Formal Specification of Control Software for a Radiation Therapy Machine (Revised)

Jonathan Jacky *
Michael Patrick
Jonathan Unger

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

Technical Report 95-12-01

January 9, 1997

Abstract

This report presents a formal (mathematical) specification for the operator's console of a computer-controlled radiation therapy machine equipped with a multileaf collimator. This formal specification, rather than the prose specification, serves as the primary reference source for programming and test planning.

Specified functions include selecting treatment setups from a database of stored prescriptions, setting up prescriptions on the treatment machine manually or semi-automatically, checking that the setup conforms to the prescription (with provision for overriding certain settings, with operator confirmation), safety interlocking and essential user interface features. The specification supports physics and experimental procedures as well as normal patient treatments.

The specification is expressed in the Z notation. It formalizes the requirements in a thorough informal (English prose) specification. Its organization suggests a detailed design.

*email `jon@radonc.washington.edu`, telephone (206)-548-4117, fax (206)-548-6218

1 Introduction

This report presents portions of a formal specification for a real medical device, a radiation therapy machine[1]. This specification, rather than the informal prose description, serves as the primary reference source for programming and test plannning. A paper [4] describes the development of part of the program based on the contents of this report.

This formal specification is based on a thorough informal (English prose) specification presented as Chapters 2 and 8 in [2]. Here we attempt to formalize the requirements in that source. We have included many cross-references. Decimal numbers and integers (as in 8.4, 191) refer to chapter, section and page numbers in [2], respectively.

The formal specification is expressed in the Z notation [7]. We have corrected syntax and type errors detected by a checker [6].

2 Overview

Much of the apparent complexity in the prose requirements arises from the interaction of several subsystems which, by themselves, are simpler. In the formal specification we partition the system into subsystems and describe simple operations on each. For each operation on the system as a whole, we define a separate operation on each affected subsystem. The complex behaviors of the whole system emerge when we compose these simpler operations together.

Each subsystem is modelled by a Z state schema and a number of operation schemas on that state. This partition can itself be represented in Z.

$$
\begin{array}{|l}
\hline
\textit{TherapyControl} \\
\hline
\textit{Session} \\
\textit{Field} \\
\textit{Intlk} \\
\ldots \\
\textit{Console} \\
\hline
\end{array}
$$

Session (section 5) models those aspects of the treatment session that are related to the prescription database (section 4 models the database itself). *Field* (section 6) models the many settings that characterize a single field. *Intlk* (section 7) models software interlocks

[1]We plan to include additional portions in future versions of this report. The present version supercedes an earlier report [3] and several earlier versions of this report.

and other flags that indicate readiness. *Console* (section 8) models the user interface. Section 9 combines operations from *Session*, *Field* and *Console*.

Related operations in different subsystems are distinguished by suffix: *ExptModeS*, *ExptModeF*, and *ExptModeC* are operations in the *Session*, *Field* and *Console* subsystems, respectively.

Each user interface operation in the *Console* subsystem ensures that corresponding operations in the *Session* and *Field* subsystems are only invoked when their preconditions are satisfied. Therefore only the *Console* operations need to be total; usually there is no need to define total operations in the *Session* and *Field* subsystems. For example the *ExptModeC* operation in the *Console* subsystem checks the precondition that only physicists can invoke this operation; in *Console* we define what happens when an operator who is not a physicist attempts to enter experiment mode. Therefore *ExptModeS* and *ExptModeF* can assume that this precondition has been satisfied and need not cover the other cases.

3 System configuration

Fixed aspects of the system configuration are represented by Z global constants: sets, functions and relations. This section introduces some of these constants. It should be possible to accommodate some configuration changes simply by changing their values. All the basic types and global constants defined in this report are collected in Appendix C.

3.1 Settings and registers

The state of the therapy machine is largely determined by the values of named *items*. A glossary of items appears in Appendix A. At this writing the list of items is

$$\begin{aligned} ITEM ::= {} & nfrac \mid dose\_tot \mid dose \mid wedge \mid w\_rot \mid filter \mid leaf0 \mid leaf39 \mid \\ & gantry \mid collim \mid turnt \mid lat \mid longit \mid height \mid doseB \mid top \mid \\ & pt\_mode \mid pt\_factor \mid press \mid temp \mid d\_rate \mid t\_fac \mid \\ & calvolt1 \mid calvolt2 \mid p\_dose \mid p\_time \mid e\_time \end{aligned}$$

For brevity we omit formal declarations of the other collimator *leaves*, $leaf1 \mathrel{..} leaf38$.

There are many items but we can identify different subsets, where all of the members of each subset are treated the same way for some particular purpose. A glossary of item groups, and tables showing the group membership of each item, appear in Appendix B.

Settings are items which are included in field prescriptions. Other items are kept in *registers*.

In particular *dose_reg* items include calibration factors and other items concerned with the dosimetry system[2].

$$| \; setting, dose\_reg : \mathbb{P}\, ITEM$$

At this writing

$$\langle setting, dose\_reg \rangle \ \mathsf{partition}\ ITEM$$

$$dose\_reg = \{pt\_mode, pt\_factor, press, temp, d\_rate, t\_fac,$$
$$calvolt1, calvolt2, p\_dose, p\_time, e\_time\}$$

Scales are items that are continously variable over some range; examples are gantry angle and every collimator leaf position. *Selections* can only take on certain discrete values; examples are wedge and flattening filter selection. *Counters* accumulate during treatment runs; examples are dose and the number of fractions.

$$| \; scale, selection, counter : \mathbb{P}\, ITEM$$

At this writing

$$\langle selection, scale, counter \rangle \ \mathsf{partition}\ ITEM$$

$$counter = \{nfrac, dose\_tot, dose\}$$
$$selection = \{wedge, w\_rot, filter, pt\_mode\}$$

A field is prescribed by determining the values of certain of its settings. Therapy fields are defined by the values of particular settings called *prescriptions*. Experiment fields are defined by the values of settings called *presets* (8.2, 171 third bullet; Table 8.2, 173). Readiness is determined by checking all the *preset* settings in experiment mode, and all the *prescr* settings (*prescrip* except the linear table motions) in therapy mode (8.9.8, 194). Most settings are machine *motions*, and the actual values of most settings are measured by *sensors*. The *calibration constants* are registers that are initially loaded with constants stored in the calibration database. At this writing

$$leaves == \{leaf0, leaf39\}$$

$$preset == leaves \cup \{wedge, w\_rot, filter\}$$

$$motion == preset \cup \{gantry, collim, turnt, lat, longit, height\}$$

$$prescrip == motion \cup counter$$

$$prescr == prescrip \setminus \{lat, longit, height\}$$

$$sensor == setting \setminus \{nfrac, dose\_tot\}$$

$$cal\_const == \{d\_rate, t\_fac, calvolt1, calvolt2\}$$

[2]In the C implementation *setting* and *dose_reg* are two different enumerations, separated so we can efficiently store and index zero-based C arrays. We may add other register enumerations in the future, for example for the LCC calibration factors.

3.2 Values

The value of every item can be represented by a number.

$$VALUE == \mathbb{Z}$$

Each item will be implemented by an appropriate (possibly floating point) numeric type. In this report it is sufficient to say they are all numbers, to indicate that we can do arithmetic with them.

Each item can assume a particular range of *valid* (physically achievable) values. For example, the gantry angle can vary from 0 to 359[3]; the available wedge selections are *no_wedge*, 30, 45 and 60. We use *valid* to do range checking on numbers that the operator types in, and also on sensor readings, to check for faults[4]. Every setting s has some valid values, and there is always a minimum and a maximum valid value. We define an uninitialized or *blank* value which is not valid for any setting. For each *scale* item, there is a *tolerance* within which variations in value are acceptable.

$$\begin{array}{|l}
blank : VALUE \\
tol : scale \longrightarrow VALUE \\
valid : ITEM \longrightarrow \mathbb{F}_1\ VALUE \\
\hline
\forall\, s : ITEM \bullet blank \notin valid\ s
\end{array}$$

4 Prescription database

The prescription database stores patients and fields. We define a basic type for the names that identify them.

$$[NAME]$$

$$PATIENT == NAME;\ FIELD == NAME$$

An item's name usually corresponds to the text string that identifies it in screen displays and log files[5].

We distinguish a special value to indicate that no name has been selected.

[3]In the implementation gantry angle varies from 0.0 to 359.9. Decimal fractions are not built into Z.

[4]The C implementation includes one *valid* array indexed by *setting* and another (with a different name) indexed by *dose_reg*.

[5]We define one type for both kinds of names so the same specifications (and code) can be used to handle lists of patients and fields. In the implementation, elements of *NAME* are integer indices into arrays, usually of C structures that include the name string as one member.

$$
\begin{array}{|l}
no\_name : NAME
\end{array}
$$

$$no\_patient == no\_name;\ no\_field == no\_name$$

In experiment mode, we store fields under *studies* which are analogous to patients. In our model they have the same type.

$$
\begin{array}{|l}
studies, patients : \mathbb{P}\, PATIENT \\
\hline
no\_patient \notin studies \land no\_patient \notin patients
\end{array}
$$

For each patient or study, several prescribed fields are stored[6]. We must check against delivering too many fractions or monitor units from the same field (8.9.4, 187 – 188), so the accumulated values of the counters are also stored (for patient fields only).

$$ACCUMULATION == counter \longrightarrow VALUE$$

$$PRESCRIPTION == prescrip \longrightarrow VALUE$$

$$
\begin{array}{|l}
Preset : studies \longrightarrow (FIELD \nrightarrow PRESCRIPTION) \\
Prescribed : patients \longrightarrow (FIELD \nrightarrow PRESCRIPTION) \\
Accumulated : patients \longrightarrow (FIELD \nrightarrow ACCUMULATION) \\
\hline
\forall\, s : studies \bullet no\_field \notin \mathrm{dom}(Preset\ s) \\
\forall\, p : patients \bullet no\_field \notin \mathrm{dom}(Prescribed\ p) \land \mathrm{dom}\,(Prescribed\ p) = \mathrm{dom}\,(Accumulated\ p)
\end{array}
$$

The *exceeded* predicate tests whether the prescribed fractional dose, total dose or number of fractions have already been delivered.

$$
\begin{array}{|l}
exceeded\_ : ACCUMULATION \leftrightarrow PRESCRIPTION \\
\hline
\forall\, counters : ACCUMULATION;\ fields : PRESCRIPTION \bullet \\
\quad exceeded(counters, fields) \Leftrightarrow (\exists\, c : counter \bullet counters\ c \geq fields\ c)
\end{array}
$$

In the following discussion consider field f of patient p; let $prescribed = Prescribed\,p\,f$ and $accumulated = Accumulated\,p\,f$. The prescription includes the number of fractions $prescribed\ n$ and the total dose $prescribed\ dose\_tot$. We also keep track of the number of fractions accumulated to date $accumulated\ n$, the number of monitor units delivered since the beginning of the day $accumulated\ dose$ and the total number of monitor units accumulated to date $accumulated\ dose\_tot$. Table 1 shows the settings and values pictured on each line of the field selection display (8.9.4, Fig. 83, 186).

[6]This differs from [1], which describes a single collection of experiment fields. Moreoever, for each experiment field we now store the same *prescrip* settings as for therapy fields, although we only check the *preset* settings for agreement with the stored prescription

Field	*field*
Fractions	*prescribed n*
To date	*accumulated n*
MU	*prescribed dose*
Total	*prescribed dose_tot*
Expected	*accumulated n ∗ prescribed dose*
To date	*accumulated dose_tot*

Table 1: Settings and values in the field list display

4.1 Operators

Our *OPERATOR* type includes the operator's username and password. A special value indicates no operator has logged in. Physicists are operators who are authorized to use the equipment in its experiment mode (8.2, 170).

$$[OPERATOR]$$

$$\begin{array}{|l} no\_operator : OPERATOR \\ operators, physicists : \mathbb{P}\ OPERATOR \\ \hline physicists \subseteq operators \end{array}$$

5 Session

In this section we model those aspects of the treatment session that are related to the prescription database. In section 9, we will combine the operations defined here with user interface operations described in section 8.

5.1 *Session* state

The *Session* state is determined by the treatment *mode*, the *operator* on duty, the currently selected *patient* and *field*, the accessible *names* (patients or studies), and the accessible prescribed *fields* and their *counters*. We first define *SessionVars* which declares all the state variables and provides predicates to ensure that the operator is authorized for the mode, and the names are consistent with the mode.

$$MODE ::= therapy \mid experiment$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{SessionVars}} \\
mode : MODE \\
operator : OPERATOR \\
patient : PATIENT \\
field : FIELD \\
names : \mathbb{P}\, PATIENT \\
fields : FIELD \nrightarrow PRESCRIPTION \\
counters : FIELD \nrightarrow ACCUMULATION \\
\hline
operator = no\_operator \lor operator \in operators \\
mode = experiment \Rightarrow operator \in physicists \\
names = \mathbf{if}\ mode = therapy\ \mathbf{then}\ patients\ \mathbf{else}\ studies \\
\hline
\end{array}
$$

Next, we define two cases. When no patient is selected, no prescribed fields are accessible; no field can be selected.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{NoPatient}} \\
SessionVars \\
\hline
patient = no\_patient \\
field = no\_field \\
fields = \varnothing \\
counters = \varnothing \\
\hline
\end{array}
$$

When a patient is selected, that patient's fields are accessible. If a field is selected, it must be one of these.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{PrescribedPatient}} \\
SessionVars \\
\hline
patient \neq no\_patient \\
patient \in names \\
field = no\_field \lor field \in \operatorname{dom} fields \\
fields = \mathbf{if}\ mode = therapy\ \mathbf{then}\ Prescribed\ patient\ \mathbf{else}\ Preset\ patient \\
mode = therapy \Rightarrow counters = Accumulated\ patient \\
\hline
\end{array}
$$

Together these define the *Session* state.

$$Session \mathrel{\widehat{=}} PrescribedPatient \lor NoPatient$$

The *Session* subsystem starts up in therapy mode with no operator and no patient.

$$
\begin{array}{|l}
\hline
\textit{InitSession} \\
\textit{NoPatient} \\
\hline
\textit{mode} = \textit{therapy} \\
\textit{operator} = \textit{no\_operator} \\
\hline
\end{array}
$$

None of the *Session* state variables are sensor inputs; all are under program control.

5.2 Operations on *Session*

In the following subsections we model the operations on *Session*. We will put together the operations defined in different states in section 9, below.

5.2.1 Experiment mode

Physicists can toggle the session from therapy mode to experiment mode and back[7]. The user interface ensures that only physicists can invoke this operation, so there is no need here to define a total operation that describes what happens when an operator who is not a physicist attempts this operation. After switching modes, no patient (study) and no field are selected (8.9.6, 190 – 191).

$$
\begin{array}{|l}
\hline
\textit{ExptModeS} \\
\Delta \textit{Session} \\
\hline
\textit{operator} \in \textit{physicists} \\
\textit{NoPatient}' \\
(\textit{mode}', \textit{names}') = \mathbf{if}\ \textit{mode} = \textit{therapy}\ \mathbf{then}\ (\textit{experiment}, \textit{studies}) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ (\textit{therapy}, \textit{patients}) \\
\textit{operator}' = \textit{operator} \\
\hline
\end{array}
$$

5.2.2 Store Field

Store Field (8.9.5, 189 – 188) accepts a new field name, which becomes the selected field and is also added to the list of fields[8].

[7] This is a change from the original requirements in [1], where **Experiment Mode** switches to experiment mode but **Select Patient** switches back to therapy mode.

[8] The prose [2] also requires that the new field be added to the prescription database for the current patient. We do not model this formally (in fact we model the prescription database as a constant). The precondition $patient \neq no\_patient$ is not explicit in the prose.

StoreFieldS

$$
\begin{array}{|l}
\Delta Session \\
field? : FIELD \\
prescribed' : PRESCRIPTION \\
accumulated' : ACCUMULATION \\
\hline
patient \neq no\_patient \\
field' = field? \\
fields' = fields \cup \{ field' \mapsto prescribed' \} \\
mode = therapy \Rightarrow counters' = counters \cup \{ field' \mapsto accumulated' \} \\
\\
mode' = mode \\
operator' = operator \\
patient' = patient \\
names' = names \\
\hline
\end{array}
$$

Here *prescribed′* and *accumulated′* are just place holders; their values are defined in the corresponding *Field* operation *StoreFieldF*.

5.2.3 Login

Login (2.5.2, 17 – 20; 8.9.1, 183) accepts a new *operator*?. The user interface ensures that the new operator is authorized.

NewOperator

$$
\begin{array}{|l}
\Delta Session \\
operator? : OPERATOR \\
\hline
operator' = operator? \\
operator' \in operators \\
\hline
\end{array}
$$

There are two variations. Usually the new operator is sufficiently privileged to keep the same mode. Otherwise the session reverts to therapy mode with no patient and no field (8.9.6, 190).

$$
\begin{array}{|l}
\hline
\textit{Privileged} \\
NewOperator \\
\hline
mode = therapy \lor operator' \in physicists \\
\\
mode' = mode \\
patient' = patient \\
names' = names \\
field' = field \\
fields' = fields \\
counters' = counters \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Unprivileged} \\
NewOperator \\
\hline
mode = experiment \\
operator \notin physicists \\
mode' = therapy \\
NoPatient' \\
\hline
\end{array}
$$

$$LoginS \mathrel{\widehat=} Privileged \lor Unprivileged$$

5.2.4 Select Patient

In **Select Patient** (8.9.3, 184 – 185) the patient's prescribed fields are loaded, but no field is selected [9]. The user interface ensures that the new patient is in the prescription database.

$$
\begin{array}{|l}
\hline
\textit{SelectPatientS} \\
\Delta Session \\
patient? : PATIENT \\
\hline
patient? \in names \\
patient' = patient? \\
field' = no\_field \\
fields' = \mathbf{if}\ mode = therapy\ \mathbf{then}\ Prescribed\ patient'\ \mathbf{else}\ Preset\ patient' \\
mode = therapy \Rightarrow counters' = Accumulated\ patient' \\
\\
mode' = mode \\
operator' = operator \\
names' = names \\
\hline
\end{array}
$$

[9] The prose [1] says that if the patient list is selected in experiment mode, the session reverts to therapy mode (8.9.3, 184 last paragraph). We have dropped this requirement.

5.2.5 Select Field

Select Field (8.9.4, 186 – 189) changes the current field. The user interface ensures this operation cannot occur if there is no patient, and ensures that the new field is prescribed.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SelectFieldS}} \\
\hline
\Delta \textit{Session} \\
\textit{field}? : \textit{FIELD} \\
\hline
\textit{patient} \neq \textit{no\_patient} \\
\textit{field}? \in \operatorname{dom} \textit{fields} \\
\textit{field}' = \textit{field}? \\
\\
\textit{operator}' = \textit{operator} \\
\textit{mode}' = \textit{mode} \\
\textit{patient}' = \textit{patient} \\
\textit{fields}' = \textit{fields} \\
\textit{counters}' = \textit{counters} \\
\hline
\end{array}
$$

6 Field

In this section, we look inside the machine state and deal with particular machine settings. We model operations that involve the many settings that characterize a single field.

6.1 *Field* state

The *Field* schema includes the state variables that represent settings for the currently selected *field* and *mode*. *Sensors* report *measured* setting values. *Prescribed* setting values are read from the prescription database.

Computed and *calibrated* item values are entered by the operator or calculated from prescribed settings and calibration constants; these are stored in *registers*. Certain *calibration constants* are stored in files (8.9.13, 213 first full paragraph; 215 last paragraph). *Counters* hold setting values that are *accumulated* over successive runs. For example, the *dose* prescribed for a single fraction may be have to be delivered in two or more treatment runs.

Some settings that do not match their prescribed values can be *overridden* by the operator (8.4, 175 second paragraph; 8.8.1, 181). It is necessary to store the value of each setting when it is overridden (see the requirement in the last paragraph under “override” on p. 181). Only settings that are prescribed can be overridden.

$$cal\_factor : cal\_const \longrightarrow VALUE$$

$$
\begin{array}{|l}
\hline
\textit{Field} \\
prescribed : PRESCRIPTION \\
accumulated : ACCUMULATION \\
measured : sensor \longrightarrow VALUE \\
overridden : prescr \mapsto VALUE \\
computed, calibrated : dose\_reg \longrightarrow VALUE \\
\hline
\end{array}
$$

The *measured* settings are read from sensors so here we cannot write any predicates that constrain them.

6.2 Relation to *Session* state

A few operations on *Field* read the *mode* and *field* state variables declared in *Session*. In therapy mode, the *prescribed* settings in the *Field* state are those from the prescription database entry for the currently selected *mode* and *field* in the *Session* state. (In experiment mode the prescribed settings are also loaded from the prescription database but may be changed subsequently. In therapy mode the counters are loaded from the prescription database when the field is selected but may be changed subsequently. See section 6.4.2).

$$
\begin{array}{|l}
\hline
\textit{PrescribedField} \\
Field \\
Session \\
\hline
field \neq no\_field \\
mode = therapy \Rightarrow prescribed = fields\ field \\
\hline
\end{array}
$$

When no field has been selected, prescribed settings and counters have no values and the computed settings dose and time indicate no dose. (8.9.7, 192, second paragraph after the bullets). No settings are overridden (8.9.8, 194; 8.9.9, 196; 8.9.10, 198).

$$no\_prescrip == (\lambda\, p : prescrip \bullet blank)$$

$$no\_counter == (\lambda\, c : counter \bullet blank)$$

$$no\_dose\_reg == (\lambda\, d : dose\_reg \bullet blank)$$

$$no\_dose == \{p\_dose \mapsto blank, p\_time \mapsto blank\}$$

$$
\begin{array}{|l}
\hline
\textit{NoFieldF} \\
\textit{Field} \\
\hline
\textit{prescribed} = \textit{no\_prescrip} \\
\textit{accumulated} = \textit{no\_counter} \\
\textit{no\_dose} \subseteq \textit{computed} \\
\textit{overridden} = \emptyset \\
\hline
\end{array}
$$

$$\textit{NoFieldS} \mathrel{\widehat=} [\textit{Session} \mid \textit{field} = \textit{no\_field}]$$

$$\textit{NoField} \mathrel{\widehat=} \textit{NoFieldF} \land \textit{NoFieldS}$$

FieldSession expresses the combined invariant:

$$\textit{FieldSession} \mathrel{\widehat=} \textit{PrescribedField} \lor \textit{NoField}$$

6.3 Initialization

Field begins with no field. The calibration factors are initialized with the constants on file (8.9.13, 213, second paragraph after bullets) and the other registers hold no values.

$$
\begin{array}{|l}
\hline
\textit{InitField} \\
\textit{NoFieldF} \\
\hline
\textit{computed} = \textit{calibrated} = \textit{no\_dose\_reg} \oplus \textit{cal\_factor} \\
\hline
\end{array}
$$

6.4 Operations on *Field*

In the following subsections we model the operations on *Field*. We will put together the operations defined in different states in section 9, below.

6.4.1 Select Patient

SelectPatient also affects *Field*: when a patient is first selected, there is no field.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SelectPatientF}} \\
\hline
\Delta \mathit{Field} \\
\hline
\mathit{NoFieldF}' \\
\mathit{computed}' = \mathit{computed} \oplus \mathit{no\_dose} \\
\mathit{calibrated}' = \mathit{calibrated} \\
\hline
\end{array}
$$

6.4.2 Select Field

When a new field is selected, its prescribed settings are loaded and no settings are overridden. This operation requires read-only access to the *fields* state variable in the *Session* schema.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NewFieldF}} \\
\hline
\Delta \mathit{FieldSession} \\
\hline
\mathit{prescribed}' = \mathit{fields}\ \mathit{field}' \\
\mathit{overridden}' = \varnothing \\
\hline
\end{array}
$$

There are two variants of *SelectField*. Experiment mode is much simpler because there is no prescribed dose. The prescribed settings are loaded. The dose and time do not change (8.9.11, 202, second paragraph from bottom).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SelectExptFieldF}} \\
\hline
\mathit{NewFieldF} \\
\hline
\mathit{mode} = \mathit{experiment} \\
\mathit{computed}' = \mathit{computed} \\
\mathit{calibrated}' = \mathit{calibrated} \\
\hline
\end{array}
$$

Selecting rectangular fields in experimental mode (8.9.4, 188 – 189) is not modelled formally.

In therapy mode, the dose for the treatment run and the treatment backup time are calculated. Treatment backup time is calculated from the dose and two calibration factors, the machine's nominal dose rate *computed d_rate* and the treatment time factor *computed t_fac* (8.9.11, 200, last paragraph; 202, second paragraph; 8.9.13, 213, first two paragraphs after bullets)[10].

[10] The backup time is given by $t\_backup = factor * dose/rate$. For example with prescribed dose 100.0 MU, dose rate 50.0 MU/min and factor 1.50 the backup time is 3.00 minutes. We do not attempt to model this floating-point calculation in Z.

SETTING	PRESCR	PRESET	ACCUM
DOSE A	*prescribed dose*	*computed p_dose*	*measured dose*
DOSE B	*prescribed dose*	*computed p_dose*	*measured doseB*
TIME	*calibrated p_time*	*computed p_time*	*calibrated e_time*

Table 2: Settings and values in the dosimetry display

$$DOSE == VALUE; RATE == VALUE; FACTOR == VALUE; TIME == VALUE$$

$$
\begin{array}{l}
t\_backup : (DOSE \times RATE \times FACTOR) \nrightarrow TIME \\
\hline
\forall d : valid\ dose; r : valid\ d\_rate; f : valid\ t\_fac \bullet \\
\quad (d, r, f) \in \operatorname{dom} t\_backup \land t\_backup(d, r, f) \in valid\ p\_time
\end{array}
$$

We keep track of the number of monitor units delivered since the beginning of the day *accumulated dose*. When the prescribed field settings are loaded, the computed dose is adjusted to deliver the remaining daily dose. This makes it easy to set up another treatment run for the same field if the earlier attempts had to be interrupted for any reason, or were used to make a port film. The treatment backup time is calculated from this adjusted dose, not the prescribed dose.

The adjusted dose and corresponding backup time are stored in *computed p_dose* and *calibrated p_time* (*computed p_dose* may differ from *prescribed dose*). There is also a register *computed p_time* where the user may optionally enter a backup time different than *calibrated p_time* (section 6.4.3, below). Table 2 shows the settings and values pictured on the dosimetry display (8.9.11, Fig. 8.8, 199; Fig. 8.9, 203; Fig 8.10, 207; Fig. 8.11, 208).

$$
\begin{array}{l}
\textit{DoseTime} \\
\hline
\Delta Field \\
\hline
(\mathbf{let}\ t == t\_backup(computed'\ p\_dose, computed'\ d\_rate, computed'\ t\_fac) \bullet \\
\quad calibrated' = calibrated \oplus \{ p\_time \mapsto t \}) \\
computed'\ p\_time = calibrated'\ p\_time \\
\{p\_dose, p\_time\} \mathbin{\lhd} computed' = \{p\_dose, p\_time\} \mathbin{\lhd} computed \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{NewTherapyField} \\
\hline
NewFieldF \\
DoseTime \\
\hline
mode = therapy \\
accumulated' = counters\ field' \\
\hline
\end{array}
$$

There are two cases. The normal case occurs when the user interface confirms that the prescribed fractional dose, total dose and number of fractions are not yet *exceeded*. The dose is read from the prescription, and no settings are overridden (8.9.4, 187).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SelectTherapyFieldF}} \\
\hline
\textit{NewTherapyField} \\
\hline
\textit{computed}'\ \textit{p\_dose} = \textit{prescribed dose} - \textit{accumulated dose} \\
\textit{overridden}' = \varnothing \\
\hline
\end{array}
$$

Together these make the simple case

$$\textit{SelectSimpleFieldF} \mathrel{\widehat{=}} \textit{SelectExptFieldF} \lor \textit{SelectTherapyFieldF}$$

The other case occurs when the user interface acquires the preset dose from the operator (often when one or more of the counter settings is *exceeded*. If this differs from the prescribed dose then dose is overridden, and any exceeded settings are also overridden (8.9.4, 188).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SelectComplexFieldF}} \\
\hline
\textit{NewTherapyField} \\
\textit{dose}? : \textit{VALUE} \\
\hline
\textit{computed}'\ \textit{p\_dose} = \textit{dose}? \\
(\textbf{let}\ \textit{ovr} == (\lambda\, c : \textit{counter} \mid \textit{accumulated}'\ c \geq \textit{prescribed}'\ c \bullet \textit{accumulated}\ c) \bullet \\
\quad \textit{overridden}' = \textbf{if}\ \textit{dose}? = \textit{prescribed}'\ \textit{dose} \\
\qquad\qquad \textbf{then}\ \textit{ovr}\ \textbf{else}\ \textit{ovr} \cup \{\textit{dose} \mapsto \textit{dose}?\}) \\
\hline
\end{array}
$$

Here we have made a few small changes from the prose requirements. According to the prose (8.9.4, 187 – 188), the **Select Field** operation includes a dialog with the operator to enter a new dose or treatment time in some cases. In our formal specification it is necessary for the operator to explicitly select the **Edit** operation after **Select Field** in order to enter a new dose or treatment time. These minor adjustments achieve the intent of the prose and simplify the program. As required by the prose, our *SelectComplexFieldF* overrides exceeded settings (after operator confirmation, enforced by the user interface)[11].

6.4.3 Edit setting

The edit operation updates a *prescribed* or *computed* item value.

[11] We also considered the slightly simpler alternative of omitting the operator confirmation and leaving $\textit{overridden} = \varnothing$ in the *exceeded* case. In that alternative, the *Intlk* subsystem (section 7) would make the offending settings *not_ready* to prevent the field being delivered unless the operator explicitly edits or overrides those settings.

$$
\begin{array}{|l}
\hline
\textit{EditF} \\
\Delta \textit{Field} \\
\textit{item}? : \textit{ITEM} \\
\textit{value}? : \textit{VALUE} \\
\hline
\textit{accumulated}' = \textit{accumulated} \\
\textit{calibrated}' = \textit{calibrated} \\
\hline
\end{array}
$$

The prose actually describes four **Edit** operations. Some features are common to all. The first variation is for preset settings; the user interface ensures this can be invoked in experiment mode only (8.8.1, 180). The prescribed value is changed, and that setting is no longer overridden.

$$
\begin{array}{|l}
\hline
\textit{EditPresetF} \\
\textit{EditF} \\
\hline
\textit{item}? \in \textit{preset} \\
\textit{prescribed}' = \textit{prescribed} \oplus \{\textit{item}? \mapsto \textit{value}?\} \\
\textit{overridden}' = \{\textit{item}?\} \lhd \textit{overridden} \\
\textit{computed}' = \textit{computed} \\
\hline
\end{array}
$$

The second variation is for calibration factors; again, the user interface only provides this in experiment mode (8.9.3, 215). Calibration factors that users can edit are modelled as *computed* settings in *registers*. Calibration factors are never considered overridden.

$$
\begin{array}{|l}
\hline
\textit{EditCalF} \\
\textit{EditF} \\
\hline
\textit{item}? \in \textit{dose\_reg} \setminus \{\textit{p\_dose}, \textit{p\_time}\} \\
\textit{computed}' = \textit{computed} \oplus \{\textit{item}? \mapsto \textit{value}?\} \\
\textit{prescribed}' = \textit{prescribed} \\
\textit{overridden}' = \textit{overridden} \\
\hline
\end{array}
$$

The third variation is for dose (8.9.11, 201–202). The computed (not prescribed) value is changed, and the dose is considered overridden (8.9.4, 188; 8.9.11, 202). The treatment times are recalculated.

$$
\begin{array}{|l}
\hline
\textit{EditDoseF} \\
\textit{EditF} \\
\textit{DoseTime} \\
\hline
item? = p\_dose \\
computed'\ p\_dose = value? \\
overridden' = overridden \oplus \{dose \mapsto value?\} \\
prescribed' = prescribed \\
\hline
\end{array}
$$

The fourth and last variation is treatment backup time, which can be edited in both modes (8.9.11, 202). Here again the computed value is changed; time is not a prescribed setting, so it cannot be overridden.

$$
\begin{array}{|l}
\hline
\textit{EditTimeF} \\
\textit{EditF} \\
\hline
item? = p\_time \\
computed' = computed \oplus \{p\_time \mapsto value?\} \\
prescribed' = prescribed \\
overridden' = overridden \\
\hline
\end{array}
$$

Here is the combined operation:

$$EditSettingF \mathrel{\widehat{=}} EditCalF \lor EditPresetF \lor EditDoseF \lor EditTimeF$$

$EditSettingF$ is not a total operation (it does not handle all possible values of $ITEM$) but the user interface ensures that its preconditions are always satisfied.

We now provide the $EditDoseF$ and $EditTimeF$ operations instead of the dialog after **Select Field** proposed in [2] (8.9.4, 187 – 188).

6.4.4 Override

Certain items can be overridden.

$$
\begin{array}{|l}
\hline
\textit{OverF} \\
\Delta Field \\
item? : ITEM \\
\hline
prescribed' = prescribed \\
accumulated' = accumulated \\
computed' = computed \\
calibrated' = calibrated \\
\hline
\end{array}
$$

We add a newly overridden setting and its currently measured value to the *overridden* function (8.4, 175 second paragraph; 8.8.1, 181). If the setting is already overridden, the override is cancelled.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{OverrideSetting}} \\
\hline
\mathit{OverF} \\
\hline
\mathit{item?} \in \mathit{prescr} \\
\mathit{overridden}' = \\
\quad \mathbf{if}\ \mathit{item?} \notin \operatorname{dom} \mathit{overridden} \\
\qquad \mathbf{then}\ \mathit{overridden} \oplus \{\mathit{item?} \mapsto \mathit{measured}\ \mathit{item?}\} \\
\qquad \mathbf{else}\ \{\mathit{item?}\} \lhd \mathit{overridden} \\
\hline
\end{array}
$$

Dose and time are special cases; overriding either makes dose overridden with its accumulated (not measured) value as the overridden value. The counters total dose *dose_tot* and number of fractions *nfrac* can only be overridden (after operator confirmation) as part of the *SelectFieldF* operation.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{OverrideDose}} \\
\hline
\mathit{OverF} \\
\hline
\mathit{item?} \in \{\mathit{p\_dose}, \mathit{p\_time}\} \\
\mathit{overridden}' = \\
\quad \mathbf{if}\ \mathit{dose} \notin \operatorname{dom} \mathit{overridden} \\
\qquad \mathbf{then}\ \mathit{overridden} \oplus \{\mathit{dose} \mapsto \mathit{accumulated}\ \mathit{dose}\} \\
\qquad \mathbf{else}\ \{\mathit{dose}\} \lhd \mathit{overridden} \\
\hline
\end{array}
$$

$$\mathit{OverrideF} \mathrel{\widehat{=}} \mathit{OverrideSetting} \lor \mathit{OverrideDose}$$

6.4.5 Store Field

This operation (8.9.5, 189 – 190) makes the prescribed settings equal to the actual machine settings, except there is no prescribed dose and the number of fractions is set to one. The accumulators are reset to zero.

$$\mathit{zero\_counter} == (\lambda\, c : \mathit{counter} \bullet 0)$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad StoreFieldF \quad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}} \\
\Delta FieldSession \\
\hline
computed' = computed \oplus no\_dose \\
prescribed' = prescribed \oplus (prescrip \lhd measured) \oplus no\_counter \oplus \{nfrac \mapsto 1\} \\
accumulated' = zero\_counter \\
overridden' = \varnothing \\
calibrated' = calibrated \\
\hline
\end{array}
$$

6.4.6 Experiment Mode

This operation toggles modes with no field. There are no dose and time (8.9.11, 202, second paragraph from bottom).

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad ExptModeF \quad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}} \\
\Delta Field \\
\hline
NoFieldF' \\
computed' = computed \oplus no\_dose \\
calibrated' = calibrated \\
\hline
\end{array}
$$

6.5 Calibration factors

6.5.1 Dosimetry calibration

Dosimetry calibration factors, including the dose rate and treatment time factor used to calculate the backup time, appear on the **Dosimetry Calibration** display (8.9.13, 213 – 214)[12]. Table 3 shows part of a possible design for this display. The *calibrated* values in the left column are read from files or measured by sensors, while the *computed* values in the right column are computed by the control program or entered by the operator using the *EditCalF* operation.

The pressure-temperature correction factors are used to adjust the standard calibration voltages for the dosimetry system (8.9.13, 213 – 215). The *calibrated* $calvolt1$ and *calibrated* $calvolt2$ represent the standard calibration voltages on file (8.9.13, 213, second paragraph from bottom), while *computed* $calvolt1$ (etc.) represent the calibration voltages actually in effect,

[12] Called **Cal Factors** in [2], since renamed to distinguish it from the forthcoming **LCC Calibration** etc.

	MEASURED/CALIBRATED	ADJUSTED
P/T MODE	*computed pt_mode (automatic/manual)*	
PRESSURE	*calibrated press*	*computed press*
TEMPERATURE	*calibrated temp*	*computed temp*
P/T CORR.	*calibrated pt_factor*	*computed pt_factor*
CALVOLT 1	*calibrated calvolt*1	*computed calvolt*1
CALVOLT 2	*calibrated calvolt*2	*computed calvolt*2
DOSE RATE	*calibrated d_rate*	*computed d_rate*
TIME FACTOR	*calibrated t_fac*	*computed t_fac*

Table 3: Dosimetry calibration display

which are obtained by adusting the standard calibration voltage by a barometric pressure/temperature correction factor (8.9.13, 213 bottom paragraph, 214 top paragraph)[13].

$$PRESSURE == VALUE; TEMPERATURE == VALUE$$

$$\begin{array}{|l}
pt_formula : (PRESSURE \times TEMPERATURE) \nrightarrow FACTOR \\
\hline
\forall p : valid\ press;\ t : valid\ temp \bullet \\
\qquad (p, t) \in \operatorname{dom} pt_formula \land pt_formula(p, t) \in valid\ pt_factor
\end{array}$$

The *computed press* and *computed temp* are the pressure and temperature entered by the operator, and while *calibrated temp* and *calibrated press* are measured continously by sensors. The *computed pt_factor* stores the *barometric pressure/temperature correction factor of the day* calculated from the readings entered by the operator (8.9.13, 214, third paragraph), while *calibrated pt_factor* stores the *automatic pressure/temperature correction factor* calculated from sensor readings (8.9.13, 214, fourth paragraph). The *pressure-temperature* interlock (section 7) accounts for the possibility that the pressure or temperature values might be invalid or expired.

The operator sets *computed pt_mode* $=$ *automatic* to use the *automatic pressure/temperature correction factor*, and *computed pt_mode* $=$ *manual* to use the correction factor that is based on the manually entered values (8.9.13, 214, fifth paragraph).

$$\begin{array}{|l} automatic, manual : VALUE \end{array}$$

The *ScanPT* operation computes the correction factors and updates the registers with the new values.

[13] The pressure-temperature factor is given by $pt\_factor = (press/1013) \times (295/(temp + 273))$, where *press* and *temp* are in mbar and deg. C, respectively. We do not attempt to model this floating-point calculation in Z.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{ScanPT}} \\
\hline
\Delta \textit{Field} \\
\hline
\textit{calibrated}'\ \textit{pt\_factor} = \textit{pt\_formula}(\textit{calibrated press}, \textit{calibrated temp}) \\
\textit{computed}'\ \textit{pt\_factor} = \textit{pt\_formula}(\textit{computed press}, \textit{computed temp}) \\
\\
(\mathbf{let}\ \textit{pt\_corr} == \mathbf{if}\ \textit{computed pt\_mode} = \textit{automatic} \\
\qquad\qquad\qquad\qquad \mathbf{then}\ \textit{calibrated}'\ \textit{pt\_factor}\ \mathbf{else}\ \textit{computed}'\ \textit{pt\_factor} \bullet \\
\qquad \textit{computed}'\ \textit{calvolt}1 = \textit{pt\_corr} * \textit{calibrated calvolt}1 \land \\
\qquad \textit{computed}'\ \textit{calvolt}2 = \textit{pt\_corr} * \textit{calibrated calvolt}2) \\
\\
\{\textit{pt\_factor}\} \lhd \textit{calibrated}' = \{\textit{pt\_factor}\} \lhd \textit{calibrated} \\
\{\textit{pt\_factor}, \textit{calvolt}1, \textit{calvolt}2\} \lhd \textit{computed}' = \{\textit{pt\_factor}, \textit{calvolt}1, \textit{calvolt}2\} \lhd \textit{computed} \\
\\
\textit{prescribed}' = \textit{prescribed} \\
\textit{accumulated}' = \textit{accumulated} \\
\textit{overridden}' = \textit{overridden} \\
\hline
\end{array}
$$

ScanPT is scheduled by the control program itself; it is not invoked by the user.

7 Software interlocks and status flags

(To come)

8 User interface

The user may provide input at the workstation at any time (by typing, pressing function keys or cursor arrow keys — in our implementation we do not use the mouse). We model each keystroke and the actions it invokes as an *Event* that accepts an *input*? that may change the *Console* state.

$$
\begin{array}{|l}
\hline
\textit{Event} \\
\Delta \textit{Console} \\
\textit{input}? : \textit{INPUT} \\
\hline
\end{array}
$$

We do not attempt to formalize any "look and feel" aspects of the user interface, such as the appearance of the display. They are already described in sufficient detail in [2], chapters 2 and 8.

INPUT is the set of inputs (keypresses) the user can provide[14]. Here is the list of inputs at this writing.

$$
\begin{aligned}
\textit{INPUT} ::= {} & \textit{filter\_wedge} \mid \textit{leaf\_collim} \mid \textit{dose\_intlk} \mid \textit{gantry\_psa} \mid \textit{dose\_cal} \mid \\
& \textit{startup} \mid \textit{help} \mid \textit{messages} \mid \textit{select\_patient} \mid \textit{select\_field} \mid \textit{field\_summary} \mid \\
& \textit{login} \mid \textit{edit\_setting} \mid \textit{edit\_dose\_reg} \mid \textit{log\_message} \mid \textit{store\_field} \mid \textit{override\_cmd} \mid \\
& \textit{cancel\_run} \mid \textit{password} \mid \textit{auto\_setup} \mid \textit{expt\_mode} \mid \textit{cancel} \mid \textit{refresh} \mid \textit{shutdown} \mid \\
& \textit{select} \mid \textit{ret} \mid \textit{character} \mid \textit{backspace} \mid \textit{delete\_key} \mid \\
& \textit{left\_arrow} \mid \textit{right\_arrow} \mid \textit{up\_arrow} \mid \textit{down\_arrow} \mid \textit{ignored}
\end{aligned}
$$

Many operations are invoked by pressing keys, so it is often convenient to identify operations with the corresponding input. Therefore we assign them to the same type. Here is the list of operations at this writing.

$$
\begin{array}{|l}
\textit{OP} : \mathbb{P}\, \textit{INPUT} \\
\hline
\textit{OP} = \{\textit{filter\_wedge}, \textit{leaf\_collim}, \textit{dose\_intlk}, \textit{gantry\_psa}, \textit{dose\_cal}, \\
\quad \textit{startup}, \textit{help}, \textit{messages}, \textit{select\_patient}, \textit{select\_field}, \textit{field\_summary}, \\
\quad \textit{login}, \textit{edit\_setting}, \textit{edit\_dose\_reg}, \textit{log\_message}, \textit{store\_field}, \textit{override\_cmd}, \\
\quad \textit{cancel\_run}, \textit{password}, \textit{auto\_setup}, \textit{expt\_mode}, \textit{cancel}, \textit{refresh}, \textit{shutdown}, \textit{select}\}
\end{array}
$$

The user interface shows many displays, for example the login display (Fig 8.1, 178), the patient list display (Fig. 8.2, 185), the leaf collimator display (Fig. 8.7, 197) etc. The operator can choose any display by pressing a key, so we can identify displays with these operations.

[14] In the implementation, inputs are X window system events and the values of *INPUT* correspond to X keysyms [5].

$$
\begin{array}{|l}
DISPLAY : \mathbb{P}\, OP \\
\hline
DISPLAY = \{\mathit{filter\_wedge}, \mathit{leaf\_collim}, \mathit{dose\_intlk}, \mathit{gantry\_psa}, \mathit{dose\_cal}, \\
\qquad \mathit{startup}, \mathit{help}, \mathit{messages}, \mathit{select\_patient}, \mathit{select\_field}, \\
\qquad \mathit{field\_summary}, \mathit{login}\}
\end{array}
$$

8.1 Console state

This section describes the variables in the *Console* state.

The first variable indicates the mode of *interaction*. If no interaction is in progress the console is *available*, or there may be a *dialog* in progress where the user is typing text into a dialog box, or there may be a *menu* displayed, or the user may be asked to *confirm* some operation by providing a yes/no answer (this mode can also be used to present informational messages).

$$INTERACTION ::= available \mid dialog \mid menu \mid confirm$$

The *op* variable keeps track of which top-level operation (described in [2]) is underway.

The *display* variable indicates which of the screen designs pictured in the informal specification is currently visible on the display. The *display* variable determines which items appear and helps determine which operations are available.

The *item* state variable holds the item which the operator has selected from a tabular display, for example the setting which the operator is editing.

The *nlist* state variable holds the list of names (of patients or fields) that appear on a list display, and *list_item* indicates the currently selected name.

The *menu_item* state variable holds the index of the current menu selection (a small integer).

$$
\begin{array}{|l}
nmax : \mathbb{N}
\end{array}
$$

$$SELECTION == \{i : \mathbb{N} \mid i \leq nmax\}$$

The *buffer* state variable models the (possibly incomplete) string that the user edits in dialog mode.

$$[STRING]$$

$$| \quad empty : STRING$$

The *keyswitch* must be unlocked to allow the console to be used (8.7, 179).

$$KEYSWITCH ::= locked \mid unlocked$$

Some operations are available only when a treatment is being set up, and are locked out while a treatment *run* is in progress (8.8.2, 183).

$$RUN ::= setup \mid running$$

The *keyswitch* and *run* variables depend on sensor inputs; they are not constrained here.

Together, these variables describe the state of the user interaction.

$$\begin{array}{|l}
\hline
\textit{Console} \\
keyswitch : KEYSWITCH \\
run : RUN \\
display : DISPLAY \\
op : OP \\
interaction : INTERACTION \\
item : ITEM \\
nlist : \mathbb{P}\, NAME \\
list_item : NAME \\
menu_item : SELECTION \\
buffer : STRING \\
\hline
\end{array}$$

When the control program starts up, the login process begins (section 8.3.13)[15].

$$\begin{array}{|l}
\hline
\textit{InitConsole} \\
Console \\
\hline
op = login \\
display = login \\
interaction = dialog \\
buffer = empty \\
\hline
\end{array}$$

8.2 Elements of user interaction

All user interactions are built up from a few elements. In this section we define the constants, states and operations that serve as building blocks.

[15]When the implementation starts up, the *startup* screen appears first. The login process does not begin until the various *Init*... conditions are established. We do not model this formally.

The *caption* type models messages or other output to the operator that appear temporarily at the console (in dialog boxes or perhaps even from the speaker, see 2.2.3, 9). Captions are distinguished from *log messages* which appear in a different location on the console and are also stored in log files along with timestamps other information (2.2.4, 9).

$$[CAPTION, MESSAGE]$$

Ignore is the default do-nothing operation that is invoked when a key is pressed but the preconditions for the associated operation are not satisfied. *Ignore* does not change the state, but issues an alert (such as sounding the workstation bell) to notify the user that the input was received but the operation is not enabled.

$$\mid \; alert : CAPTION$$

$$\begin{array}{|l}
\textit{Ignore} \\
\hline
Event \\
\Xi Console \\
caption! : CAPTION \\
\hline
caption! = alert \\
\hline
\end{array}$$

The keyswitch must be unlocked for any operation to occur. When the keyswitch is locked, input is ignored:

$$Unlocked \mathrel{\widehat=} [\, Console \mid keyswitch = unlocked \,]$$

$$EventUnlocked \mathrel{\widehat=} Event \land Unlocked$$

Many operations are invoked by pressing the *select* key.

$$Select \mathrel{\widehat=} [\, EventUnlocked \mid input? = select \,]$$

It is convenient to describe the operations that can occur in each of the interaction modes. Each mode is described in a following subsection.

8.2.1 Available

Most of the top-level operations described in [2] can only be selected when the console is available.

$$Available \mathrel{\widehat=} [\, Console \mid interaction = available \,]$$

$$
\begin{array}{|l}
\hline
\textit{Op} \\
\hline
\textit{EventUnlocked} \\
\hline
\textit{Available} \\
\textit{input?} \in \textit{OP} \\
\hline
\end{array}
$$

Certain operations have stronger preconditions: they cannot occur when a run is in progress (8.8.2, 183). A few operations occur only when a run is in progress (8.9.11, 209-210).

$$Setup \mathrel{\widehat{=}} [\, Available \mid run = setup \,]$$

$$Running \mathrel{\widehat{=}} [\, Available \mid run = running \,]$$

When the console is available, the user may select a new *display*. The console remains available.

$$
\begin{array}{|l}
\hline
\textit{SelectDisplay} \\
\hline
\textit{Op} \\
\hline
\textit{input?} \in \textit{DISPLAY} \\
\textit{display}' = \textit{input?} \\
\textit{op}' = \textit{display}' \\
\textit{Available}' \\
\hline
\end{array}
$$

SelectDisplay operations may change *item* and *list_item* (see below) but do not change other state variables (for brevity we omit the $x' = x$ "nothing changes" predicates).

When an interaction is in progress, the console is *Engaged*. The *Done* operation schema describes what happens when an interaction completes: the console returns to the *Available* state, and *op* returns to its value when the display was selected.

$$Engaged \mathrel{\widehat{=}} [Console \mid interaction \neq available]$$

$$
\begin{array}{|l}
\hline
\textit{Done} \\
\hline
\textit{EventUnlocked} \\
\hline
\textit{Engaged} \\
\textit{op}' = \textit{display} \\
\textit{display}' = \textit{display} \\
\textit{Available}' \\
\hline
\end{array}
$$

The *Cancel* operation is used to end an interaction without making permanent changes to the underlying machine state.

$$Cancel \mathrel{\widehat{=}} [Done \mid input? = cancel]$$

8.2.2 Lists

Certain displays show a list of names (patients or fields). When a list display is selected, *nlist* is loaded, and the default *list_item* is assigned. If the list is not empty, the *List* state results (the patient list might be empty if there are no patients on file; the field list is always empty when there is no patient, and may be empty if there are no fields on file for the selected patient).

$$\begin{array}{|l} list : \mathbb{P}\ DISPLAY \\ default\_name : \mathbb{P}_1\ NAME \longrightarrow NAME \\ \hline \forall\ list : \mathbb{P}_1\ NAME \bullet default\_name\ list \in list \end{array}$$

$$List \mathrel{\widehat{=}} [\,Available \mid display \in list \land nlist \neq \varnothing \land list\_item \in nlist\,]$$

$$\begin{array}{|l} \textit{SelectList} \\ \hline SelectDisplay \\ \hline input? \in list \\ ((nlist = \varnothing \land list\_item' = no\_name) \\ \lor\ (List' \land list\_item' = default\_name\ nlist')) \\ \hline \end{array}$$

Here $display \in list \land nlist \neq \varnothing$ distinguishes the *List* state, and this test occurs explicitly in the implementation. In contrast, $list\_item \in nlist$ is an invariant. It need not be coded as an explicit test but it must be maintained or else the implementation might abort (because *list_item* is used as an index into *nlist*).

The console indicates *list_item* (for example by placing a highlight or cursor over that name in the list). Subsequently the user can choose a new name from the list by using the up and down-arrow keys. The function *aname* calculates the new name by "dead reckoning" from the old name, the list, and the arrow key (it is not necessary for the program to poll the console for the cursor position). The list remains visible.

$$v\_arrow == \{up\_arrow, down\_arrow\}$$

$$\begin{array}{|l} aname : (v\_arrow \times NAME \times \mathbb{P}_1\ NAME) \longrightarrow NAME \\ \hline \forall\ a : v\_arrow;\ n : NAME;\ list : \mathbb{P}_1\ NAME \bullet aname(a, n, list) \in list \end{array}$$

$$Continue \mathrel{\widehat{=}} [\,\Delta Console \mid interaction' = interaction \land op' = op \land display' = display\,]$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{GetListArrow}} \\
EventUnlocked \\
\Delta List \\
\hline
input? \in v\_arrow \\
list\_item' = aname(input?, list\_item, nlist) \\
Continue \\
\hline
\end{array}
$$

This is a *Continue* operation that does not change *interaction*, *op*, or *display*. Here *list_item* is the only state variable that changes. We do not completely specify *default_name* and *aname*; we leave that to the implementation. Here we merely provide the predicates needed to ensure that the implementation does not abort.

The user presses the *select* key to choose the current *list_item* for some purpose. The selection is logged; *nmessage* converts the name to a log message.

$$
\mid\ selected\_msg : NAME \longrightarrow MESSAGE
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{SelectName}} \\
Select \\
name! : NAME \\
message! : MESSAGE \\
\hline
List \\
name! = list\_item \\
message! = selected\_msg\ name! \\
\hline
\end{array}
$$

GetListArrow and *SelectName* are not total operations; they do not handle the case where $nlist = \varnothing$. The latter case is handled by a default do-nothing operation, *IgnoreOthers* (section 8.4).

8.2.3 Tables

Certain displays show a table of items (settings for one subsystem, calibration factors etc.). The constant *table_items* tells which items on each table can be selected for editing or overriding (additional items may be displayed as well). When a tabular display is selected, the default *item* is assigned, and the *Table* state results.

$$
\mid\ table : \mathbb{P}\, DISPLAY
$$

$$
\begin{array}{|l}
default\_item : table \longrightarrow ITEM \\
table\_items : table \longrightarrow \mathbb{P}_1\, ITEM \\
\hline
\forall\, d : table \bullet default\_item\ d \in table\_items\ d
\end{array}
$$

$$Table \mathrel{\widehat{=}} [\, Available \mid display \in table \land item \in table\_items\ display \,]$$

$$
\begin{array}{|l}
\hline
\textit{SelectTable} \\
SelectDisplay \\
\hline
input? \in table \\
item' = default\_item\ display' \\
Table' \\
\hline
\end{array}
$$

Subsequently the user can indicate a new item on the table by using all four arrow keys.

$$arrow == \{right\_arrow, left\_arrow\} \cup v\_arrow$$

$$
\begin{array}{|l}
asetting : (arrow \times ITEM \times table) \longrightarrow ITEM \\
\hline
\forall\, a : arrow;\ s : ITEM;\ d : table \bullet asetting(a, s, d) \in table\_items\ d
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{GetSettingArrow} \\
EventUnlocked \\
\Delta Table \\
\hline
input? \in arrow \\
item' = asetting(input?, item, display) \\
Continue \\
\hline
\end{array}
$$

Here *item* is the only state variable that changes.

Items can be selected from tabular displays for editing or overriding. Editing or overriding is only enabled in the *Setup* state (when a treatment run is not in progress, see 8.8.2, 183). Pressing the *select* key when certain tabular displays are present invokes an editing operation: *edit_setting* if the selected item is a *setting* and *edit_dose_reg* if it is a *dose_reg* (notice that here op' is *not* the same as $input?$). Therefore it is necessary to separate *setting* and *dose_reg* items on different tables[16].

[16] Because the implementation cannot distinguish *setting* from *dose_reg* based on *item* alone; *item* values are just C `enum` values (integers).

$$
\begin{array}{|l}
setting\_table, dose\_reg\_table : \mathbb{P}\, table \\
\hline
\forall d : setting\_table \bullet table\_items\ d \subseteq setting \\
\forall d : dose\_reg\_table \bullet table\_items\ d \subseteq dose\_reg
\end{array}
$$

$$
\begin{array}{|l}
\textit{SelectItem} \\
\hline
Select \\
\hline
Setup \\
Table \\
item' = item \\
(op' = edit\_dose\_reg \land display \in dose\_reg\_table \lor \\
op' = edit\_setting \land display \in setting\_table) \\
\hline
\end{array}
$$

The postcondition here implies *Editing*, the invariant of the editing state[17].

$$
\begin{array}{|l}
\textit{Editing} \\
\hline
Console \\
\hline
interaction \in \{dialog, menu\} \\
(op = edit\_dose\_reg \land item \in dose\_reg \lor \\
op = edit\_setting \land item \in setting) \\
\hline
\end{array}
$$

The *Setup* precondition of *SelectItem* prevents the console entering the *Editing* state when a run is in progress. Other mechanisms prevent the machine from beginning a run while in the *Editing* state.

8.2.4 Confirm

Confirm interactions present a query ("Are you sure ... ?") and wait for the user to provide a yes/no answer, indicated by the *select* or *cancel* keys (for example see 8.9.11, 210). Each *Confirm* operation presents a confirmation box (a sort of dialog box) with a *caption* that identifies the operation, and the *query*. The display under the confirmation box does not change.

$$Confirm \mathrel{\widehat{=}} [\, Console \mid interaction = confirm \,]$$

$$
\begin{array}{|l}
ocaption : OP \longrightarrow CAPTION
\end{array}
$$

[17] The implementation uses *op* to determine whether *item* is an index into *setting* or *dose_reg*.

$$
\begin{array}{|l}
\hline
\textit{ConfirmOp} \\
\textit{Op} \\
\textit{caption!}, \textit{query!} : \textit{CAPTION} \\
\hline
\textit{caption!} = \textit{ocaption}\ \textit{op}' \\
\textit{display}' = \textit{display} \\
\textit{Confirm}' \\
\hline
\end{array}
$$

$$\textit{AcceptConfirm} \mathrel{\widehat{=}} \textit{Confirm} \land \textit{Select} \land \textit{Done}$$

8.2.5 Menu

When the console is *Available* the user can invoke a menu, then make a selection from the menu. Each menu includes a caption and a list of menu entries. The display does not change.

$$
\begin{array}{|l}
\textit{default\_selection} : \textit{SELECTION}
\end{array}
$$

$$\textit{Menu} \mathrel{\widehat{=}} [\, \textit{Editing} \mid \textit{interaction} = \textit{menu} \,]$$

$$
\begin{array}{|l}
\hline
\textit{MenuOp} \\
\textit{Op} \\
\textit{caption!} : \textit{CAPTION} \\
\textit{menu!} : \mathrm{iseq}\, \textit{CAPTION} \\
\hline
\textit{menu\_item}' = \textit{default\_selection} \\
\textit{display}' = \textit{display} \\
\textit{Menu}' \\
\hline
\end{array}
$$

Here *op* also changes; the other state variables retain the same values.

Menus are used to choose new values for *selection* items; valid *selection* values are small integers. Combining *MenuOp* with *SelectItem* yields the *MenuEdit* operation. The menu shows the item name and a sequence of descriptive strings indexed by the corresponding item values. Here again, the *Editing* postcondition of *SelectItem* guarantees that *op* can be used to help look up *selection_values item*.

$$
\begin{array}{|l}
\textit{setting\_info\_name} : \textit{ITEM} \longrightarrow \textit{CAPTION} \\
\textit{setting\_value} : \textit{selection} \longrightarrow \mathrm{iseq}\, \textit{CAPTION} \\
\hline
\forall\, s : \textit{selection} \bullet \mathrm{dom}\,(\textit{setting\_value}\ s) = \textit{valid}\ s
\end{array}
$$

$$\begin{array}{|l}
\multicolumn{1}{l}{_\ MenuEdit\ __________} \\
MenuOp \\
SelectItem \\
\hline
item \in selection \\
caption! = setting_info_name\ item;\ menu! = setting_value\ item \\
\hline
\end{array}$$

There are functions to return the default menu selection and the new selection after each up or down-arrow keypress.

$$\begin{array}{|l}
amenu : (v_arrow \times SELECTION \times selection) \longrightarrow SELECTION \\
\hline
\forall\, s : selection \bullet (\mathbf{let}\ n == \#(valid\ s) \bullet \\
\qquad \forall\, a : v_arrow;\ i : SELECTION \bullet default_selection \leq n \land amenu(a, i, s) \leq n)
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{_\ GetMenuArrow\ __________} \\
EventUnlocked \\
\Delta Menu \\
\hline
input? \in v_arrow \\
menu_item' = amenu(input?, menu_item, item) \\
Continue \\
\hline
\end{array}$$

Here *item* is the only variable that changes.

The user presses *select* to accept the current menu item and the console becomes available again.

$$AcceptMenu \mathrel{\widehat=} Menu \land Select \land Done$$

$$\begin{array}{|l}
\multicolumn{1}{l}{_\ MenuSettingC\ __________} \\
AcceptMenu \\
item! : ITEM \\
value! : VALUE \\
\hline
Editing \\
item! = item \\
value! = menu_item \\
\hline
\end{array}$$

8.2.6 Dialog

When the console is *Available* the user can begin a dialog, then type and edit text in a dialog box. The dialog box contains a *caption* and a *prompt* that may include the values of other state variables. The display under the dialog box does not change.

$$Dialog \mathrel{\widehat=} [\, Console \mid interaction = dialog \,]$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad DialogOp \quad}} \\
Op \\
caption!, prompt! : CAPTION \\
\hline
caption! = ocaption\ op' \\
display' = display \\
Dialog' \\
\hline
\end{array}$$

Here only *interaction*, *buffer*, and *op* change. The *buffer* may be emptied, or may be filled with a convenient default value. We'll describe changes to *op* later, with each dialog operation.

The console remains in *Dialog* while the user types and edits. The *GetChar* operation gets a single character and updates the buffer as described by the *modify* function (append printing characters to the end of *buffer*, and do the appropriate things with editing characters).

$$\begin{array}{|l}
CHAR : \mathbb{P}\, INPUT
\end{array}$$

$$\begin{array}{|l}
modify : (STRING \times CHAR) \longrightarrow STRING
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad GetChar \quad}} \\
EventUnlocked \\
\Delta Dialog \\
\hline
input? \in CHAR \\
buffer' = modify(buffer, input?) \\
Continue \\
\hline
\end{array}$$

Here *buffer* is the only variable that changes.

When a dialog is done, the dialog box disappears and the console becomes available again. At any time the user can *cancel* the dialog and discard the input. To submit the input, the user presses a *terminator* key; the program can *Accept* the input or *Reprompt* (the user may also *Cancel* the dialog).

$$\begin{array}{|l}
terminator : \mathbb{P}\, INPUT
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Accept \quad}} \\
Done \\
\hline
Dialog \\
input? \in terminator \\
\hline
\end{array}$$

$$
\begin{array}{|l}
\hline
\textit{Reprompt} \\
\textit{EventUnlocked} \\
\Delta \textit{Dialog} \\
\hline
\textit{input?} \in \textit{terminator} \\
\textit{buffer}' = \textit{empty} \\
\textit{Continue} \\
\hline
\end{array}
$$

Here again, *buffer* is the only variable that changes.

Dialogs are frequently used to edit item values. Combining *DialogOp* with *SelectItem* yields the *DialogEdit* operation. Dialog box editing begins if the selected item is not a *selection* (does not have just a few discrete values). The program captions the dialog box with the item name and the minimum and maximum valid item values[18].

$$MIN == VALUE; MAX == VALUE$$

$$\textit{setting\_info} : ITEM \times MIN \times MAX \longrightarrow CAPTION$$

$$
\begin{array}{|l}
\hline
\textit{DialogEdit} \\
\textit{DialogOp} \\
\textit{SelectItem} \\
\hline
\textit{item} \notin \textit{selection} \\
\textit{prompt!} = (\mathbf{let}\ v == \textit{valid item} \bullet \textit{setting\_info}(\textit{item}, \textit{min}\ v, \textit{max}\ v)) \\
\hline
\end{array}
$$

The implementation uses the value of *op* guaranteed by the *Editing* postcondition of *SelectItem* to look up *valid item*; there are separate *valid* arrays for *dose_reg* and *setting*.

When the user presses a terminator key, the program attempts to convert the buffer contents to a (numeric) value (non-numeric strings are always converted to an out-of-range value). If the conversion succeeds and the value is valid for the item, the dialog ends and the item and its value are reported; otherwise, the program reprompts.

$$\textit{sval} : STRING \longrightarrow VALUE$$

$$
\begin{array}{|l}
\hline
\textit{EditSettingC} \\
\textit{Accept} \\
\textit{item!} : ITEM \\
\textit{value!} : VALUE \\
\hline
\textit{Editing} \\
\textit{item!} = \textit{item} \\
(\mathbf{let}\ v == \textit{sval buffer} \bullet v \in \textit{valid item} \land \textit{value!} = v) \\
\hline
\end{array}
$$

[18] The dialog box caption also includes the units, but we do not model this formally.

$$InvalidSetting \mathrel{\widehat=} [\, Reprompt \mid Editing \land sval\ buffer \notin valid\ item \,]$$

In the implementation it is convenient to combine these two operations[19].

$$EditOrInvalidSetting \mathrel{\widehat=} EditSettingC \lor InvalidSetting$$

8.2.7 Summary

Table 4 lists the schemas defined in the preceding subsections. Underlined names are state schemas, others are operation schemas.

The table shows the schema inclusion hierarchy. Schemas are indented under the state schemas they include (for example the *SelectDisplay* operation and the *List* state both include the *Available* state). State schemas indented at the same level are mutually exclusive or independent of one another (*List* and *Table* are mutually exclusive while *List* and *Setup* are independent). Operation schemas are followed in parentheses by the operation schemas they include (so *DialogEdit* includes the *SelectItem* and *DialogOp* operations).

The table also shows transition involving certain state variables, especially *interaction* and *op*. Operation schema names are followed by their postconditions, so the postconditions of the *DialogOp* include the *Dialog* state and the postcondition of *SelectItem* include the *Editing* state. Only postconditions that indicate state changes are shown; $x' = x$ "no change" postconditions are not shown. Postconditions are not shown when they can be inferred from included operations, for example *DialogEdit* includes both *DialogOp* and *SelectItem*, so its postcondition includes both *Dialog* and *Editing*.

The table shows how the program can alternate between *Available* and *Engaged* states. Under *Available* the *DialogOp*, *MenuOp*, and *ConfirmOp* operations result in *Dialog*, *Menu* and *Confirm*, respectively, under *Engaged*. From there, the *Done* operations *Accept* and *Cancel* (etc.) return to *Available*.

The tables shows changes in *op*: *SelectDisplay* and *Done* set $op' = display'$, *SelectItem* sets op' to one of the *edit* operations. *GetChar*, the *GetArrow* operations and *Reprompt* do not change *op*. Other changes to op' are determined in the specific operations in the application (below), not these building blocks.

[19] In *EditOrInvalidSetting*, the two outputs *item*! and *value*! are not used in the *Invalid* case.

- $\underline{\mathit{Unlocked}}$
    - *Select*
    - $\underline{\mathit{Available}}$
        - *Op*
        - *SelectDisplay (Op)*, $op' = display' = input?$
        - *SelectList (SelectDisplay)*, $\underline{\mathit{List}'}$
        - *SelectTable (SelectDisplay)*, $\underline{\mathit{Table}'}$
        - *ConfirmOp (Op)*, $\underline{\mathit{Confirm}'}$
        - *MenuOp (Op)*, $\underline{\mathit{Menu}'}$
        - *DialogOp (Op)*, $\underline{\mathit{Dialog}'}$
        - $\underline{\mathit{List}}$
            - *GetListArrow*
            - *SelectName (Select)*
        - $\underline{\mathit{Table}}$
            - *GetSettingArrow*
        - $\underline{\mathit{Setup}}$
            - ($\underline{\mathit{Table}}$)
                - *SelectItem (Select)*, $\underline{\mathit{Editing}'}$, $op' = edit\_setting \lor op' = edit\_cal$
                - *MenuEdit(SelectItem, MenuOp)*
                - *DialogEdit(SelectItem, DialogOp)*
        - $\underline{\mathit{Running}}$
    - $\underline{\mathit{Engaged}}$
        - *Done*, $\underline{\mathit{Available}'}$, $op' = display$
        - *Cancel (Done)*
        - $\underline{\mathit{Confirm}}$
            - *AcceptConfirm (Select, Done)*
        - $\underline{\mathit{Menu}}$
            - *GetMenuArrow*
            - *AcceptMenu (Select, Done)*
            - ($\underline{\mathit{Editing}}$)
                - *MenuSettingC (AcceptMenu)*
        - $\underline{\mathit{Dialog}}$
            - *GetChar*
            - *Accept (Done)*
            - *Reprompt*
            - ($\underline{\mathit{Editing}}$)
                - *EditOrInvalidSetting (Accept* or *Reprompt)*

Table 4: User interface building blocks

8.3 Therapy console operations

In this subsection we present the operations described in [2], in the order their constituent building blocks appear in Table 4.

Several building block operations require no further elaboration: *SelectDisplay*, *SelectTable*, *GetListArrow*, *GetSettingArrow*, *Cancel*, *GetMenuArrow*, *MenuSettingC*, *GetChar*, and *EditOrInvalidSetting* and are already complete. Others require further specialization in the following subsections.

8.3.1 Relation to *Session* state

A few *Console* operations read (but do not change) variables from the *Session* state (section 5). When the *Console* shows the patient or field list, its *nlist* state variable holds the patients or fields from the *Session* state. This is expressed by the *ConsoleSession* invariant:

$$
\begin{array}{|l}
\textit{ConsoleSession} \\
\hline
\textit{Console} \\
\textit{Session} \\
\hline
\textit{display} = \textit{select\_patient} \Rightarrow \textit{nlist} = \textit{names} \\
\textit{display} = \textit{select\_field} \Rightarrow \textit{nlist} = \operatorname{dom} \textit{fields} \\
\hline
\end{array}
$$

8.3.2 *Op* operations

Several operations are based only on *Op*. They are *Continue* operations because they do not involve any ongoing interaction, just a single keypress.

$$\textit{SimpleOp} \mathrel{\widehat{=}} \textit{Op} \wedge \textit{Continue}$$

Experiment Mode (8.9.6, 190):

$$
\begin{array}{|l}
\textit{ExptModeC} \\
\hline
\textit{SimpleOp} \\
\Xi \textit{Session} \\
\Delta \textit{ConsoleSession} \\
\hline
\textit{Setup} \\
\textit{input?} = \textit{expt\_mode} \\
\textit{operator} \in \textit{physicists} \\
\hline
\end{array}
$$

Auto Setup (8.8.1, 181):

$$auto\_setup\_display == \{field\_summary, filter\_wedge, leaf\_collim, dose\_intlk\}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{AutoSetupC} \\
SimpleOp \\
\Xi ConsoleSession \\
subsystem! : auto_setup_display \\
\hline
Setup \\
field \neq no_field \\
display \in auto_setup_display \\
input? = auto_setup \\
subsystem! = display \\
\hline
\end{array}$$

8.3.3 *SelectDisplay* operations

There are a few simple displays that provide no selections or interactive editing, **Field Summary** (8.9.7, 191) and the *help* display (not discussed in [2]):

$$simple\_display == \{field\_summary, help\}$$

8.3.4 *SelectList* operations

The specializations of *SelectList* are **Select Patient** (8.9.3, 184) and **Select Field** (8.9.4, 186). The latter operation only makes sense when a patient has been selected:

$$list = \{select\_patient, select\_field\}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{SelectPatientList} \\
SelectList \\
\Xi Session \\
\Delta ConsoleSession \\
\hline
input? = select_patient \\
nlist' = names \\
\hline
\end{array}$$

$$
\begin{array}{|l}
\hline
\textit{SelectFieldList} \\
\hline
\textit{SelectList} \\
\Xi \textit{Session} \\
\Delta \textit{ConsoleSession} \\
\hline
\textit{patient} \neq \textit{no\_patient} \\
\textit{input?} = \textit{select\_field} \\
\textit{nlist}' = \operatorname{dom} \textit{fields} \\
\hline
\end{array}
$$

8.3.5 *SelectTable* operations

The *table* displays are **Gantry/PSA** (8.9.8, 193), **Filter/Wedge** (8.9.9, 194), **Leaf Collimator** (8.9.10, 196), **Dosimetry/Therapy Interlocks** (8.9.11, 199) and **Calibration Factors** (8.9.13, 213):

$$\textit{table} = \{\textit{gantry\_psa}, \textit{filter\_wedge}, \textit{leaf\_collim}, \textit{dose\_intlk}, \textit{dose\_cal}\}$$

The constant *table_items* tells which items on each table can be selected for editing or overriding (additional items may be displayed as well).

$$
\begin{aligned}
\textit{table\_items} = \{&\textit{gantry\_psa} \mapsto \{\textit{gantry}, \textit{collim}, \textit{turnt}\}, \textit{filter\_wedge} \mapsto \{\textit{filter}, \textit{wedge}, \textit{w\_rot}\}, \\
&\textit{leaf\_collim} \mapsto \textit{leaves}, \textit{dose\_intlk} \mapsto \{\textit{p\_dose}, \textit{p\_time}\}, \\
&\textit{dose\_cal} \mapsto \{\textit{pt\_mode}, \textit{press}, \textit{temp}, \textit{d\_rate}, \textit{t\_fac}\}\}
\end{aligned}
$$

It is necessary to separate *setting* and *dose_reg* items on different tables:

$$\textit{setting\_table} = \{\textit{gantry\_psa}, \textit{filter\_wedge}, \textit{leaf\_collim}\}$$

$$\textit{dose\_reg\_table} = \{\textit{dose\_intlk}, \textit{dose\_cal}\}$$

Now that these constants are defined, the previously defined *SelectTable* operation requires no further specialization.

8.3.6 *ConfirmOp* operations

To begin **Cancel Run** (8.9.11, 209 – 210):

$$
\begin{array}{|l}
\textit{cancel\_run\_query} : \textit{CAPTION}
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{SelectCancelRun} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\mathit{ConfirmOp} \\
\hline
\mathit{Running} \\
input? = cancel\_run \\
op' = input? \\
query! = cancel\_run\_query \\
\hline
\end{array}
$$

The complementary *AcceptConfirm* operation is *CancelRunC* (below).

8.3.7 *MenuOp* operations

At this writing there are no simple *MenuOp* operations, only *MenuEdit* operations (under *Setup*, below).

8.3.8 *DialogOp* operations

$$
\begin{array}{|l}
type\_message\_prompt, store\_field\_prompt : CAPTION
\end{array}
$$

To begin **Write Log Message** (2.5.1, 17):

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{TypeMessage} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\mathit{DialogOp} \\
\hline
input? = log\_message \\
op' = input? \\
prompt! = type\_message\_prompt \\
\hline
\end{array}
$$

To begin **Store Field** (8.9.5, 189 – 188):

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{EditField} \quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad}} \\
\mathit{DialogOp} \\
\hline
\mathit{Setup} \\
input? = store\_field \\
op' = input? \\
prompt! = store\_field\_prompt \\
\hline
\end{array}
$$

These two operations are completed by the complementary *Accept* operations, *WriteMessageC* and *StoreFieldC* (below).

8.3.9 *Setup* operations

Under *Setup*, there are *SelectName*, *SelectItem*, *MenuEdit* and *DialogEdit* operations.

Select Patient (8.9.3, 184 – 185):

$$
\begin{array}{|l}
\hline
\textit{SelectPatientC} \\
\textit{SelectName} \\
\hline
\textit{Setup} \\
\textit{display} = \textit{select\_patient} \\
\textit{Continue} \\
\hline
\end{array}
$$

At this writing, *SelectPatient* is a *Continue* operation; the patient list remains on the screen[20].

SelectField (8.9.4, 186 – 189): There are three cases. The simplest case occurs during experiment mode, or when the chosen field has not yet been delivered today and the prescribed total dose and number of fractions has not yet been exceeded (8.9.4, 187).

$$
\begin{array}{|l}
\hline
\textit{NewFieldC} \\
\textit{SelectName} \\
\textit{ConsoleSession} \\
\hline
\textit{Setup} \\
\textit{display} = \textit{select\_field} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{SelectSimpleFieldC} \\
\textit{NewFieldC} \\
\hline
\textit{mode} = \textit{experiment} \\
\lor\ (\textit{counters}\ \textit{name}!\ \textit{dose} = 0 \land \neg\ \textit{exceeded}(\textit{fields}\ \textit{name}!, \textit{counters}\ \textit{name}!)) \\
\textit{Continue} \\
\hline
\end{array}
$$

The more complicated cases arise in therapy mode when the operator must be warned of some unusual condition (8.9.4, 187-188). These are *DialogOp* operations. The name of the new field must be stored during the dialog. The operator may enter a preset dose or cancel the dialog (so no new field is selected).

[20] We also considered establishing $\textit{display}' = \textit{select\_field}$ in *SelectPatientC*. It would not be difficult to adopt this alternative later.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad Console1 \quad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
ConsoleSession\\
new\_field : FIELD\\
\hline
new\_field \in \mathrm{dom}\, fields\\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad DoseDialogOp \quad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
NewFieldC\\
DialogOp\\
\Delta Console1\\
\hline
op' = select\_field\\
new\_field' = name!\\
mode = therapy\\
\hline
\end{array}
$$

There are two such cases. The first arises when the same field has already been delivered on the same day but the prescribed daily dose has not yet been reached; the remaining dose is offered as the default (8.9.4, 187):

$$
\begin{array}{|l}
sprintf : VALUE \longrightarrow STRING\\
delivered\_prompt : NAME \times VALUE \times VALUE \times VALUE \longrightarrow CAPTION
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad SelectDeliveredField \quad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
DoseDialogOp\\
\hline
counters\ new\_field'\ dose > 0\\
\neg\ exceeded(fields\ new\_field', counters\ new\_field')\\
(\mathbf{let}\ d == fields\ new\_field'\ dose;\ c == counters\ new\_field'\ dose \bullet\\
\qquad (\mathbf{let}\ default\_dose == d - c \bullet\\
\qquad\qquad buffer' = sprintf\ default\_dose \wedge\\
\qquad\qquad prompt! = delivered\_prompt(new\_field', d, c, default\_dose)))\\
\hline
\end{array}
$$

The other case arises when the daily dose, the total dose or the number of fractions has been exceeded (8.9.4, 188). No default dose is provided.

$$
\begin{array}{|l}
exceeded\_prompt : NAME \times ACCUMULATION \times ACCUMULATION \longrightarrow CAPTION
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad SelectExceededField \quad\qquad\qquad\qquad\qquad\qquad\qquad}}\\
DoseDialogOp\\
\hline
exceeded(fields\ new\_field', counters\ new\_field')\\
prompt! = exceeded\_prompt(new\_field', fields\ new\_field', counters\ new\_field')\\
buffer' = empty\\
\hline
\end{array}
$$

The complete operation is composed of all these cases.

$$SelectFieldC \mathrel{\widehat=} SelectSimpleFieldC \lor SelectDeliveredField \lor SelectExceededField$$

After $SelectSimpleFieldC$, nothing more need be done. $SelectExceededField$ and $SelectDeliveredField$ are succeeded by the $SelectFieldOp$ state, which is handled by the $SelectComplexFieldS$ operation.

$$\begin{array}{|l}
\multicolumn{1}{l}{SelectFieldOp} \\
\hline
Console1 \\
Dialog \\
\hline
op = select_field \\
\hline
\end{array}$$

Override (8.4, 175; 8.8.1, 181; 8.8.2, 183) is also a $ConfirmOp$ operation, enabled only when a field has been selected. The name of the item that the operator selected is echoed in the confirmation dialog. To begin **Override**:

$$override\_table == \{filter\_wedge, leaf\_collim, gantry\_psa, dose\_intlk\}$$

$$\begin{array}{|l}
override_query : CAPTION \longrightarrow CAPTION
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{SelectOverride} \\
\hline
ConfirmOp \\
SelectItem \\
\Xi Session \\
\hline
field \neq no_field \\
display \in override_table \\
input? = override_cmd \\
op' = input? \\
query! = override_query(setting_info_name\ item) \\
\hline
\end{array}$$

The complementary $AcceptConfirm$ operations is $OverrideC$ (below).

Edit: There are three cases. Calibration factors can be edited in experiment mode (8.9.13, 215), preset dose and time can be edited in both modes when a field is selected (8.9.11, 201–202), and other preset items can be edited in experiment mode when a field is selected (8.8.1, 180–181; 8.9.8, 194; 8.9.9, 196; 8.9.10, 198).

$$cal\_table == \{dose\_cal\}$$

$$dose\_table == \{dose\_intlk\}$$

$$preset\_table == \{filter\_wedge, leaf\_collim\}$$

$$CalTable \mathrel{\widehat=} [\, ConsoleSession \mid mode = experiment \land display \in cal\_table \,]$$

The precondition $field \neq no\_field$ occurs elsewhere so it is convenient to collect two cases together.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SettingTable}} \\
ConsoleSession \\
\hline
display \in dose_table \lor (mode = experiment \land display \in preset_table) \\
\hline
\end{array}$$

Finally

$$SelectCalMenu \mathrel{\widehat=} CalTable \land MenuEdit$$

$$SelectCalDialog \mathrel{\widehat=} CalTable \land DialogEdit$$

$$SelectSettingMenu \mathrel{\widehat=} SettingTable \land MenuEdit$$

$$SelectSettingDialog \mathrel{\widehat=} SettingTable \land DialogEdit$$

The complementary operations are *EditSettingC* and *MenuSettingC* (section 8.2, above).

8.3.10 *Cancel* operations

There is a special *cancel* operation for the login process (below) so we have to strengthen the preconditions on *Cancel*:

$$LoggedIn \mathrel{\widehat=} [Console \mid op \notin \{login, password\}]$$

$$CancelOp \mathrel{\widehat=} LoggedIn \land Cancel$$

8.3.11 *AcceptConfirm* operations

To complete **Override** (8.4, 175; 8.8.1, 181; 8.8.2, 183):

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{OverrideC}} \\
AcceptConfirm \\
item! : ITEM \\
\hline
op = override_cmd \\
item! = item \\
\hline
\end{array}$$

To complete **Cancel Run** (8.9.11, 209 – 210):

$$CancelRunC \mathrel{\widehat=} [\, AcceptConfirm \mid op = cancel\_run \,]$$

8.3.12 *Accept* operations

To complete **Write Log Message** (2.5.1, 17). The *smessage* function turns a string into a log message by prepending the timestamp and other information.

$$log\_msg : STRING \longrightarrow MESSAGE$$

$$
\begin{array}{|l}
\hline
\textit{WriteMessageC} \\
Accept \\
message! : MESSAGE \\
\hline
op = log\_message \\
message! = log\_msg\ buffer \\
\hline
\end{array}
$$

To complete **Store Field** (8.9.5, 189 – 188)[21].

$$
\begin{array}{|l}
sname : STRING \longrightarrow NAME \\
store\_msg : NAME \longrightarrow MESSAGE
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{StoreFieldC} \\
Accept \\
field! : NAME \\
message! : MESSAGE \\
\hline
op = store\_field \\
field! = sname\ buffer \\
message! = store\_msg\ field! \\
\hline
\end{array}
$$

The complex variants of **Select Field** (8.9.4, 187–188) are handled by *SelectComplexFieldC*, which is similar to *EditSettingC*:

$$
\begin{array}{|l}
\hline
\textit{SelectComplexFieldC} \\
\Delta Console1 \\
Accept \\
field! : FIELD \\
dose! : VALUE \\
\hline
SelectFieldOp \\
field! = new\_field \\
(\mathbf{let}\ d == sval\ buffer \bullet d \in valid\ dose \land dose! = d) \\
\hline
\end{array}
$$

[21] The *message*! output from *StoreFieldC* is not mentioned in [2]; here we correct the omission.

We have to make the operation total

$$InvalidDose \mathrel{\widehat=} [\, Reprompt;\ \Delta Console1 \mid SelectFieldOp \land sval\ buffer \notin valid\ dose \,]$$

$$ComplexOrInvalidField \mathrel{\widehat=} SelectComplexFieldC \lor InvalidDose$$

8.3.13 Logout and login

The **Login** process (Fig. 2.6, 19) can be seen as editing the value of *operator*. Messages are logged at logout and login. **Logout** (2.5.2, 17 – 18; 8.9.2, 184) is is similar to *SelectDisplay* and *DialogOp*; it reads *Session* for the operator's ID in the logout message.

$$o\_msg, lo\_msg : OPERATOR \longrightarrow MESSAGE$$

$$
\begin{array}{|l}
\textbf{Logout} \\
\hline
Op \\
\Xi Session \\
message! : MESSAGE \\
\hline
Setup \\
input? = login \\
display' = input? \\
op' = display' \\
buffer' = empty \\
message! = lo\_msg\ operator \\
Dialog' \\
\hline
\end{array}
$$

A successful **Login** (2.5.2, 17 – 20; 8.9.1, 183) occurs when a user enters a valid operator identification. The process of logging back in is broken into two steps. In the first step, the user types their username into the buffer. When the user types a terminator the username is saved in another buffer; the *Console*1 state is *Console* with this buffer added[22]. The console remains logged out after this step. This *EnterUsername* operation is similar to *Accept* and *Continue*.

$$
\begin{array}{|l}
\textbf{Console2} \\
\hline
Console \\
username : STRING \\
\hline
\end{array}
$$

[22]The password is also typed into *buffer* so the *GetChar* operation can be used.

$$
\begin{array}{|l}
\hline
\textit{EnterUsername} \\
\Delta Console2 \\
\Delta Dialog \\
EventUnlocked \\
\hline
op = login \\
input? \in terminator \\
username' = buffer \\
buffer' = empty \\
op' = password \\
display' = display \\
\hline
\end{array}
$$

In the second step, the user enters their password. If their username, password pair is found in operator database, the user is logged in, the help screen is displayed, and the console becomes available. This *LoginC* operation is similar to *Accept* and *SelectDisplay*.

The operator authorization file is modelled by Z global constants *operators* and *physicists* (physicists are authorized to use the equipment in its experiment mode, see 8.2, 170). Values of *OPERATOR* include the operator's password in addition to the operator's printed name.

$$USERNAME == STRING; PASSWORD == STRING$$

$$
\begin{array}{|l}
soper : (USERNAME \times PASSWORD) \longrightarrow OPERATOR
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{LoginC} \\
\Delta Console2 \\
EventUnlocked \\
operator! : OPERATOR \\
message! : MESSAGE \\
\hline
op = password \\
soper(username, buffer) \in operators \\
input? \in terminator \\
display' = help \\
op' = display \\
operator! = soper(username, buffer) \\
message! = o\_msg\ operator! \\
Available' \\
\hline
\end{array}
$$

If the username, password pair is not found in the authorization file, the console remains logged out.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Unauthorized}} \\
\hline
\Delta Console2 \\
Reprompt \\
\hline
op = password \\
soper(username, buffer) \notin operators \\
username' = username \\
\hline
\end{array}
$$

$$LoginOrUnauthorized \mathrel{\widehat=} LoginC \lor Unauthorized$$

Users may cancel a login attempt while entering their username or password. The login process begins anew.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{CancelUsername}} \\
\hline
\Delta Console2 \\
\Delta Dialog \\
EventUnlocked \\
\hline
op \in \{login, password\} \\
input? = cancel \\
op' = login \\
buffer' = empty \\
username' = empty \\
display' = display \\
\hline
\end{array}
$$

8.3.14 Other operations

Several operations not described in [2] are included for development purposes[23]. They are always enabled.

$$Refresh \mathrel{\widehat=} [\, Event \mid input? = refresh \,]$$
$$Shutdown \mathrel{\widehat=} [\, Event \mid input? = shutdown \,]$$

This concludes our presention of the states and operations in the user interface.

8.3.15 Summary

Table 5 shows all of the top-level operations that the user can invoke, in almost the same format as the building block operations in Table 4. Some building block operations also

[23] *Shutdown* will not be included in production versions.

appear in Table 5; new operations are followed in parentheses by the names of the building block operations they use.

8.4 Implementation

In [4] we describe how to implement a user interface specified in our style. The method requires that each operation be expressed as a conjunction of three separate schemas: a state schema for the preconditions involving the state variables, a state schema for the preconditions involving the input variable, and an operation schema. From the operation schemas we have already defined, we factor out the following schemas for states and inputs:

$$Physicist \mathrel{\widehat=} [ConsoleSession \mid operator \in physicists]$$

$$PatientList \mathrel{\widehat=} [Console \mid display = select\_patient \land nlist \neq \varnothing]$$

$$PatientSelected \mathrel{\widehat=} [ConsoleSession \mid patient \neq no\_patient]$$

$$FieldList \mathrel{\widehat=} [Console \mid display = select\_field \land nlist \neq \varnothing]$$

$$FieldSelected \mathrel{\widehat=} [ConsoleSession \mid field \neq no\_field]$$

$$AutoSetupDisplay \mathrel{\widehat=} [Console \mid display \in auto\_setup\_display]$$

$$OverrideTable \mathrel{\widehat=} [Console \mid display \in override\_table]$$

$$MenuItem \mathrel{\widehat=} [Console \mid item \in selection]$$

$$DialogItem \mathrel{\widehat=} [Console \mid item \notin selection]$$

$$LoggedOut \mathrel{\widehat=} [Console \mid op \in \{login, password\}]$$

$$OverrideOp \mathrel{\widehat=} [Console \mid op = override\_cmd]$$

$$CancelRunOp \mathrel{\widehat=} [Console \mid op = cancel\_run]$$

$$LogMessageOp \mathrel{\widehat=} [Console \mid op = log\_message]$$

$$StoreFieldOp \mathrel{\widehat=} [Console \mid op = store\_field]$$

$$UsernameOp \mathrel{\widehat=} [Console \mid op = login]$$

$$PasswordOp \mathrel{\widehat=} [Console \mid op = password]$$

$$Input \mathrel{\widehat=} [\, input? : INPUT \,]$$

$$DisplayKey \mathrel{\widehat=} [Input \mid input? \in simple\_display]$$

$$PatientKey \mathrel{\widehat=} [Input \mid input? = select\_patient]$$

$$FieldKey \mathrel{\widehat=} [Input \mid input? = select\_patient]$$

$TableKey \mathrel{\widehat=} [Input \mid input? \in table]$

$MessageKey \mathrel{\widehat=} [Input \mid input? = log\_message]$

$VArrowKey \mathrel{\widehat=} [Input \mid input? \in v\_arrow]$

$SelectKey \mathrel{\widehat=} [Input \mid input? = select]$

$ArrowKey \mathrel{\widehat=} [Input \mid input? \in arrow]$

$ExptModeKey \mathrel{\widehat=} [Input \mid input? = expt\_mode]$

$AutoSetupKey \mathrel{\widehat=} [Input \mid input? = auto\_setup]$

$StoreFieldKey \mathrel{\widehat=} [Input \mid input? = store\_field]$

$LoginKey \mathrel{\widehat=} [Input \mid input? = login]$

$OverrideKey \mathrel{\widehat=} [Input \mid input? = override\_cmd]$

$CancelRunKey \mathrel{\widehat=} [Input \mid input? = cancel\_run]$

$CancelKey \mathrel{\widehat=} [Input \mid input? = cancel]$

$CharKey \mathrel{\widehat=} [Input \mid input? \in CHAR]$

$TerminatorKey \mathrel{\widehat=} [Input \mid input? \in terminator]$

$RefreshKey \mathrel{\widehat=} [Input \mid input? = refresh]$

$ShutdownKey \mathrel{\widehat=} [Input \mid input? = shutdown]$

Table 6 (essentially Table 5 reformatted) expresses the entire user interface in the format required by our implementation method [4][24]. The table represents *ConsoleOp*: all the top level operations from Table 5 combined into a single operation.

$$ConsoleOp \mathrel{\widehat=} SelectDisplay \lor SelectPatientList \lor SelectFieldList \lor \ldots \lor IgnoreOthers$$

IgnoreOthers is the default do-nothing operation.

$$IgnoreOthers \mathrel{\widehat=} [Ignore \mid \ldots]$$

IgnoreOthers ensures that *ConsoleOp* is total; its precondition is the negation of the disjunction of the preconditions of all the other operations. We do not define this precondition explicitly; we code the implementation so control reaches *IgnoreOthers* when no other operations are enabled.

ConsoleOp defines a state transition system where each disjunct defines a single transition. Table 6 is the state transition table. There is an entry (row) in the table for each top-level operation schema. The first column names the state precondition, the second column names

[24] The LaTeX source for Table 6 and code skeletons for the implementation (in C) are generated automatically (by `sed` and `awk` scripts) from the same text file.

- *Refresh*
- *Shutdown*
- *Unlocked*
  - *Available*
    - *SelectDisplay (Op)*
    - *SelectPatientList (SelectList)*
    - *SelectTable (SelectDisplay)*
    - *TypeMessage (DialogOp)*
    - *List*
      - *GetListArrow*
    - *Table*
      - *GetSettingArrow*
    - *PatientSelected*
      - *SelectFieldList (SelectList)*
    - *Setup*
      - *Logout (*similar to *SelectDisplay, DialogOp)*
      - *ExptModeC (Op)*
      - *(List)*
        - *SelectPatientC (SelectName)*
        - *SelectFieldC (SelectName, DialogOp)*
      - *(PatientSelected)*
        - *EditField (DialogOp)*
      - *(CalTable)*
        - *SelectMenuItem (MenuEdit)*
        - *SelectDialogItem (DialogEdit)*
      - *(FieldSelected)*
        - *AutoSetupC (Op)*
        - *SelectOverride (SelectItem, ConfirmOp)*
        - *(SettingTable)*
          - *SelectMenuItem (MenuEdit)*
          - *SelectDialogItem (DialogEdit)*
    - *Running*
      - *SelectCancelRun (ConfirmOp)*
  - *Engaged*
    - *(LoggedIn)*
      - *CancelOp (Cancel)*
    - *Confirm*
      - *OverrideC (AcceptConfirm)*
      - *CancelRunC (AcceptConfirm)*
    - *Menu*
      - *GetMenuArrow*
      - *(Editing)*
        - *MenuSettingC (AcceptMenu)*
    - *Dialog*
      - *GetChar*
      - *WriteMessageC (Accept)*
      - *ComplexOrInvalidField (Accept* or *Reprompt)*
      - *StoreFieldC (Accept)*
      - *(Editing)*
        - *EditOrInvalidSetting (Accept* or *Reprompt)*
      - *(LoggedOut)*
        - *CancelUsername (*similar to *Cancel)*
        - *EnterUsername (*similar to *Accept, Continue)*
        - *LoginOrUnauthorized (*similar to *Accept* and *SelectDisplay*, or *Reprompt)*

Table 5: Therapy operations

	State precondition	Input precondition	Operation
0	*Z_True*	*RefreshKey*	*Refresh*
0	*Z_True*	*ShutdownKey*	*Shutdown*
0	*Unlocked*	*NoKey*	*NoOp*
1	*Available*	*DisplayKey*	*SelectDisplay*
1	.	*PatientKey*	*SelectPatientList*
1	.	*TableKey*	*SelectTable*
1	.	*MessageKey*	*TypeMessage*
2	*List*	*VArrowKey*	*GetListArrow*
2	*Table*	*ArrowKey*	*GetSettingArrow*
2	*PatientSelected*	*FieldKey*	*SelectFieldList*
2	*Setup*	*LoginKey*	*Logout*
3	*Physicist*	*ExptModeKey*	*ExptModeC*
3	*PatientList*	*SelectKey*	*SelectPatientC*
3	*FieldList*	*SelectKey*	*SelectFieldC*
3	*PatientSelected*	*StoreFieldKey*	*EditField*
3	*CalTable*	*NoKey*	*NoOp*
4	*MenuItem*	*SelectKey*	*SelectMenuItem*
4	*DialogItem*	*SelectKey*	*SelectDialogItem*
3	*FieldSelected*	*NoKey*	*NoOp*
4	*AutoSetupDisplay*	*AutoSetupKey*	*AutoSetupC*
4	*OverrideTable*	*OverrideKey*	*SelectOverride*
4	*SettingTable*	*NoKey*	*NoOp*
5	*MenuItem*	*SelectKey*	*SelectMenuItem*
5	*DialogItem*	*SelectKey*	*SelectDialogItem*
2	*Running*	*CancelRunKey*	*SelectCancelRun*
1	*Engaged*	*NoKey*	*NoOp*
2	*LoggedIn*	*CancelKey*	*CancelOp*
2	*Confirm*	*NoKey*	*NoOp*
4	*OverrideOp*	*SelectKey*	*OverrideC*
4	*CancelRunOp*	*SelectKey*	*CancelRunC*
2	*Menu*	*VArrowKey*	*GetMenuArrow*
3	*Editing*	*SelectKey*	*MenuSettingC*
2	*Dialog*	*CharKey*	*GetChar*
3	*LogMessageOp*	*TerminatorKey*	*WriteMessageC*
3	*SelectFieldOp*	*TerminatorKey*	*ComplexOrInvalidField*
3	*StoreFieldOp*	*TerminatorKey*	*StoreFieldC*
3	*Editing*	*TerminatorKey*	*EditOrInvalidSetting*
3	*LoggedOut*	*CancelKey*	*CancelUsername*
3	*UserNameOp*	*TerminatorKey*	*EnterUsername*
3	*PasswordOp*	*TerminatorKey*	*LoginOrUnauthorized*

Table 6: Therapy console state transition table

the input precondition, and the last column names the operation schema itself. In this table the operations appear in the same order as they do in Table 5.

In our table, sequence order and nesting level (indicated by indentation and the number in the first column) represent the nesting of states that is expressed in Z by schema inclusion. A greater nesting level indicates that a table entry is a *substate* of preceding entries at lesser nesting levels. *The full state precondition of a substate is formed by conjoining the state preconditions of the preceding entries at lesser nesting levels.* For example, the full state precondition for the *SelectPatientC* operation is $Unlocked \land Available \land Setup \land PatientList$. *When the first column in a row is blank, the substate is the same as the last preceding nonblank column at the same nesting level*: the precondition for the *Logout* operation is $Unlocked \land Available \land Setup$. Each line also applies to any included substates, so *DisplayKey* elicits the *SelectDisplay* operation in the *Available* state, and also in its substates *Setup* and *Running*, and in its sub-substates *PatientList* and *FieldList* etc.

The table requires these place holders.

$$True \mathrel{\widehat{=}} Console$$

$$NoOp \mathrel{\widehat{=}} \Xi Console$$

$$NoKey \mathrel{\widehat{=}} [Input \mid false]$$

9 Combining the subsystems

In this section we combine related operations from the *Console*, *Session* and *Field* subsystems. In cases where no data is transferred between subsystems, we can simply conjoin the separate operations:

$$ExptMode \mathrel{\widehat{=}} ExptModeC \land ExptModeS \land ExptModeF$$

Here the conjunction just expresses that the named operations in all three subsystems are triggered by the *expt_mode* input at the console. In this report both *ExptModeC* and *ExptModeS* contain the precondition $operator \in physicists$, but this predicate only needs to appear once; we include it in both schemas for clarity[25].

In other cases, data is transferred. For example, in the *Override* operation, the $item!$ output from the *Console* subsystem is consumed by the $item?$ input in the *Field* subsystem (the *Session* subsystem does not participate in this operation). We wish to express

[25] In our implementation, we observe the convention that preconditions of combined operations are always tested in the *Console* operations. For example $operator \in physicists$ is tested by code in `zconsole.c`, not `zsession.c`.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Override}} \\
\hline
\textit{OverrideC} \\
\textit{OverrideF} \\
\hline
\textit{item}! = \textit{item}? \\
\hline
\end{array}
$$

This can be expressed more concisely using the Z *pipe* operator *pipe*, which has the effect of connecting corresponding input and output variables ([7], p. 78)[26].

$$\textit{Override} \mathrel{\widehat=} \textit{OverrideC} \gg \textit{OverrideF}$$

In *Login*, the new operator name $\textit{operator}!/\textit{operator}?$ is the piped variable.

$$\textit{Login} \mathrel{\widehat=} \textit{LoginC} \gg \textit{LoginS}$$

In *EditSetting*, both the item name $\textit{item}!/\textit{item}?$ and the new setting value $\textit{value}!/\textit{value}?$ are piped:

$$\textit{EditSetting} \mathrel{\widehat=} \textit{EditSettingC} \gg \textit{EditSettingF}$$

Sometimes the output from the *Console* subsystem is piped to inputs in both the *Session* and *Field* subsystems. In *StoreField*, the new field name $\textit{field}!/\textit{field}?$ is the piped variable:

$$\textit{StoreField} \mathrel{\widehat=} \textit{StoreFieldC} \gg (\textit{StoreFieldS} \land \textit{StoreFieldF})$$

Here the conjunction $\textit{StoreFieldS} \land \textit{StoreFieldF}$ ensures that $\textit{prescribed}'$ in $\textit{fields}' = \textit{fields} \cup \{\textit{field}' \mapsto \textit{prescribed}'\}$ from *StoreFieldS* is the same as $\textit{prescribed}'$ in $\textit{prescribed}' = \textit{prescribed} \oplus \textit{measured} \ldots$ from *StoreFieldF*.

In *SelectPatient*, the new patient name $\textit{patient}!/\textit{patient}?$ is the piped variable ($\textit{patient}?$ does not appear in *SelectPatientF*).

$$\textit{SelectPatient} \mathrel{\widehat=} \textit{SelectPatientC} \gg (\textit{SelectPatientS} \land \textit{SelectPatientF})$$

There are two variations of *SelectField*. In both variants $\textit{field}!/\textit{field}?$ is piped from the *SelectFieldC* variant to *SelectFieldS*. In the complex variant, $\textit{dose}!/\textit{dose}?$ is piped from *SelectComplexFieldS* to *SelectComplexFieldF*.

$$\textit{SelectSimpleField} \mathrel{\widehat=} \textit{SelectSimpleFieldC} \gg (\textit{SelectFieldS} \land \textit{SelectSimpleFieldF})$$

$$\textit{SelectComplexField} \mathrel{\widehat=} \textit{SelectComplexFieldC} \gg (\textit{SelectFieldS} \land \textit{SelectComplexFieldF})$$

[26] The Z pipe operator also hides the piped variables.

References

[1] 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.

[2] 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.

[3] Jonathan Jacky and Jonathan Unger. Formal specification of control software for a radiation therapy machine. Technical Report 94-07-01, Radiation Oncology Department, University of Washington, Seattle, WA, July 1994.

[4] Jonathan Jacky and Jonathan Unger. From Z to code: A graphical user interface for a radiation therapy machine. In J. P. Bowen and M. G. Hinchey, editors, *ZUM '95: The Z Formal Specification Notation*, pages 315 – 333. Ninth International Conference of Z Users, Springer-Verlag, 1995. Lecture Notes in Computer Science 967.

[5] Adrian Nye. *Xlib Programming Manual.* O'Reilly and Associates, Inc., Sebastopol, CA, 1988.

[6] J. M. Spivey. *The fUZZ Manual.* J. M. Spivey Computing Science Consultancy, Oxford, second edition, July 1992.

[7] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, New York, second edition, 1992.

A Glossary of items

A.1 Settings

These are the elements of *setting*:

collim Collimator rotation angle (8.9.8, Fig. 8.5, 193 – 194).

dose Dose per fraction. The *prescribed* value is the dose prescribed to be delivered from the current field in a single fraction. It is read from the prescription file, and appears in the MU column of the field list display (Fig 8.3, 186) and the PRESCR column of the dosimetry display (Fig. 8.8, 199 etc.). The same *prescribed* values appear in both the A and B rows of the dosimetry display. The *measured* value is the dose accumulated in DMC channel A since it was last reset, in monitor units. It appears in the ACCUM column in the dosimetry display. The *accumulated* value is the total dose accumulated in all treatment runs since the beginning of the day, in monitor units (8.9.4, 187 bottom half – 188 top half). See also the *dose_reg* element *p_dose*, below.

doseB Dose accumulated in DMC channel B since it was last reset, in monitor units. This backup to channel A is used internally by the DMC but its only role in our control program is to be displayed in the ACCUM column in the dosimetry display.

dose_tot Total dose intended to be delivered from the current field over the entire course of treatment, in monitor units. The *prescribed* value is from the prescription file, and appears in the Total column of the field list display (Fig 8.3, 186). The *accumulated* value is the dose actually delivered to date, in monitor units (8.9.4, Fig 8.3, 186; 187 bottom half – 188 top half), and appears in the last To date column on the field list display (Fig 8.3, 186).

filter Flattening filter selection (no_filter, small field, large field) (8.9.9, Fig. 8.6, 194 – 196).

gantry Gantry rotation rotation angle (8.9.8, Fig. 8.5, 193 – 194).

height Couch height position. Table positions are not included in determining *ready* status, but may be stored in prescription file (8.9.8, Fig. 8.5, 193 – 194).

lat Couch lateral position (8.9.8, Fig. 8.5, 193 – 194).

leaf 0, *leaf* 39 Collimator leaf positions (8.9.10, Fig. 8.7, 196 – 198). There are actually forty leaves, not just two, but for brevity we omit *leaf* 1 . . *leaf* 38 from the formal description. Our *leaf* 0 has the same *valid* values as *leaf* 1 . . *leaf* 19, and *leaf* 39 has the same *valid* values as *leaf* 20 . . *leaf* 38.

longit Couch longitudinal position (8.9.8, Fig. 8.5, 193 – 194).

nfrac Intended total number of fractions for this field, over the entire course of treatment (8.9.4, 188). The *prescribed* value appears in the first Fractions column on the field list display (Fig 8.3, 186). The *accumulated* value is the number of fractions accumulated for this field to date (8.9.4, 187 – 188), and appears in the first To date column on the field list display (Fig 8.3, 186).

top Couch top swivel rotation (8.9.8, Fig. 8.5, 193 – 194).

turnt Turntable rotation (8.9.8, Fig. 8.5, 193 – 194).

w_rot Wedge rotation (8.9.9, Fig. 8.6, 194 – 199).

wedge Wedge selection (8.9.9, Fig. 8.6, 194 – 199).

A.2 Registers

These are the elements of *dose_reg*:

*calvolt*1 DMC standard calibration voltage. The *calibrated* value is read from a file, while the *computed* value is computed from the *calibrated* value by adjusting by a pressure-temperature correction factor (8.9.13, 213 – 214). The *computed* value is actually loaded into the DMC before each run.

*calvolt*2 DMC standard calibration voltage, etc.

d_rate Nominal dose rate, "dose rate of the day," used to calculate treatment backup time, *p_time* (8.9.11, 200, last paragraph, 202, second paragraph; 8.9.13, 213, first two paragraphs after bullets). The *calibrated* value is read from a file; the *computed* value is initialized to the same value but may be edited by the operator.

e_time Elapsed time. The *calibrated e_time* is the beam-on time elapsed since the DMC was last reset, and appears in ACCUM column in dosimetry display (8.9.11, Fig. 8.8, 199 etc.)

p_dose Preset dose loaded into the DMC at the beginning of each treatment run. The *computed* value is computed by the control program when the field is selected and is usually set equal to the *prescribed dose*, except (for example) after an interrupted treatment run. Subsequently it may be edited by the operator (8.9.4, 187 - 188; 8.9.11, 201 – 202). It appears in the PRESET column of the dosimetry display. The same *computed* values appear in both the A and B rows of the dosimetry display. See also the *setting* elements *dose* and *doseB*.

p_time Preset time, the treatment backup time. The *calibrated p_time* is initially calculated from preset dose *p_dose*, dose rate *d_rate*, and time factor *t_fac* (8.9.11, 202; 8.9.13, 213); it appears in PRESCR column in dosimetry display (Fig 8.8, 199 etc.). The *computed p_time* is initially set equal to *calibrated p_time* but may be edited by the operator (8.9.11, 202; 8.9.13, 213). The *computed p_time* is actually loaded into the DMC before each run.

press Barometric pressure used to compute pressure/temperature correction factor. The *calibrated* value is measured continuously, while the *computed* value is entered by the operator (8.9.13, 213 – 214).

pt_factor Pressure/temperature correction factor used to compute *computed calvolt*1 and *computed calvolt*2 (8.9.13, 214). The *calibrated* value is computed from *calibrated press* and *calibrated tempo* measured from sensors, while the *computed* value is computed from *computed press* and *computed tempo* entered by the operator.

pt_mode Pressure/temperator correction factor selection (either automatic or manual) (8.9.13, 214). In *automatic* mode, *computed calvolt*1 and *computed calvolt*2 are calculated from *calibrated pt_factor* derived from sensor readings, while in *manual* mode they are calculated from the *computed pt_factor* derived from values entered by the operator.

t_fac Treatment time factor (8.9.11, 200, last paragraph, 202, second paragraph; 8.9.13, 213, first two paragraphs after bullets). The *calibrated* value is read from a file; the *computed* value is initialized to the same value but may be edited by the operator.

temp Temperature used to compute pressure/temperature correction factor. The *calibrated* value is measured by sensors, while the *computed* value is entered by the operator (8.9.13, 213 – 214).

B Groups of items

cal_const Calibration constants stored in files: *d_rate*, *t_fac*, *calvolt*1 and *calvolt*2 (8.9.13, 213).

counter Items whose actual values increase during a treatment run (such as *dose*) or over the entire course of treatment (such as the number of fractions *nfrac* and total dose *dose_tot*). Domain of *accumulated* in *Field*. Counters in *prescr* can only be *ready* when their *accumulated* values are less than their *prescribed* values. The only three counters are *dose*, *n* and *dose_tot*.

dose_reg Calibration factors and other items related to the dosimetry system which are not stored prescription files. Some are stored in calibration files, others are computed or entered by the operator. Domain of *calibrated* and *computed* in *Field*. They are *pt_mode*, *pt_factor*, *press*, *temp*, *d_rate*, *t_fac*, *calvolt*1, *calvolt*2, *p_dose*, *p_time*, and *e_time*.

motion Settings that represent internal or external motions (for example, wedge rotation and gantry rotation, respectively). Each motion is either enabled or disabled (8.5, bottom 176 – top 177). The domain of *drive* in *Intlk*. They are all the *leaves* including *leaf*0 and *leaf*39, the filter settings *wedge*, *w_rot*, *filter*, the rotations *gantry*, *collim*, *turnt*, and the linear table motions *height*, *lat* and *long*.

prescrip Settings whose values are read from the prescription file for the selected patient and field. Subsequently, some may be edited by the operator. The domain of *prescribed* in *Field*. They are all the motions and all the counters (8.2, 171, third bullet).

prescr Settings whose prescribed values are checked against measured values in therapy mode. The domain of *status* in the *Intlk* schema. Same as *prescrip* except it excludes the three linear table motions, *height*, *lat* and *long*.

preset Settings whose values are read from the file of presets for the selected experiment field. They include only the leaves and the filter settings, omitting the external motions and the counters.

scale Items that vary continuously over some range, such as leaf position or gantry rotation. Scales in *prescr* are *ready* when their *measured* values lie within tolerance of their *prescribed* values. The domain of the global function *tol*. They include all the motions except the filter settings, and all the registers except the pressure-temperature factor correction mode.

selection Items that only take on discrete values, such as flattening filter selection. Selections are *ready* only when their *measured* values are exactly equal to their *prescribed* values. They are the three filter selections *filter*, *wedge*, and *w_rot* and the pressure-temperature factor correction mode *pt_mode*.

sensor Settings whose values are measured by sensors. Domain of function *measured* in *Field* schema. They are all the settings except the two counters *nfrac* and *dose_tot*.

setting All the items stored in the prescription database, with *doseB* and *top* as well.

This information is summarized in Table 7.

ITEM	*setting*	*d_reg.*	*cal.*	*prescrip*	*prescr*	*preset*	*motion*	*scale*	*sel.*	*counter*
nfrac	•			•	•					•
dose_tot	•			•	•					•
dose	•			•	•					•
wedge	•			•	•	•	•		•	
w_rot	•			•	•	•	•		•	
filter	•			•	•	•	•		•	
*leaf*0	•			•	•	•	•	•		
*leaf*39	•			•	•	•	•	•		
gantry	•			•	•		•	•		
collim	•			•	•		•	•		
turnt	•			•	•		•	•		
lat	•			•			•	•		
longit	•			•			•	•		
height	•			•			•	•		
doseB	•									
top	•									
pt_mode		•							•	
pt_factor		•								
press		•								
temp		•								
d_rate		•	•							
t_fac		•	•							
*calvolt*1		•	•							
*calvolt*2		•	•							
p_dose		•								
p_time		•								
e_time		•								

Table 7: Groups of items

C Types and constants

This appendix collects together the types and constants that define the system configuration.

C.1 Settings and registers

From section 3.1.

$$
\begin{aligned}
ITEM ::= {} & nfrac \mid dose\_tot \mid dose \mid wedge \mid w\_rot \mid filter \mid leaf0 \mid leaf39 \mid \\
& gantry \mid collim \mid turnt \mid lat \mid longit \mid height \mid doseB \mid top \mid \\
& pt\_mode \mid pt\_factor \mid press \mid temp \mid d\_rate \mid t\_fac \mid \\
& calvolt1 \mid calvolt2 \mid p\_dose \mid p\_time \mid e\_time
\end{aligned}
$$

$$
\begin{array}{|l}
setting, dose\_reg : \mathbb{P}\, ITEM \\
\hline
\langle setting, dose\_reg \rangle \ \mathsf{partition}\ ITEM \\
dose\_reg = \{pt\_mode, pt\_factor, press, temp, d\_rate, t\_fac, \\
\qquad calvolt1, calvolt2, p\_dose, p\_time, e\_time\}
\end{array}
$$

$$
\begin{array}{|l}
scale, selection, counter : \mathbb{P}\, ITEM \\
\hline
\langle selection, scale, counter \rangle \ \mathsf{partition}\ ITEM \\
counter = \{nfrac, dose\_tot, dose\} \\
selection = \{wedge, w\_rot, filter, pt\_mode\}
\end{array}
$$

$$leaves == \{leaf0, leaf39\}$$

$$preset == leaves \cup \{wedge, w\_rot, filter\}$$

$$motion == preset \cup \{gantry, collim, turnt, lat, longit, height\}$$

$$prescrip == motion \cup counter$$

$$prescr == prescrip \setminus \{lat, longit, height\}$$

$$sensor == setting \setminus \{nfrac, dose\_tot\}$$

$$cal\_const == \{d\_rate, t\_fac, calvolt1, calvolt2\}$$

C.2 Values

From section 3.2.

$$VALUE == \mathbb{Z}$$

$$\begin{array}{|l}
blank : VALUE \\
tol : scale \longrightarrow VALUE \\
valid : ITEM \longrightarrow \mathbb{P}\, VALUE \\
\hline
\forall\, s : ITEM \bullet blank \notin valid\ s
\end{array}$$

C.3 Prescription database

From section 4.

$$[NAME]$$

$$PATIENT == NAME;\ FIELD == NAME$$

$$\begin{array}{|l}
no_name : NAME
\end{array}$$

$$no\_patient == no\_name;\ no\_field == no\_name$$

$$\begin{array}{|l}
studies, patients : \mathbb{P}\, PATIENT \\
\hline
no_patient \notin studies \land no_patient \notin patients
\end{array}$$

$$ACCUMULATION == counter \longrightarrow VALUE$$

$$PRESCRIPTION == prescrip \longrightarrow VALUE$$

$$\begin{array}{|l}
Preset : studies \longrightarrow (FIELD \pfun PRESCRIPTION) \\
Prescribed : patients \longrightarrow (FIELD \pfun PRESCRIPTION) \\
Accumulated : patients \longrightarrow (FIELD \pfun ACCUMULATION) \\
\hline
\forall\, s : studies \bullet no_field \notin \mathrm{dom}(Preset\ s) \\
\forall\, p : patients \bullet no_field \notin \mathrm{dom}(Prescribed\ p) \land \mathrm{dom}\,(Prescribed\ p) = \mathrm{dom}\,(Accumulated\ p)
\end{array}$$

$$\begin{array}{|l}
exceeded_ : \mathbb{P}(ACCUMULATION \times PRESCRIPTION) \\
\hline
\forall\, counters : FIELD \pfun ACCUMULATION;\ fields : FIELD \pfun PRESCRIPTION \bullet \\
\qquad exceeded(counters, fields) \Leftrightarrow (\exists\, c : counter \bullet counters\ c \geq fields\ c)
\end{array}$$

C.3.1 Operators

From section 4.1.

$$[OPERATOR]$$

$$\begin{array}{|l} operators, physicists : \mathbb{P}\ OPERATOR \\ \hline physicists \subseteq operators \end{array}$$

C.4 Session

From section 5.

$$MODE ::= therapy \mid experiment$$

C.5 Field

From section 6.

$$\begin{array}{|l} automatic, manual : VALUE \end{array}$$

$$\begin{array}{|l} cal\_factor : cal\_const \longrightarrow VALUE \end{array}$$

$$PRESSURE == VALUE; TEMPERATURE == VALUE$$

$$DOSE == VALUE; RATE == VALUE; FACTOR == VALUE; TIME == VALUE$$

$$\begin{array}{|l} t\_backup : (DOSE \times RATE \times FACTOR) \nrightarrow TIME \\ pt\_formula : (PRESSURE \times TEMPERATURE) \nrightarrow FACTOR \end{array}$$

C.6 User interface

From section 8.

$$[CAPTION, MESSAGE]$$

$$alert : CAPTION$$
$$ocaption : OP \longrightarrow CAPTION$$

$$RUN ::= setup \mid running$$

$$KEYSWITCH ::= locked \mid unlocked$$

$$INTERACTION ::= available \mid dialog \mid menu \mid confirm$$

$$\begin{aligned} INPUT ::= {} & filter\_wedge \mid leaf\_collim \mid dose\_intlk \mid gantry\_psa \mid dose\_cal \mid \\ & startup \mid help \mid messages \mid select\_patient \mid select\_field \mid field\_summary \mid \\ & login \mid edit\_setting \mid edit\_dose\_reg \mid log\_message \mid store\_field \mid override\_cmd \mid \\ & cancel\_run \mid password \mid auto\_setup \mid expt\_mode \mid cancel \mid refresh \mid shutdown \mid \\ & select \mid ret \mid character \mid backspace \mid delete\_key \mid \\ & left\_arrow \mid right\_arrow \mid up\_arrow \mid down\_arrow \mid ignored \end{aligned}$$

$$OP : \mathbb{P}\, INPUT$$

$$\begin{aligned} OP = \{ & filter\_wedge, leaf\_collim, dose\_intlk, gantry\_psa, dose\_cal, \\ & startup, help, messages, select\_patient, select\_field, field\_summary, \\ & login, edit\_setting, edit\_dose\_reg, log\_message, store\_field, override\_cmd, cancel\_run, \\ & password, auto\_setup, expt\_mode, cancel, refresh, shutdown, select\} \end{aligned}$$

$$DISPLAY : \mathbb{P}\, OP$$

$$\begin{aligned} DISPLAY = \{ & filter\_wedge, leaf\_collim, dose\_intlk, gantry\_psa, dose\_cal, \\ & startup, help, messages, select\_patient, select\_field, \\ & field\_summary, login\} \end{aligned}$$

$$list, table : \mathbb{P}\, DISPLAY$$

$$\begin{array}{l} default\_item : table \longrightarrow ITEM \\ table\_items : table \longrightarrow \mathbb{P}\, ITEM \\ setting\_table, dose\_reg\_table : \mathbb{P}\, table \\ \hline \forall\, d : table \bullet default\_item\ d \in table\_items\ d \\ \forall\, d : setting\_table \bullet table\_items\ d \subseteq setting \\ \forall\, d : dose\_reg\_table \bullet table\_items\ d \subseteq dose\_reg \end{array}$$

$$v\_arrow == \{up\_arrow, down\_arrow\}$$

$$arrow == \{right\_arrow, left\_arrow\} \cup v\_arrow$$

$$\begin{array}{l} asetting : (arrow \times ITEM \times table) \longrightarrow ITEM \\ aname : (v\_arrow \times NAME \times \mathbb{P}_1\, NAME) \longrightarrow NAME \\ \hline \forall\, a : arrow;\ s : ITEM;\ d : table \bullet asetting(a, s, d) \in table\_items\ d \\ \forall\, a : v\_arrow;\ n : NAME;\ list : \mathbb{P}_1\, NAME \bullet aname(a, n, list) \in list \end{array}$$

$$nmax : \mathbb{N}$$

$$SELECTION == \{i : \mathbb{N} \mid i \leq nmax\}$$

$$default\_selection : SELECTION$$

$$MIN == VALUE;\ MAX == VALUE$$

$$\begin{array}{l} setting\_info\_name : ITEM \longrightarrow CAPTION \\ setting\_value : selection \longrightarrow \mathrm{iseq}_1\, CAPTION \\ setting\_info : ITEM \times MIN \times MAX \longrightarrow CAPTION \\ \hline \forall\, s : selection \bullet \mathrm{dom}\,(setting\_value\ s) = valid\ s \end{array}$$

$$[STRING]$$

$$\begin{array}{l} empty : STRING \\ CHAR : \mathbb{P}\, INPUT \\ terminator : \mathbb{P}\, INPUT \\ sprintf : VALUE \longrightarrow STRING \end{array}$$

$$modify : (STRING \times CHAR) \longrightarrow STRING$$

C.6.1 Therapy console operations

From section 8.3.

$$
\begin{array}{|l}
log\_msg : STRING \longrightarrow MESSAGE \\
o\_msg, lo\_msg : OPERATOR \longrightarrow MESSAGE \\
selected\_msg, store\_msg : NAME \longrightarrow MESSAGE
\end{array}
$$

$$
\begin{array}{|l}
cancel\_run\_query : CAPTION \\
override\_query : CAPTION \longrightarrow CAPTION \\
type\_message\_prompt, store\_field\_prompt : CAPTION \\
delivered\_prompt : NAME \times VALUE \times VALUE \times VALUE \longrightarrow CAPTION \\
exceeded\_prompt : NAME \times ACCUMULATION \times ACCUMULATION \longrightarrow CAPTION
\end{array}
$$

$$USERNAME == STRING; PASSWORD == STRING$$

$$
\begin{array}{|l}
soper : (USERNAME \times PASSWORD) \longrightarrow OPERATOR
\end{array}
$$

$$list = \{select\_patient, select\_field\}$$

$$table = \{gantry\_psa, filter\_wedge, leaf\_collim, dose\_intlk, dose\_cal\}$$

$$setting\_table = \{gantry\_psa, filter\_wedge, leaf\_collim\}$$

$$dose\_reg\_table = \{dose\_intlk, dose\_cal\}$$

$$
\begin{aligned}
table\_items = \{&gantry\_psa \mapsto \{gantry, collim, turnt\}, filter\_wedge \mapsto \{filter, wedge, w\_rot\}, \\
&leaf\_collim \mapsto leaves, dose\_intlk \mapsto \{p\_dose, p\_time\}, \\
&dose\_cal \mapsto \{pt\_mode, press, temp, d\_rate, t\_fac\}\}
\end{aligned}
$$

$$simple\_display == \{field\_summary, help\}$$

$$auto\_setup\_display == \{field\_summary, filter\_wedge, leaf\_collim, dose\_intlk\}$$

$$override\_table == \{filter\_wedge, leaf\_collim, gantry\_psa, dose\_intlk\}$$

$$cal\_table == \{dose\_cal\}$$

$$dose\_table == \{dose\_intlk\}$$

$$preset\_table == \{filter\_wedge, leaf\_collim\}$$

D States and invariants

D.1 Session

From section 5.

$$
\begin{array}{|l}
\hline
\textit{SessionVars} \\
\hline
\textit{mode} : \textit{MODE} \\
\textit{operator} : \textit{OPERATOR} \\
\textit{patient} : \textit{PATIENT} \\
\textit{field} : \textit{FIELD} \\
\textit{names} : \mathbb{P}\,\textit{PATIENT} \\
\textit{fields} : \textit{FIELD} \mathrel{+\!\!\!\to} \textit{PRESCRIPTION} \\
\textit{counters} : \textit{FIELD} \mathrel{+\!\!\!\to} \textit{ACCUMULATION} \\
\hline
\textit{operator} = \textit{no\_operator} \lor \textit{operator} \in \textit{operators} \\
\textit{mode} = \textit{experiment} \Rightarrow \textit{operator} \in \textit{physicists} \\
\textit{names} = \mathbf{if}\ \textit{mode} = \textit{therapy}\ \mathbf{then}\ \textit{patients}\ \mathbf{else}\ \textit{studies} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{NoPatient} \\
\hline
\textit{SessionVars} \\
\hline
\textit{patient} = \textit{no\_patient} \\
\textit{field} = \textit{no\_field} \\
\textit{fields} = \varnothing \\
\textit{counters} = \varnothing \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{PrescribedPatient} \\
\hline
\textit{SessionVars} \\
\hline
\textit{patient} \neq \textit{no\_patient} \\
\textit{patient} \in \textit{names} \\
\textit{field} = \textit{no\_field} \lor \textit{field} \in \operatorname{dom} \textit{fields} \\
\textit{fields} = \mathbf{if}\ \textit{mode} = \textit{therapy}\ \mathbf{then}\ \textit{Prescribed}\ \textit{patient}\ \mathbf{else}\ \textit{Preset}\ \textit{patient} \\
\textit{mode} = \textit{therapy} \Rightarrow \textit{counters} = \textit{Accumulated}\ \textit{patient} \\
\hline
\end{array}
$$

$$\textit{Session} \mathrel{\widehat{=}} \textit{PrescribedPatient} \lor \textit{NoPatient}$$

$$
\begin{array}{|l}
\hline
\textit{InitSession} \\
\textit{NoPatient} \\
\hline
mode = therapy \\
operator = no\_operator \\
\hline
\end{array}
$$

D.2 Field

From section 6.

$$
\begin{array}{|l}
\hline
\textit{Field} \\
prescribed : PRESCRIPTION \\
accumulated : ACCUMULATION \\
measured : sensor \longrightarrow VALUE \\
overridden : prescr \nrightarrow VALUE \\
computed, calibrated : dose\_reg \longrightarrow VALUE \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{PrescribedField} \\
\textit{Field} \\
\textit{Session} \\
\hline
field \neq no\_field \\
mode = therapy \Rightarrow prescribed = fields\ field \\
\hline
\end{array}
$$

$$no\_prescrip == (\lambda\, p : prescrip \bullet blank)$$

$$no\_counter == (\lambda\, c : counter \bullet blank)$$

$$no\_dose\_reg == (\lambda\, d : dose\_reg \bullet blank)$$

$$no\_dose == \{p\_dose \mapsto blank, p\_time \mapsto blank\}$$

$$
\begin{array}{|l}
\hline
\textit{NoFieldF} \\
\textit{Field} \\
\hline
prescribed = no\_prescrip \\
accumulated = no\_counter \\
no\_dose \subseteq computed \\
overridden = \varnothing \\
\hline
\end{array}
$$

$NoFieldS \mathrel{\widehat=} [Session \mid field = no\_field]$

$NoField \mathrel{\widehat=} NoFieldF \land NoFieldS$

$FieldSession \mathrel{\widehat=} PrescribedField \lor NoField$

$$
\begin{array}{|l}
\hline
\textit{InitField} \\
\hline
NoFieldF \\
\hline
computed = calibrated = no\_dose\_reg \oplus cal\_factor \\
\hline
\end{array}
$$

E User interface

From section 8.

$$
\begin{array}{|l}
\hline
\textit{Console} \\
\hline
keyswitch : KEYSWITCH \\
run : RUN \\
display : DISPLAY \\
op : OP \\
interaction : INTERACTION \\
item : ITEM \\
nlist : \mathbb{P}\ NAME \\
list\_item : NAME \\
menu\_item : SELECTION \\
buffer : STRING \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{InitConsole} \\
\hline
Console \\
\hline
op = login \\
display = login \\
interaction = dialog \\
buffer = empty \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{ConsoleSession} \\
\hline
Console \\
Session \\
\hline
display = select\_patient \Rightarrow nlist = names \\
display = select\_field \Rightarrow nlist = \operatorname{dom} fields \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{Console}1} \\
\mathit{Console} \\
\mathit{new\_field} : \mathit{FIELD} \\
\hline
\mathit{new\_field} \in \operatorname{dom} \mathit{fields} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{Console}2} \\
\mathit{Console} \\
\mathit{username} : \mathit{STRING} \\
\hline
\end{array}
$$

F Reference material

Field	*field*
Fractions	*prescribed n*
To date	*accumulated n*
MU	*prescribed dose*
Total	*prescribed dose_tot*
Expected	*accumulated n* $*$ *prescribed dose*
To date	*accumulated dose_tot*

Table 8: Settings and values in the field list display

SETTING	PRESCR	PRESET	ACCUM
DOSE A	*prescribed dose*	*computed p_dose*	*measured dose*
DOSE B	*prescribed dose*	*computed p_dose*	*measured doseB*
TIME	*calibrated p_time*	*computed p_time*	*calibrated e_time*

Table 9: Settings and values in the dosimetry display

	MEASURED/CALIBRATED	ADJUSTED
P/T MODE	*computed pt_mode (automatic/manual)*	
PRESSURE	*calibrated press*	*computed press*
TEMPERATURE	*calibrated temp*	*computed temp*
P/T CORR.	*calibrated pt_factor*	*computed pt_factor*
CALVOLT 1	*calibrated calvolt1*	*computed calvolt1*
CALVOLT 2	*calibrated calvolt2*	*computed calvolt2*
DOSE RATE	*calibrated d_rate*	*computed d_rate*
TIME FACTOR	*calibrated t_fac*	*computed t_fac*

Table 10: Dosimetry calibration display

	State precondition	Input precondition	Operation
0	*Z_True*	*RefreshKey*	*Refresh*
0	*Z_True*	*ShutdownKey*	*Shutdown*
0	*Unlocked*	*NoKey*	*NoOp*
1	*Available*	*DisplayKey*	*SelectDisplay*
1	.	*PatientKey*	*SelectPatientList*
1	.	*TableKey*	*SelectTable*
1	.	*MessageKey*	*TypeMessage*
2	*List*	*VArrowKey*	*GetListArrow*
2	*Table*	*ArrowKey*	*GetSettingArrow*
2	*PatientSelected*	*FieldKey*	*SelectFieldList*
2	*Setup*	*LoginKey*	*Logout*
3	*Physicist*	*ExptModeKey*	*ExptModeC*
3	*PatientList*	*SelectKey*	*SelectPatientC*
3	*FieldList*	*SelectKey*	*SelectFieldC*
3	*PatientSelected*	*StoreFieldKey*	*EditField*
3	*CalTable*	*NoKey*	*NoOp*
4	*MenuItem*	*SelectKey*	*SelectMenuItem*
4	*DialogItem*	*SelectKey*	*SelectDialogItem*
3	*FieldSelected*	*NoKey*	*NoOp*
4	*AutoSetupDisplay*	*AutoSetupKey*	*AutoSetupC*
4	*OverrideTable*	*OverrideKey*	*SelectOverride*
4	*SettingTable*	*NoKey*	*NoOp*
5	*MenuItem*	*SelectKey*	*SelectMenuItem*
5	*DialogItem*	*SelectKey*	*SelectDialogItem*
2	*Running*	*CancelRunKey*	*SelectCancelRun*
1	*Engaged*	*NoKey*	*NoOp*
2	*LoggedIn*	*CancelKey*	*CancelOp*
2	*Confirm*	*NoKey*	*NoOp*
4	*OverrideOp*	*SelectKey*	*OverrideC*
4	*CancelRunOp*	*SelectKey*	*CancelRunC*
2	*Menu*	*VArrowKey*	*GetMenuArrow*
3	*Editing*	*SelectKey*	*MenuSettingC*
2	*Dialog*	*CharKey*	*GetChar*
3	*LogMessageOp*	*TerminatorKey*	*WriteMessageC*
3	*SelectFieldOp*	*TerminatorKey*	*ComplexOrInvalidField*
3	*StoreFieldOp*	*TerminatorKey*	*StoreFieldC*
3	*Editing*	*TerminatorKey*	*EditOrInvalidSetting*
3	*LoggedOut*	*CancelKey*	*CancelUsername*
3	*UserNameOp*	*TerminatorKey*	*EnterUsername*
3	*PasswordOp*	*TerminatorKey*	*LoginOrUnauthorized*

Table 11: Therapy console state transition table