screen design requested in the input, and that the correct association of parameter names and units is made by using the domain restriction operator $\rhd$ to select the appropriate subset of the previously defined function u_0.

Then we need a similar operation to update the display with the current values of all the associated quantities, using the appropriate conversion formulae, something like:

$$
\begin{array}{|l}
\hline
\textit{UpdateStatus} \\
\Xi\, \textit{StatusDisplay} \\
\textit{updates}! : \textit{PNAME} \nrightarrow \textit{FLOAT} \\
\hline
\textit{updates}! = \mathrm{dom}\, \textit{screen} \rhd \textit{pread} \fatsemi \\
\qquad \lambda\, \textit{DisplayParam} \bullet \textit{displayed\_pread}! \\
\hline
\end{array}
$$

Here again, the output is a function, to indicate that the correct pairing of values with parameters must be maintained (e.g., by positioning the values next to the corresponding labels and units on the display screen). The predicate is supposed to convey that the values displayed by this operation are exactly those that would be obtained by applying the *DisplayParam* operation to each of the parameters in the set *screen*. The $\fatsemi$ indicates function composition, and $\lambda\, \textit{DisplayParam} \bullet \textit{displayed\_pread}$ indicates the projection function that extracts the *displayed_pread*!

component from the *DisplayParam* schema (The predicate is not quite right because the range of *pread* is not the same type as the domain of $\lambda \ldots$).

It is useful to remark what is *not* stated in the previous two schemas. *SelectStatus* does not say how the operator actually performs the selection; neither schema describes what the screen actually looks like. It is implicitly understood, but not expressed in Z, that the *DisplayStatus* operation is repetitively applied, and the results are somehow portrayed on the screen, until some other display is selected.

There are analogous schemas *TUModule*, *SelectTUM*, and *ScanTUM* to connect groups of parameters to a bank of input dials called *Tuning Modules* which are repetitively sampled at a high rate.

5.5 Ordering constraints

Our specification thus far models each parameter as a box (like a lab bench power supply) that has a dial (*psetting*), a meter (*pread*) and an output terminal (*pset*). We still can't control the cyclotron because we haven't defined any operations that change the *pset*'s. We need an an ON/OFF switch, and an operation to turn on the switch. An obvious solution is:

$$
\begin{array}{|l}
\textit{ChangeParam} \\
\hline
\Delta \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\hline
\textit{pset}'\ \textit{param}? = \textit{psetting}\ \textit{param}? \\
\hline
\end{array}
$$

However, naive implementation of this schema could destroy the equipment! About forty of the parameters represent power supplies that deliver current, often 50 amps or more, to the magnets that steer and focus the beam: Power supply switching is implemented by relay contactors at the output of each supply. If these relays should close when the *pset* demands a high current, the resulting sudden rush of current creates an inductive transient that could damage the magnets and/or the supplies. We include this information in the relevant schemas:

$$PS = \{\textit{MAINFLD}, \textit{SWTMAG}, \ldots\}$$

$$\textit{switch\_state} = \{\textit{OPEN}, \textit{CLOSED}\}$$

$$
\begin{array}{|l}
\textit{Cycloparams} \\
\hline
\textit{CycloParams} \\
\textit{switch} : \textit{PNAME} \rightarrow\!\!\!\!\!+\; \textit{switch\_state} \\
\hline
\mathrm{dom}\ \textit{switch} = PS \\
\hline
\end{array}
$$

We need to specify a *software interlock* that allows the contactors to close only when the *pset* is near zero. Our solution is to specify two separate operations. This operation closes the contactors:

$$
\begin{array}{|l}
\textit{NormalSwitchPS} \\
\hline
\Delta \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\hline
\textit{param}? \in PS \\
\textit{pset}\ \textit{param}? \leq \textit{small} \\
\textit{switch}'\ \textit{param}? = \textit{CLOSED} \\
\hline
\end{array}
$$

The predicate says that this operation is meaningful only for power supplies, and that the switch is closed only if the power supply output is low. For the case where the output is too high, there is another schema:

$$
\begin{array}{|l}
\textit{TooHighPS} \\
\hline
\Xi \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\textit{error}! : \textit{MESSAGE} \\
\hline
\textit{param}? \in PS \\
\textit{pset}\ \textit{param}? > \textit{small} \\
\textit{error}! = \text{"}\texttt{Output too high;} \\
\qquad \texttt{power supply } \textit{param}?\texttt{ not switched}\text{"} \\
\hline
\end{array}
$$

It is understood that the error message is displayed and logged somehow. The two schemas are combined:

$$\textit{SwitchPS} \mathrel{\widehat=} \textit{NormalSwitchPS} \lor \textit{TooHighPS}$$

This example illustrates a use of the *schema calculus*. It is customary Z style to separate error handling in this way. The operation to change the power supply current is handled similarly:

$$
\begin{array}{|l}
\textit{NormalChangePS} \\
\hline
\textit{ChangeParam} \\
\hline
\textit{param}? \in PS \\
\textit{switch}\ \textit{param}? = \textit{CLOSED} \\
\hline
\end{array}
$$

(Note the inclusion of the *ChangeParam* schema)

$$
\begin{array}{|l}
\textit{OpenedPS} \\
\hline
\Xi \textit{CycloParams} \\
\textit{param}? : \textit{PNAME} \\
\textit{error}! : \textit{MESSAGE} \\
\hline
\textit{param}? \in PS \\
\textit{switch}\ \textit{param}? = \textit{OPEN} \\
\textit{error}! = \text{"}\texttt{Contactor open;} \\
\qquad \texttt{current } \textit{param}?\texttt{ not changed}\text{"} \\
\hline
\end{array}
$$

$$\textit{ChangePS} \mathrel{\widehat=} \textit{NormalChangePS} \lor \textit{OpenedPS}$$

Now we combine these operations:

$$\textit{TurnOnPS} \mathrel{\widehat=} \textit{SwitchPS} \fatsemi \textit{ChangePS}$$

The $\fatsemi$ indicates schema composition, and captures the effect of *SwitchPS then ChangePS*. This operation closes the contactors and then sets the output current to the most recently requested value of *psetting*.

5.6 Timing constraints: timeouts

Schema *TurnOnPS* does not yet capture all the essential features. It is possible that the *ChangePS* operation may not succeed, even if its preconditions are met. Some parameters are driven by mechanical mechanisms (e.g., sliders on variacs); the mechanism may jam or perhaps the drive belt falls off. The control program detects such occurences by noting that the corresponding *pread* has not approached its expected value after appreciable time has passed. We say that such an operation has *timed out*; an error should be indicated. Z and VDM do not provide any built-in way to express the passage of time. Other authors have noted the difficulties of representing time in Z [6] and VDM [2]. Bowen proposed a technique for dealing with timeouts in Z [3], which is outlined (as we understand it) here:

Define a type *INTERVAL* to represent time intervals, and use a variable *duration* which is understood to represent the amount of time elapsed since the operation begins:

$$\begin{array}{|l}
\hline
\mathit{TimelyChangeParam} \\
\mathit{ChangeParam} \\
\mathit{duration} : \mathit{INTERVAL} \\
\hline
\mathit{plow} \leq \mathit{pread}' \leq \mathit{phigh} \\
\mathit{duration} \leq \mathit{timelimit} \\
\hline
\end{array}$$

and

$$\begin{array}{|l}
\hline
\mathit{TimeOut} \\
\mathit{ChangeParam} \\
\mathit{duration} : \mathit{INTERVAL} \\
\mathit{error}! : \mathit{MESSAGE} \\
\hline
\mathit{duration} > \mathit{timelimit} \\
(\mathit{pread}' < \mathit{plow}) \lor (\mathit{pread}' > \mathit{phigh}) \\
\mathit{error}! = \text{"}\mathtt{Operation\ timed\ out}\text{"} \\
\hline
\end{array}$$

obtaining

$$\begin{array}{l}
\mathit{TimedChangeParam} \mathrel{\widehat{=}} \\
\qquad \mathit{TimelyChangeParam} \lor \mathit{TimeOut}
\end{array}$$

This is just a sketch; we actually require a different time limit for each combination of parameter and operation.

5.7 Timing constraints: time-varying functions

There are additional complications: If the power supply output current should change abruptly even while the relays are closed, damage from inductive transients may still occur. Internal circuitry protects some supplies, but others depend on the control system to gradually change, or *ramp* the output current to the new value. In other words, the time course of the transition of *pset* is constrained.

Here we approach the limits of notations such as Z that model systems as collections of discreet operations and provide no built-in way to represent the intervals while operations are in progress. One might propose adding to operation schemas a variable called *history* which is the time-course of the change in the output. For *ChangeParam* The function *history* is the sequence of pairs $(t_i, pset_i)$ observed at a set of times t_i while the operation is in progress. One could imagine predicates like:

$$\begin{array}{|l}
\hline
\mathit{RampPS} \\
\ldots \\
\hline
\forall\, t_i, t_j : \mathit{INTERVAL} \mid \\
\qquad t_i, t_j \in \operatorname{dom} \mathit{history} \land t_i \neq t_j \bullet \\
\qquad \mid \mathit{history}(t_i) - \mathit{history}(t_j)/(t_i - t_j) \mid \\
\qquad\qquad \leq \mathit{ratelimit} \\
\hline
\end{array}$$

The constraint on *history* is simple: the rate of change observed between any two intervals cannot be too large. This must be combined with a timeout constraint to ensure that the rate of change is not too slow.

On the other hand, it might be better to consider whether time-varying outputs should be modelled by some other technique entirely.

5.8 Combined operations with serial and parallel components

Usually the operator selects values for all of the parameters while the system is in a power-conserving quiescent state called *Standby 1*, in which most power supplies are turned off. The operator turns them all on at once by pressing a button labelled, *Standby 2*. The intent is that all the contactors will close, and then all of the supplies will ramp up. The operation is considered complete when every parameter's *pread* is within the window determined by its *phigh* and *plow*. The machine is then in the Standby 2 state, in which it is ready to produce beam. Alternatively, the transition fails if any of the parameters times out or reports other errors.

It is desirable that as many supplies as possible ramp up at the same time, so that the operation may be completed as quickly as possible. In other words, the Standby 1 to Standby 2 operation proceeds in parallel. However, a few of the parallel threads include sequential constraints. For example, the high-voltage deflector supply must not be turned on until the main magnet current exceeds a certain value; otherwise, in the absence of the magnetic field, high voltage sparks can occur that may damage the equipment.

The Z schema composition operator $\fatsemi$ does provide a built-in way to build up sequential operations from schemas. For our application we would like to define operations that mostly consist of multiple instances of (something like) our *TurnOnPS* schema running in series and in parallel, with each instance instantiated with different values for its variables. This is another area where other notations may be more suitable than Z.

5.9 Event-driven Operations

Every operation described so far is invoked by the cyclotron operator. The specification must also describe actions that are triggered by events internal to the machinery. These events may occur asynchronously with respect to the operator's actions. For example, there are many hardware and software interlocks that must turn off the beam rapidly if faults occur.

The beam is actually created by turning on radio frequency (RF) power amplifiers that pump ions around the cyclotron. The computer only allows the RF drive to be turned on if the system has reached Standby 2. The RF drive signal from the computer reaches the RF power amplifiers through a hardwired, non-programmable interlock chain. This is equivalent to a series of switches in series; each switch is opened by a different fault condition; if any switch is open, the drive signal is interrupted and the amplifiers cannot turn on. The condition of each switch is visible to the control computer, but cannot be changed by it:

$$STANDBY\_STATE = \{SB1, SB2, \ldots\}$$

$$INTLK = \{Vaultdoor, TreatKey, \ldots\}$$

$$OFF\_ON = \{OFF, ON\}$$

CycloState

$$
\begin{array}{l}
CycloParams \\
Standby : STANDBY\_STATE \\
Interlocks : \mathbb{P}\, INTLK \\
RF : OFF\_ON \\
\hline
RF = ON \Rightarrow \\
\quad Standby = SB2 \land Interlocks = \varnothing
\end{array}
$$

There is an operation to turn on the beam:

TurnOnBeam

$$
\begin{array}{l}
\Delta\, CycloState \\
\hline
RFDrive' = ON
\end{array}
$$

These two schemas convey that the beam can only be turned on if all the interlocks are clear and the system is in Standby 2. They even convey that the beam will turn off if any interlock is set or any parameter exceeds its windows. However, we need to say more: when any interlock is set, or any parameter exceeds its windows, that is an *event* which is logged in an error file and which also causes the RF drive to turn off:

OutsideWindows

$$
\begin{array}{l}
\Delta\, CycloState \\
error! : MESSAGE; \\
\hline
RFDrive = ON \\
\exists\, p : PNAME \bullet \\
\quad ((pread'\ p < plow\ p) \lor \\
\quad (pread'\ p > phigh\ p)) \\
RFDrive' = OFF \\
error! = \text{"}\texttt{Parameter } p \texttt{ outside window}\text{"}
\end{array}
$$

The first line in the predicate indicate that an error is logged only if the RF drive is on when the window is exceeded. The second line describes the condition that triggers the operation and the last two lines describe the consequences of this condition. However, this schema doesn't really convey that it is the *event* of the *pread* crossing the window limit that triggers this operation, and that the state of *RFDrive* has to be changed as soon as possible after this happens.

5.10 Concurrency

The Z and VDM notations do not provide any way to express concurrency; however, they do not *preclude* describing concurrent activities. It can be understood that different operations which do not depend on the same components of the system state may proceed at the the same time.

Difficulties arise only when concurrent operations must interact (because they use the same components of the system state). This is a serious practical limitation. Some of the operations described by *ChangeParam* can take several minutes. It is sometimes necessary for the operator to cancel or modify these operations while they are in progress. For example, the operation *SetParam* may occur while *ChangeParam* is in progress, and the new value of *psetting* determined by *SetParam* must immediately supercede the final value being approached by *ChangeParam*. There is no way to express this in Z or VDM.

6 Alternatives to Z and VDM

We encounter difficulties using Z notation to represent interdependent collections of serial and parallel operations, event-driven operations, and concurrency. Other formal notations are designed to handle those.

6.1 Petri Nets

Petri nets are well-suited to concurrent activities. They have an intuitively appealing graphic representation, which we have found useful in discussing requirements with physicists and engineers. A Petri net representation of the transition from Standby 1 to Standby 2 appears in Fig. 1. Conditions are represented by circles called *places* and operations are shown as horizontal lines called *transitions*. A Petri net is executed by placing dots called *tokens* in the place or places representing the initial conditions. Any transition whose antecedent places are all marked with tokens may *fire*; one token is removed from each antecedent place and one token is put on every successor place. Each combination of tokens and places is called a *marking* and represents a possible state of the system.

In Fig. 1 the marking in which the entire upper row of places contains tokens represents Standby 1, and the entire lower row represents Standby 2. The figure clearly shows that most transitions may occur in parallel but the main field current must precede the

Figure 1: Petri net representation of Standby 1 to Standby 2 transition

deflector voltage (note interlock I between $MAINFLD$ and $DFLVOLT$). The figure is incomplete because it only shows successful transitions. Petri nets can also represent failures, including timeouts, and have been applied in safety analyses [15].

6.2 SCR and RTL

The notation developed by the Naval Research Laboratory's Software Cost Reduction project (SCR) is well-suited to event-driven operations [8]. To express the event that occurs when a particular parameter passes outside its window limits while the beam is on, one writes:

```
  @T(!DFLVOLT outside window!)
WHEN (*BEAM ON*)
```

Where `!` and `*` are delimiters indicating conditions and modes defined formally elswhere. Events defined in this way can be used as triggers for event-driven *demand functions*, which in this case would include `Write Error Log` and `Turn Off Beam`.

Specifications expressed in an SCR-like notation can be transformed into a formal logic called Real Time Logic (RTL) [10]. This enables formal timing and safety analyses to be performed.

7 Conclusions

The sample specifications presented here address a significant portion of the cyclotron control programs. We conclude that writing comprehensive formal specifications for our application is feasible.

However, unsolved problems remain. This study indicates that several formal specification notations appear promising for our application, but none will completely suffice. Different notations each deal with different parts of the problem and there is not much guidance about how to integrate them. Z deals with input/output relations but not concurrency and timing; Petri nets deal with concurrency but not timing constraints; RTL deals with timing constraints but is weak on input/output relations, and so on. An obvious approach is to use two or more notations, but the semantics of such combinations is not clear.

In addition, the large size of our system confronts us with problems of style and organization whose solutions are not apparent from the small case studies found in the literature.

7.1 The Z notation

We anticipate using the Z specification notation to express most requirements that do not include timing constraints nor interaction of concurrent processes.

Much of the effort in specifying large applications like ours is devoted to enumerating the objects and values that comprise the system, and listing the operations that must be provided, taking care that nothing is omitted and no inconsistencies in naming or usage are introduced. Z provides a well-defined discipline for doing this, that is explained in good textbooks and supported by document preparation tools. Even when used in such an unambitious way, Z promises to be a useful supplement to the usual specifications in English prose plus tables and diagrams.

The Z specifications also suggest (though they need not dictate) an implementation strategy. The data schemas *CycloParams*, *StatusDisplay*, and *CycloState* can be seen as models of regions of computer memory, or of data structures in the implementation language. The various operation schemas can be seen as specifications for procedures that operate on one or another of these data structures. The schema signatures can be readily translated to prodedure declarations in any implementation language, and the predicates can serve as guides for programming and test case selection even if they are not used to perform a full-blown formal verification.

We anticipate considerable effort developing a notational style in Z for presenting compact, easily understood specifications for complex process control systems. Although our system is large, we believe its specification may be concise because much of its size derives from repetition of similar elements. Almost half the total signals are contributed by about forty high-current power supplies. These are not identical but share many features in common. The Z schema calculus should permit common features to be described with texts that apply to all, supplemented with brief texts that address the differences.

7.2 Other notations

In addition to Z, we will require a different notation or notations to express timing requirements and interaction of concurrent processes. Other notations may also offer advantages of style or clarity. Candidates include (but are not limited to) Petri nets, SCR notation and RTL.

7.3 Future work

The work presented here represents the simplest part of the system. Treatment operations, and the interaction between the treatment operations and the cyclotron controls, will be more difficult. There are much graver safety implications, and many potential faults, where safe and graceful recovery must be provided. We anticipate that these developments will provide more opportunities to investigate formal methods in a large and important application.

8 Acknowledgements

The author thanks Stan Brossard, Jüri Eenmaa, Lee Hutter, Ira Kalet, Peter Wootton, and especially Ruedi Risler for sharing their knowledge of cyclotrons.

References

[1] H. Bassen, J. Silberberg, F. Houston, W. Knight, C. Christman, and M. Greberman. Computerized medical devices: Usage trends, problems, and safety technology. In James C. Lin and Barry N. Feinberg, editors, *Frontiers of Engineering and Computing in Health Care: Proceedings of the Seventh Annual Conference of the IEEE/Engineering in Medicine and Biology Society*, pages 180–185, 1985.

[2] R. E. Bloomfield and P. K. D. Froome. The application of formal methods to high integrity software. *IEEE Transactions on Software Engineering*, SE-12(9):988–993, 1986.

[3] Jonathan Bowen. Specification of timing constraints. *Z Forum*, 1(6), 1986. June 3.

[4] Anders Brahme. Optimization of conformation and general moving beam radiation therapy techniques. In I. A. D. Bruinvis, P. H. van der Giessen, H. J. van Kleffens, and F. W. Wittkämper, editors, *Proceedings of the Ninth International Conference on Computers in Radiation Therapy*, pages 227–230, Amsterdam, 1987. North-Holland.

[5] B. Cohen, W. T. Harwood, and M. I. Jackson. *The Specificiation of Complex Systems.* Addison Wesley, 1986.

[6] Norman Delisle and David Garlan. Formally specifying electronic instruments. In *Proceedings of the Fifth International Workshop on Software Specification and Design*, Washington, DC, 1989. IEEE Computer Society Press. (also ACM Software Engineering Notes 14(3), May 1989).

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

[8] 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.

[9] J. Jacky, R. Risler, S. Brossard, and I. Kalet. Relational database: a radiation therapy machine control software development tool. In Yongmin Kim and Francis A. Spelman, editors, *Images of the Twenty-First Century: Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Volume 11*, pages Part 6, 1918–1919, 1989.

[10] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, SE-12(9):890–904, 1986.

[11] Cliff B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[12] E. J. Joyce. Malfunction 54: Unraveling deadly medical mystery of computerized accelerator gone awry. *American Medical News*, 1986. Oct. 3, page 1.

[13] Ira J. Kalet. Radiotherapy treatment planning tools technical progress report 1a. Technical report, Radiation Oncology Department, University of Washington, Seattle, Washington, 1989.

[14] C. J. Karzmark and Neil C. Pering. Electron linear accelerators for radiation therapy: History, principles and contemporary developments. *Physics in Medicine and Biology*, 18(3):321–354, May 1973.

[15] Nancy G. Leveson and Janice L. Stolzy. Safety analysis using Petri nets. *IEEE Transactions on Software Engineering*, SE-13(3):386–397, 1987.

[16] Eric R. Lindstrom and Louise S. Naylor, editors. *Proceedings of the 1987 IEEE Particle Accelerator Conference: Accelerator Engineering and Technology.* IEEE, 1987.

[17] M. Loyd, H. Chow, J. Laxton, I. Rosen, and R. Lane. Dose delivery error detection by a computer-controlled linear accelerator. *Medical Physics*, 16(1):137–139, 1989.

[18] 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.

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

[20] Martin S. Weinhous, James A. Purdy, and Conrad O. Granda. Testing of a medical linear accelerator's computer-control system. *Medical Physics*, 17(1):95–102, Jan/Feb 1990.