A Control System for a Radiation Therapy Machine

Jonathan Jacky*, Ruedi Risler, David Reid,
Robert Emery, Jonathan Unger†, Michael Patrick‡

Radiation Oncology, Box 356043
University of Washington
Seattle, WA 98195-6043

Technical Report 2001-05-01

May 2001

*email jon@u.washington.edu, telephone (206)-598-4117, fax (206)-598-6218
†Present address: Radionics, Burlington MA
‡Present address: Hydro-Aire, Burbank CA

Abstract

This report describes a new computer control system for a radiation therapy machine with an isocentric gantry and a multileaf collimator. It discusses the motivation and rationale for some of the features and development activities, and reports several measures of development effort, performance, and quality, determined after almost two years of operating experience.

Notable features of the control system include construction based on standard (vendor-independent) hardware and software components connected by a network, a simple and efficient user interface, close integration with the treatment planning system, automated record keeping for patient quality assurance and machine maintenance, and ease of maintenance and upgrading. Separation of control functions from data management functions results in a control program which is small, fast, and feasible to analyze thoroughly. Choice of features and internal design were informed by fifteen years experience using and maintaining computer controlled therapy machines.

The development method was chosen to ensure exceptional safety, reliability, and clinical acceptability. Much of the detailed specification was expressed in *formal* (mathematical) notation. This formal specification (not the prose description) served as the basis for most coding, test planning, and safety analysis. Testing and reviews were supplemented by new automated analyses methods including *theorem proving* and *model checking*.

1 Introduction

Several qualities are required of any radiation therapy machine: accuracy, ease of use, efficiency, and a manifestly safe and correct design. Additional qualities become important when a unique machine must be supported by a small in-house engineering staff: ease and economy of upgrades and maintenance. Here we report several design features and development activities which enabled us to achieve these qualities in a therapy machine control system.

We developed a new computer control system (both hardware and software) for the isocentric treatment unit at the Clinical Neutron Therapy System (CNTS) at the University of Washington Medical Center in Seattle [33, 22].

We describe three kinds of innovations: in the features visible to the users of the machine, in the system internals which are only visible to the engineers, and in the development method used to write the control program and assure its safety and correctness.

Choice of features and internal design for the new control system were informed by fifteen years experience using and maintaining a one-of-a-kind computer controlled therapy machine. By commonly used definitions, we have been routinely performing conformal treatment with a computer-controlled therapy machine since 1984. We also have several years of experience with computer-controlled linear accelerators from commercial manufacturers.

1.1 Related work

The literature on therapy machine control systems is scant. Karzmark [23] provides an early review of therapy machine technology including controls. Early computer control systems used a monolithic design with a single master control computer and no network [27, 26, 40]. Distributed control systems based on networked components have been used at large research accelerators for some time [4] and have also been applied to proton therapy [36, 32]. Mageras et al. [30, 29] and Fraass et al. [5] constructed conformal therapy control systems by networking a workstation running custom software to the control computer provided by the therapy machine manufacturer. Approaches to integrating treatment planning computers and therapy control systems are reviewed by Kalet et al. [22].

2 System description and history

Except for particle type (neutrons) and size (150 cm source to axis), our neutron therapy machine is similar to a modern therapy linac with a leaf collimator (Fig. 3)[1]. Clinical trials have shown that neutron radiation is the most effective therapy for some cancers [25].

Computer control is a vital part of the system and has dramatic clinical impact because it is necessary to produce shaped neutron fields with the leaf collimator in order to avoid unacceptable complications in neutron therapy [1]. Our collimator has no jaws or field shaping components other than the leaves themselves (it does not support external shielding blocks). The only way to deliver any field (even a square) is to set up the collimator leaves under computer control.

The Medical Center plans to operate this unique facility for an extended time into the future. In July 1999 our new system replaced the manufacturer's original control hardware and software which had been in use since the machine was installed in 1984. We anticipate a long lifetime for the new system as well.

3 Overall requirements and design philosophy

In order to design a computer control system, it is necessary to have an overall philosophy concerning the purpose of computer control and the appropriate roles of the computer, traditional (non-programmable) control mechanisms, and the human operators. We believe that poor choices at this level (or failure to make deliberate choices at all) were the root cause of accidents involving computer-controlled radiation therapy machines [27, 26]. Therefore we make our design philosophy explicit here.

The primary purpose of the computer control system is to enforce this safety requirement: *The therapy beam can only turn on or remain on when the actual setup of the machine matches a stored prescription that the operator has selected and approved.* Most of the design arises from elaborating this single requirement: the computer provides a database of prescribed fields and a way for the operator to select one; the computer continually scans all the actual settings and compares them to the prescribed settings, and sets interlocks that keep the beam off if they do not match. A secondary purpose of the computer control system is to make treatments more efficient by automatically setting up internal motions (leaves, wedges, etc.) and other items (the dosimetry system) that do not present collision hazards. Another secondary purpose of the computer control system is to log information used for record keeping, quality assurance, and machine maintenance.

[1] This figure appears at the end of the report

Control software is laborious to develop and difficult to make correct. We only use it where we can justify the effort and risk: in data-intensive operations where it is unlikely that a non-programmable mechanism could be made efficient and reliable. We continue to use hard-wired mechanisms where they are effective, including many safety and equipment protection functions.

The therapy machine remains safe if any computer halts or resets or its program aborts (*crashes*) at any time. Local (nonprogrammable) mechanisms place the equipment in a safe state (beam off, motions disabled) if the controlling computer fails. Essential data are saved: electromechanical counters retain the most recently measured dose, even if all power to the facility is lost.

We distinguish control functions from data management functions and delegate them to different programs and computers. Control functions must be executed to actually irradiate the patient, while data management functions are preparations or followups to treating the patient (such as maintaining the prescription file). Most of the control functions execute in a *therapy control program* that runs on a special-purpose embedded computer, while data management functions execute in several programs that run on general-purpose desktop computers [2]. Data management programs prepare input files (such as the prescription file) and the therapy control program only reads them. The therapy control program only writes certain output files (such as treatment records) and data management programs analyze, display, and print them[3]. This separation between control and data management keeps the control program small and enables us to concentrate effort on making it correct, reliable, and efficient.

Operators (and patients) should never have to wait for the computer. When there is a perceptible delay, it should arise from some unavoidable process in the environment or the controlled equipment (for example, waiting for the collimator leaves to reach their prescribed positions). Time-consuming file operations should be avoided by keeping programs and data in control computer memory whenever possible.

Primary responsibility for the safety and correctness of therapy remains with the human operators. It is the therapist's job to identify the patient, select the fields prescribed for that patient on each treatment day, and confirm that the prescribed settings and other information stored in the computer are consistent with other documentation (paper chart, printed treatment plan etc.). It is the therapist's job to position the patient, to guide external motions of the therapy equipment that might present collision hazards (gantry etc.), and to indicate when to turn on the therapy beam. It is the therapist's job to observe the patient and machinery during the treatment, to stop the treatment if problems occur, and to note unusual occurences and report them to the engineering staff. It is the therapist's job to keep a paper record of which treatments have occurred and the dose delivered for each field on each treatment day, as a cross-check on the computer records.

[2] There are additional control programs and embedded computers (section 5.1).

[3] A network makes it possible for programs running on different computers to use the same files (section 5.1).

4 Features visible to users

In this subsection we describe features visible to therapists and other users: the operator's console, integration with the treatment planning system, and record keeping. We devoted great effort to achieving usability and efficiency. These are not merely matters of convenience. Poorly designed features can make a system error-prone and difficult to check. Details appear in the therapist's guide [15] and the reference manual [14][4].

4.1 Operator's console

The operator's console provides a color graphic workstation with a keyboard, but the beam-on and beam-off functions are provided by hard-wired buttons on a control panel (Fig. 1). There are also lamps and numeric LED indicators on the panel for a few conditions and quantities which require conspicuous and uninterrupted display. A few indicators and functions that are provided at the workstation are also provided by lamps and buttons in the therapy room as well, to save therapists' steps and time.

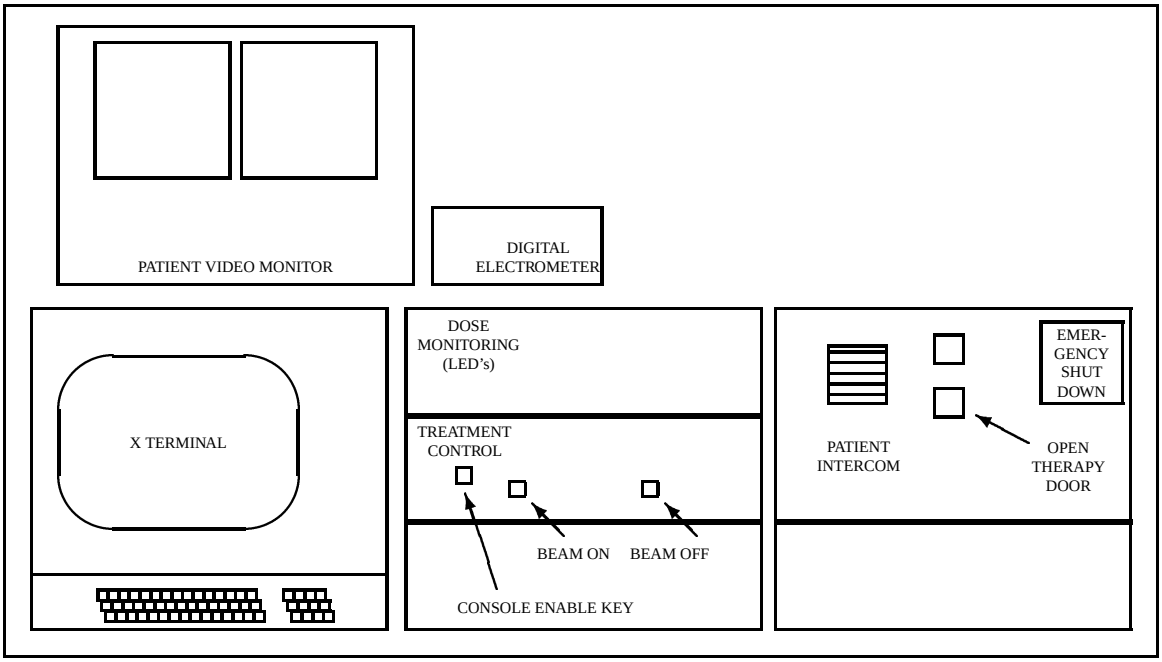

Figure 1: Therapy control console

[4]Most project documents are available from `http://www.radonc.washington.edu/physics/cnts/`

There is just one workstation at the console, with a single video display and keyboard. Some computer-controlled therapy machines use two or more displays and keyboards (with a separate workstation just for the leaf collimator, for example). We believe this is unnecessarily complicated and wastes space.

It has become usual for computer programs to display several (or many) panels or *windows*, controlled by a pointing device or *mouse* (our own treatment planning program [21] is one example). This is called the *desktop metaphor*; it was invented to help automate office work. We believe this style is not well-suited to a therapy machine. Treating a patient is not like typical office work; the therapist's attention should usually be focused on the patient, not the computer.

We adopted a *control panel metaphor* instead. The workstation display resembles a control panel with lamps and other indicators (Figs 4, 5, 6, 7)[5]. The operator issues commands by pressing keys on a keypad, like pushing buttons on a control panel (not like typing at a keyboard). We do not use the mouse (which would be difficult to use among the charts and films that often occupy the console desk surface). In general, the display shows fewer items but is clearer than typical office software (ours uses a larger typeface, color coding, and no overlapping windows).

Several indicators are always visible on the display: the currently selected patient and field, the operator (therapist or physicist) on duty, and six subsystem lamps. Each lamp is red when its subsystem is not ready for a treatment and green when it is ready; yellow indicates the operator has deliberately overridden a condition which would normally be indicated in red. When all lamps are green or yellow, the operator may turn on the beam by pushing a button on the control panel. When any condition occurs that causes the beam to turn off, at least one of the lamps turns red.

The operator presses keys to select the contents of the large central part of the display. Some screens show the patients and fields in the prescription database, so the operator can select one (Fig. 4). Another screen shows a summary of the currently selected field (Fig. 5). Other screens show details of particular subsystems (Fig. 6), and others show calibrations and other diagnostic information (Fig. 7). There are 20 different screens. Of these, 11 (including Fig. 7) are intended for engineers and technicians, not therapists. Therapists can perform typical (uneventful) treatments using only three: the patients screen, the fields screen (Fig. 4) and the summary screen (Fig. 5).

For a typical treatment the therapist selects a patient, selects a field, displays the summary screen, presses a key to automatically set up the dosimetry system and internal motions which pose no collision danger (leaf positions etc.), and enters the treatment room to position the patient and set up external motions (gantry rotation etc.). As each subsystem sets up, its lamp turns green. When all lamps are green, the therapist pushes the beam-on button. When the dosimetry system detects that the prescribed dose has been delivered, the beam turns off automatically and the dosimetry subsystem lamp turns red.

[5]These figures appear at the end of the report. The patient name in the figures is fictional.

The control program does not require therapists to perform operations in a particular fixed sequence. Most operations are available at all times, although some commands are disabled by particular conditions. For example the therapist can always display the list of fields (Fig. 4) but it is not possible to select a new field while the beam is on.

Certain operations (changing calibration factors etc.) are only available to physicists (including engineers and technicians). Each operator must log in by typing a username and password (this is the only operation where a therapist must type at the keyboard), and is identified as a therapist or physicist at this time.

4.2 Integration with the treatment planning system

The only way to enter therapy fields into the control system is to transfer them from the treatment planning program. We have observed this discipline without exception since the machine began operation [22]. There is no way to enter or edit fields at the therapy control console. The therapy control program only reads the prescription file, it never writes it or updates it. The prescription file is maintained by two data management programs: the treatment planning program and the archiver.

New patients and fields are added to the prescription file by the (locally developed) treatment planning program [21]. A dosimetrist at a treatment planning workstation uses this program to store the completed neutron treatment plan in a file server and notifies the therapist that the plan is ready. The therapist runs the treatment planning program at a different workstation in the neutron therapy control room, retrieves the plan, reviews it, and (if it is satisfactory) selects and confirms the planning program's transfer operation. At this point the therapist takes responsibility for performing the proper treatment. The planning program writes out the pertinent patient and field setup information and appends it to the control system prescription file.

It is usual to transfer additional fields for the the same patient on different days, after physicians and dosimetrists revise the plan or add boost fields. New fields are simply appended to the end of the file. To change a field (even one leaf), it is necessary to transfer a complete new version; the earlier version remains in the file but the therapist marks it *superceded*.

All fields for a patient remain in the prescription file until the patient completes the course of treatment. Then the therapist runs the archiver program to remove the patient and all of the fields to a permanent archive.

To read the latest version of the prescription file into the therapy control program, the therapist at the console presses the key to display the list of patients (this is the usual preliminary to selecting a patient, whether or not the prescription file has changed). The program reads the entire file into memory, replacing all of the previous contents. If the program is unable to read the file, or if it

encounters an error or an indication that the file has become corrupted, it retains all of the previous data so memory contents remain consistent. Treatments can continue (but perhaps without the most recent patients and fields) even when there is a problem with the treatment planning file server or the prescription file.

4.3 Record keeping for patient quality assurance and maintenance

The control program and the data management programs maintain a complete permanent record of successive steps in the treatment process for every patient.

The prescription file records all prescribed fields for each patient currently under treatment, including the date each field was transferred from the treatment planning system and the therapist who performed transfer (Fig. 4).

The therapy control program keeps a record in memory of each field's total dose to date, dose for the current day, and number of treatment days to date. This information is displayed for the therapist (Fig. 4). The program issues a warning if the therapist selects a completed field (whose prescribed totals have been reached) or a superceded field.

The dose-to-date records for all fields are written out to a disk file at the end of each treatment run so the information is not lost if the program shuts down. Each time the program restarts, it reads this file.

The control program appends a record to a treatment record file each time the therapy beam turns on or off, and at the end of each treatment run. Each record stores all of the pertinent machine state at that moment, including the prescribed and actual setting values, calibration factor values, the elapsed time and accumulated dose for the run, the currently selected patient, field and operator on duty, etc.

The same unique identifier for each treatment field is used in the treatment planning program, the prescription file and dose-to-date file, and the treatment records. This makes it possible to check for errors by comparing successive steps in the treatment process from planning through delivery. We call this *end-to-end checking.*

When patients complete their course of treatment, their records are removed from the prescription file and dose-to-date file and are stored in a permanent archive on disk. All treatment records are also stored. There is enough capacity in the disk file system to store many years of records on-line.

The therapy control program also writes a log that records pertinent treatment and machine events. At the operator's command the program can write out files showing the recent message traffic among

the networked control computers. These can be useful for machine maintenance and troubleshooting. They are also kept on-line permanently.

5 Internals

In this subsection we describe the system internals which are only visible to engineers: the overall architecture and the components. Details appear in the operations manual [9].

Long product life and ease of maintenance and upgrades are an overriding concern. Custom software development is so laborious and costly that we cannot afford to rewrite the entire control software every few years. The entire system design was very much driven by our need to postpone obsolesence.

5.1 Network-based hardware and software architecture

We chose a control system architecture composed of several computers connected by a network (Fig. 2). This confers several advantages: It eases development and maintenance because the machine can be operated (in some modes) even when some of the computers are disconnected or inoperable. It provides flexibility in locating equipment. It postpones obsolesence because we can perform future upgrades in stages, replacing components one at a time. We hope that the cost and effort of occasionally replacing components and (rarely) rewriting (small) portions of the custom software will be much less than the initial cost and effort of creating the new control system.

At this writing the control system includes eight networked computers (not counting the treatment planning computers). Each computer executes its own control program and communicates with the others by exchanging messages over the network.

The *X terminal* (actually a kind of computer) provides the operator's keyboard and video display. The *therapy control computer* executes our custom therapy control program. The *host* provides the file system that stores the executable control program files and all the data, including the prescription file derived from treatment plans. These computers replace a single master computer that provided all three functions in the original control system.

Five more computers sense and control the therapy equipment (relays, motors, ion chambers etc.): the *leaf collimator controller* (LCC), the *dose monitor controller* (DMC), the *treatment motion controller* (TMC), and two *programmable logic controllers* (PLCs). The PLCs handle single-point sensors and controls (limit switches, pushbuttons, lamps) and a few analog signals. They replace a part of the digital input/output system of the master computer in the old control system. The LCC,

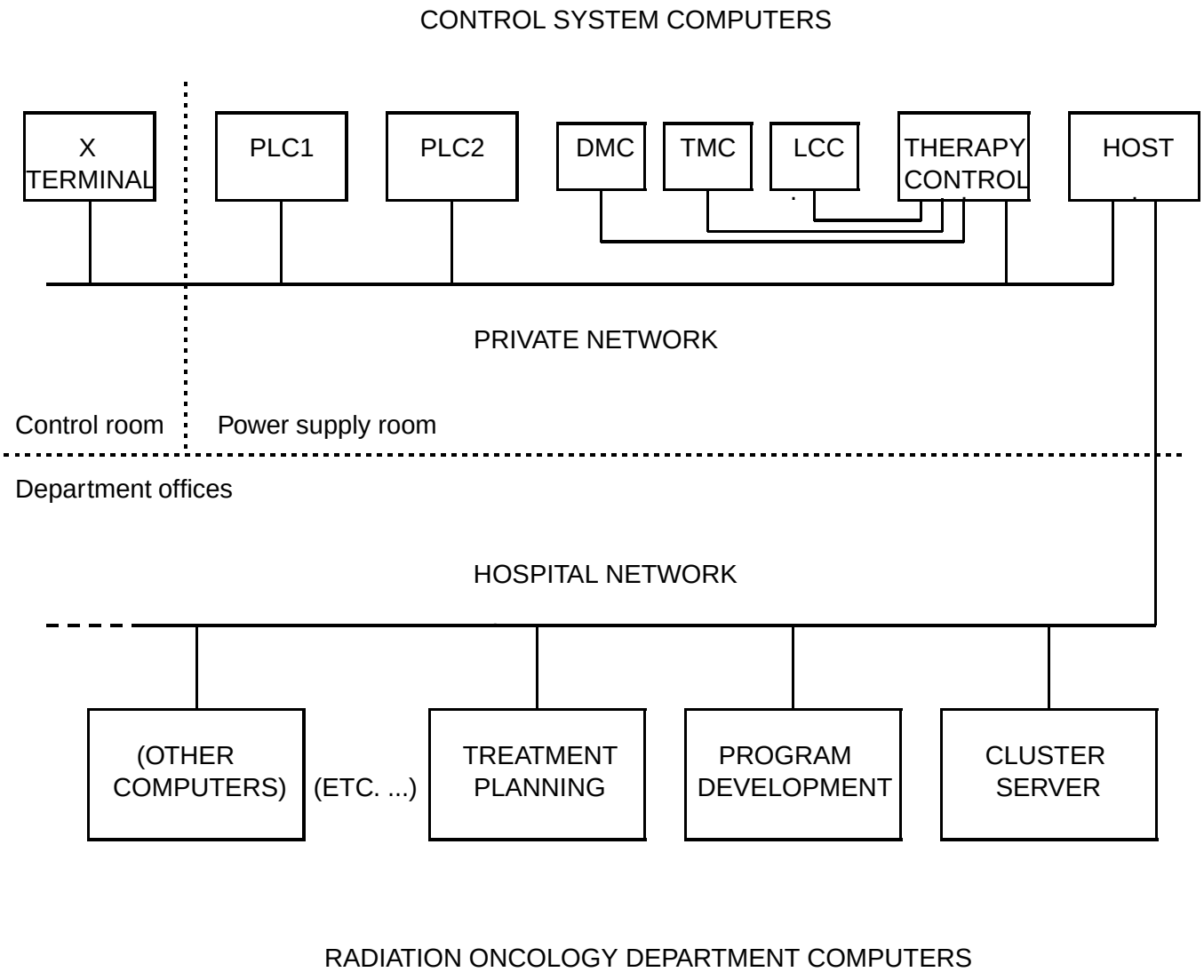

Figure 2: Network

DMC and TMC are retained from the old control system.

Each computer is largely autonomous and can continue to perform its essential functions even if contact with the others is lost. The safety of the therapy machine never depends on messages being received within some particular deadline, or being received at all. Each computer can detect communication errors or timeouts in the others and respond appropriately by indicating the problem and placing its local controls in a safe state. The network is used to share data and coordinate activities, to make progress while local controls ensure safety. Two or more computers must cooperate to begin any potentially hazardous operation but each computer by itself can return the equipment to a safe state. For example the therapy control computer must load the DMC with the prescribed dose and command it to close its relays in order to turn on the therapy beam. After that, the DMC itself opens its relays (forcing the beam off) when it detects that the prescribed dose has been delivered or an error has occurred.

The control system continues working if the host shuts down or becomes inaccessible. The therapy control program stores all data in program memory and only reads or writes files when new data become available. If the control program is unable to access the host it continues running with the data already in memory and notifies the operator. The operator can command the program to try again when the host becomes accessible.

We chose a centralized architecture where the therapy control computer acts as master and sends commands to all the others, which respond only to the master. This is a software design decision; the network supports messages among any computers.

The control computers and the host communicate over a private network that is separate from the hospital network (Fig. 2). This provides security and minimizes network congestion. The host is also connected to the hospital network so it can communicate with the treatment planning program and the other data management programs.

5.2 Selecting hardware and software products

Having decided upon the overall network-based architecture, our next task was to select particular hardware and software products (we use the word “products” broadly to include programming languages and software protocols, and items available at no cost from academic and research institutions).

To avoid obsolesence, we chose products which are now widely used, appear to have a long future, and are likely to be replaced by functionally similar products. Where possible, we chose products that are available from several manufacturers and that conform to some vendor-independent standard. To obtain reliability and stability, we chose products which have already been in wide use for

many years, so the worst defects have been shaken out and the remaining limitations are well understood. Where there was a choice between a simple product and a more complex one, we choose the simpler. Where a product offers a large and complex feature set, we limit our use to a stable (older, better understood) subset. We avoid products which seem overly complex or which might require excessive maintenance. This should reduce the number of failure modes and also reduce our dependence on particular suppliers.

The network is Ethernet hardware with TCP/IP software [39] (at this writing the LCC, DMC and TMC are connected using separate RS232 cables). The host communicates with the therapy control computer using NFS (network file system) [39], a standard software protocol. NFS enables programs to be written as if all files were attached locally; the actual location of the files on the network is specified in a configuration file that the program reads when it starts up. TCP/IP and NFS are now universally supported by manufacturers so almost any recent model workstation or personal computer could be used as the host[6]. The console computer is an X terminal [7], a computer configured by its manufacturer to display graphics expressed in X, another standard protocol [35] (most workstations and personal computers could also be so configured). The therapy control computer is a single-board computer from a semiconductor manufacturer [8]. It has no keyboard, display, or disks, but has onboard RS232 ports and an Ethernet network interface. It has an industry standard size, shape and edge connector (VME bus [28]) so it can be housed in a generic cabinet. The PLCs are industrial programmable logic controllers [9]. The LCC, DMC and TMC were provided by the cyclotron manufacturer.

Most of our programming effort was devoted to the therapy control program. The program is written in C [24], using only universally supported standard features and libraries, except for a few well-encapsulated modules that use the (also widely supported) `Xlib` library for X graphics [31] and (proprietary) operating system libraries for tasking, synchronization, and device control. The therapy control computer runs a real-time operating system [41] [10], which provides deterministic timing and is much smaller and simpler than a general purpose operating system such as Unix or Windows.

We wrote all the code “by hand”. The program does not include any database or graphics products. Such products can be complex and often require extensive operating system support, which tends to produce a large program that is difficult to analyze. Moreover, these products often change over time as new versions are released. This can make maintenance difficult.

We also wrote the PLC programs in ladder logic notation, using a language processor provided by the PLC manufacturer. The programs for the other computers were provided by their manufacturers.

[6] At this writing the host is a Hewlett Packard B160 workstation running the HP-UX 10.20 operating system.
[7] Tektronix XP217C
[8] Motorola MVME167-004B with a 25MHZ 68040 processor and 32MB memory
[9] Modicon Quantum series, Schneider Electric, Inc.
[10] VxWorks, Wind River Systems Inc.

6 Development method

In this subsection we describe the development method used to design, code, analyze and test the therapy control program.

Programming is notoriously fallible. It is typical for serious errors to escape detection in testing. Errors in a therapy machine control program have caused fatal accidents [27, 26]. Finding better methods for preventing and detecting errors is an active area of research and controversy. We decided to try *formal methods*: express the design in mathematics (this is called a *formal specification*), and use mathematical analyses (*theorem proving, model checking*) to detect errors and (eventually) demonstrate correctness. As far as we know, ours is the first project to apply formal methods in radiation therapy, although there are examples in other industries [6].

In particular, we hoped to demonstrate that the detailed design satisfies the central safety requirement: the beam can only turn on (or remain on) when the actual setup of the machine conforms to the stored prescription selected by the operator.

6.1 Writing the requirements

The requirements [16, 17, 12] express the properties that the system must have to be acceptable to its users, expressed in prose and diagrams that they can understand and critique. The authors and reviewers include the physicist who defined the physical and clinical requirements for the original machine, engineers who installed and maintain the machine, and clinical users of the machine. Writing the requirements was a major portion of the whole project effort, not just a preliminary.

6.2 Writing the formal specification

The formal specification is a collection of mathematical formulas that express the requirements as constraints on a set of *state variables.* It is expressed in Z [38, 7], a machine-readable notation for logic, set theory and arithmetic. The full text appears in [13]; excerpts and commentary appear in [18, 7, 19, 10].

The set of state variables represents all pertinent features of the treatment machinery and the treatment session. There are state variables for the prescribed and actual values for all the settings, the status of operations in progress, the patient and field identification used to access stored prescriptions, etc. We identified 410 scalar state variables (Table 2). Z provides constructs that make it convenient to deal with large numbers of variables.

We wrote formulas to express the constraints that must hold among the values of the state variables at all times. These formulas are called *invariants.* All safety requirements can be expressed as invariants. Invariants evaluate to *true* in situations that satisfy the constraints and *false* otherwise. The English paraphrase of one invariant says: "If the beam is on, all leaves are within tolerance of their prescribed values". A condition that causes an invariant to evaluate to *false* violates a safety requirement[11].

The invariants express precisely what it means for the program to be in a safe state. Most of the value added to the project by the formal specification inheres in the invariants because they express information that is not present in the program and cannot be inferred from the program. The program code expresses a great many state transitions. With great effort, we might be able to analyze the code to determine the set of reachable states. But this cannot tell us if the reachable states are safe. The program might be able to reach an unsafe state, due to a design oversight or a programming error. In fact, the safety analysis is actually nothing more than checking that the set of reachable states is a subset of the set of safe states. For this we need an explicit description of the safe states which is independent of the program, and that is what the formal specification provides.

In addition to the invariants, we also wrote formulas to express the state transitions. Unlike the invariants, these formulas correspond closely to the program code. However they are written in the same mathematical notation as the invariants, and they are more concise than code because they only model its effects, not its details. Each operation is described by a formula called a *precondition* which describes the states where the operation can begin and another called a *postcondition* which describes the state after the operation completes (the postcondition can be a function of the precondition). We described 105 state transitions. We wrote 2103 lines of Z in all (Table 2).

The structure of the formal specification also expresses the modular design of the program. We believe that writing and revising the formal specification helped us make a clear and concise design.

6.3 Analysing the formal specification

To demonstrate safety we must show that every reachable state is safe, as follows: First, show that the initial states are safe. Then, for each state transition, show that if its starting state is safe, then its ending state is safe also. That's all. It is not necessary to consider the (vast) collection of all possible program executions. We need only consider the initial state (trivial) and then individually analyze each of our state transitions (tedious but quite feasible). We check that each transition does not violate any invariants. It is not necessary to explicitly consider all possible values of the state variables because the transitions and the invariants are both represented symbolically (with variables instead of explicit values). We need only show that the formula that describes the transition implies the formula that expresses the invariants. This is usually a straightforward exercise in the simplification

[11]The formula paraphrased *if p then q* is *false* when *p* is *true* and *q* is *false*, and is *true* in all other situations.

of logical expressions. It should be possible to automate much of this using an automated theorem prover, but we used inspection (by eye) and a few pencil-and-paper calculations. Some examples appear in [19, 10]. Since these analyses were not automated, they are fallible and rely on subjective judgment regarding where to concentrate the inspection effort.

Some properties of a formal specification can be checked automatically. We checked for syntax and type errors using a tool [37] similar to a programming language compiler, and checked for *domain errors* (somewhat like array-out-of-bounds errors) using an automated theorem prover [34].

We analyzed some of our design's tasking and timing properties [8] with an exhaustive simulation technique called *model checking* [3]. We expressed portions of our design in a special programming language used by the checker, writing a much-simplified version of our program that nevertheless exercised certain requirements of interest. Then we expressed those requirements in another formal notation called CTL that includes temporal operators such as *until* and *eventually* so it can check progress properties, not just invariance. The checker tool constructs the (very large but finite) set of states reachable by the (simplified) program, performing a simulation of all possible program executions (including all possible interleavings of a multitasking program). The tool checks that each simulated execution path satisfies the requirements expressed in CTL. If it does not, it prints the entire execution path that contains the error.

The automated analyses achieve complete coverage within their (limited) scope; in that sense they are infallible. However, the choice of what to analyze and the judgment that the formal notation expresses a sufficiently accurate model of the real system is subjective and fallible.

6.4 Coding the program

The formal specification also served as the detailed design for most of the code (but not all code was formally specified, see Table 2). We used no other design notation (besides English). We coded using methods recommended in a textbook [7]. Usually we could implement each Z paragraph in less than a page of C code so it is easy to confirm by inspection that the code correctly achieves the intent of its specification. We performed a pencil-and-paper formal verification of one page of code [18]. Details about the code and program internals appear in the implementation report [11].

We had only limited access to the therapy machine (which was already in use, running under the old control system), so we developed program versions that do not require access (Table 3). The *demonstration* version runs on an ordinary desktop workstation. The *simulation* version executes on the single-board computer with the real-time operating system, but inputs and outputs to the therapy equipment are simulated by stub routines that read and write files instead. A third version executes on the therapy machine. Most of the code is the same in all versions.

The program was built up incrementally and was under continual revision. We recorded every addition, revision and correction to the program in a change log (Table 3)[12].

6.5 Testing and evaluating the program

The formal analyses described previously were applied to the formal specification, not the code. They were intended to reveal design errors, and cannot reveal coding errors. We used code inspections and testing to detect these errors.

We distinguish testing from evaluation. Testing is devoted to detecting errors where program behavior violates the specification [20]. Evaluation is devoted to clarifying requirements, assessing usability and obtaining suggestions for improvement. Most program use during development was evaluation, not testing; most changes to the program were additions and improvements, not corrections (Table 3)[13].

In the demonstration and simulation stages most tests were performed automatically by executing test scripts: files of simulated operator keystrokes and simulated inputs from the treatment machinery. Using scripts ensures that tests are repeatable (and are easy to repeat). The outcome of each test is recorded in log files written by the program under test. In the evaluation and acceptance test phases we ran the treatment machine (including the new control program) in exactly the same configuration as a therapist would use to treat patients. Therefore we could not use scripts. Instead, testers worked at the therapy console from a written test plan. Some tests simulated equipment and program failures during treatment operations (by substituting jumpers for relays, resetting computers, etc.).

7 Program size and other project metrics

The products of the development include documentation and assurance evidence (test plans etc.), not just the program itself (Table 1).

Our therapy control program is small, has minimal dependence on other products, and requires little operating system support, so it should be reliable and easy to maintain. It comprises 15,786 lines of C programming language code (not counting comments and blank lines), divided into 96 files (Table 2). Of these, 75 use only ANSI standard C constructs and libraries; 4 more use the

[12] The log does not distinguish changes made in the simulation and evaluation stages (except corrections). The entry in the table (742) is the total for both stages.

[13] We began the acceptance test phase when we considered the program finished. We subsequently decided on further revisions so the table shows both changes and corrections in this phase also.

PRODUCT	SIZE	
Requirements: overview [16]	106 pages	
Requirements: user interface (chapters 2, 6–9 in [17])	106 pages	
Requirements: hardware and files [12]	131 pages	
Formal specification (Z notation) [13]	129 pages	(2103 lines)
Scheduling model (Z notation) [8]	246 lines	
Scheduling simulation (SMV model checker, CTL) [8]	423 lines	
Program change logs	3144 lines	
Test data and scripts	30725 lines	
Acceptance test plan	42 pages	
Program code (C programming language)	15786 lines	(96 files)
Therapist's Guide [15]	41 pages	
Operations Manual [9]	125 pages	
Reference Manual [14]	233 pages	
Implementation Manual [11]	70 pages	

Table 1: Development products (documents and code)

X graphics libraries. These files were written and first tested on a general-purpose workstation. Only 17 files use the real-time operating system libraries; these had to be developed on the control computer. We only use 32 X library functions (out of more than 360 provided [31]) and 31 operating system functions (out of more than 1400 provided [42]). The total size of the executable program files loaded into the therapy control computer is 1,611,668 bytes (1,611 kilobytes, 1.6 megabytes): 716,380 bytes for the operating system (including network support), 502,429 bytes for X graphics support, and 392,859 bytes for our custom therapy control program.

Our control software is much smaller and simpler than typical application programs for general purpose computers, which often occupy several megabytes and are strongly interdependent with operating systems that occupy many megabytes more. It is thought to be infeasible to analyze such complex systems thoroughly, and many errors are detected by customers after long use.

Thanks to its small size, the therapy control program is fast. The program boots and completes its initialization sequence in about 35 seconds. The entire program and all data remain in computer memory so operators almost never perceive delays due to computer activity.

8 Operational experience

The new control system was put into operation in July 1999. At this writing (May 2001) it has delivered over 9000 fields (Table 3). We have never delivered an incorrect treatment, and we have

Category	Formal specification in Z			Code in C	
	variables	transitions	lines	files	lines
Pervasive constants and types	0	0	55	6	491
User interface (except graphics)	10	34	658	2	2058
In-memory database	185	9	333	6	686
Interlocking logic	114	2	188	2	474
Process and event handling	4	1	40	19	776
Low-level device control	14	18	375	13	2898
Files and persistent data	65	13	186	13	1598
Treatment sequence	18	28	268	2	1663
Graphics (utilities)	0	0	0	5	1006
Graphics (application-specific)	0	0	0	28	4136
Total	410	105	2103	96	15786

Table 2: Therapy control program: size of formal specification and code

.

never had to reschedule (delay) a treatment for technical reasons. Therapists and other staff say the new system is a big improvement in convenience and efficiency over the old system. It has been easy to revise the program to improve efficiency and to correct a few errors. We have already replaced three of the networked computers (both PLCs and the host).

8.1 Revisions after installation

In the first few months of operation we made many minor revisions to the therapy control program to improve efficiency and usability, and to deal with situations not anticipated when the specifications were written (Table 3). Some involved the user interface, others dealt with noise in analog signals, error messages from the controllers, and timing and interleaving of controller activity. None required extensive changes to the design or code.

We made several changes to the network before arriving at the configuration pictured in Fig. 2. At first the control computers were connected to a segment of the hospital network and the department cluster server was also the control system host. Operators sometimes experienced appreciable delays when the control program attempted to read or write files; during one incident the program crashed, during another a file was corrupted. Eventually the problem was traced to an intermittently faulty device on the hospital network. We provided the control system with its own host and a private network, where there are many fewer devices and (we hope) fewer failure modes. Since then we have experienced no network problems. Network reconfigurations required no changes to the therapy control program, and no changes to the procedures used by therapists at the treatment

Project phase			Sessions		Changes	
Name	Environment	Begun	Days	Fields	total	errors
Demonstration	Desktop workstation	Oct 1996	—	—	965	126
Simulation	Single-board computer	Jan 1998	—	—	742	57
Evaluation	Therapy machine	Feb 1998	42	582	¨	27
Acceptance test	Therapy machine	Apr 1999	21	403	91	8
Initial use	Therapy machine	Jul 1999	91	2015	35	6
Routine use	Therapy machine	Nov 1999 – Apr 2001	370	7165	30	4

Table 3: Experience during testing and use

planning workstations or the therapy console.

Recently we replaced both PLCs with newer models that use Ethernet (instead of RS232) and a different command protocol. This required changing four source files in the therapy control program.

8.2 Error detection and correction

The number of errors discovered and corrected has declined in each project stage (Table 3). Eight errors were discovered during acceptance testing. Ten errors escaped detection during development and were discovered by therapists or physicists during actual use. At this writing no errors have been discovered in the last eight months; this suggests the program is now nearly error free. The program has a delivered error density of 10/15,786, less than one error per thousand lines of code (KLOC). Typical commercial software products are delivered with one to ten errors/KLOC [2]. We delivered our earlier treatment planning system with 2.1 errors/KLOC [20]. In that project we also used detailed prose requirements and thorough testing (but no automated test scripts), so we conclude that formal methods and (and possibly test automation) contributed to the improvement in quality.

Our efforts to prevent errors concentrated on preventing the most serious incidents: setting up the machine incorrectly, or disabling the machine so it could not complete a treatment. No such incidents have occurred during acceptance testing or use. The most serious incidents that did occur involved record keeping, where the program recorded the wrong number of treatments or the wrong dose in the in-memory dose-to-date records. Therapists noticed the errors on the first day when a discrepancy appeared (they also could have been discovered by reviewing treatment records).

On several occasions the program has halted spontaneously (*crashed*). Three such incidents occurred during power or network disturbances. Twelve occured at the end of a run, shortly after the

beam turns off. This suggests a programming error but we have not discovered it (these incidents occur less than once a month, less than once every 770 fields). Such incidents were anticipated in the system design. There was no hazard and operations resumed promptly and without difficulty when the program was restarted.

9 Conclusions

A therapy machine control system can be constructed largely from standard (vendor-independent) hardware and software components connected by a network. This eases development and maintenance and postpones obsolescence. Operations can be made robust and safe despite occasional failures in the network or individual components.

A modern radiation therapy machine can have a simple and efficient user interface, provided that designers pay careful attention to the operators' tasks, so they can eliminate unnecessary steps and distracting elements.

Separating control functions from data management functions makes it possible to write a control program which is small, fast, and feasible to analyze thoroughly.

Close integration with the treatment planning system together with automated record keeping can help ensure accurate and correct treatments. Providing each field with the same unique identifier in treatment planning, machine setup, and treatment records makes it possible to check for errors by comparing successive steps in the treatment process from planning through delivery. Much of this checking could be performed automatically.

Formal software development methods including automated analyses can assist in creating a concise, correct design and can help prevent and detect programming errors. Formal methods have potential for introducing additional automation into the development process that might save time and improve quality further.

References

[1] Mary Austin-Seymour, Richard Caplan, Kenneth Russell, George Laramore, Jon Jacky, Peter Wootton, Sharon Hummel, Karen Lindsley, and Thomas Griffin. Impact of a multileaf collimator on treatment morbidity in localized carcinoma of the prostate. *International Journal of Radiation Oncology Biology Physics*, 30(5):1065–1071, Dec 1994.

[2] Boris Beizer. *Software System Testing and Quality Assurance*. Van Nostrand Reinhold, New York, 1984.

[3] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J. W. De Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency*, pages 124–175, Noordwijkerhout, The Netherlands, 1993. REX School/Symposium, Springer-Verlag. Lecture Notes in Computer Science, vol. 803.

[4] L. R. Dalesio, M. R. Kraimer, and A. J. Kozubal. EPICS architecture. In C. O. Pak, S. Kurokawa, and T. Katoh, editors, *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, pages 278–282, 1991. ICALEPCS, KEK, Tsukuba, Japan.

[5] B. A. Fraas, D. L. McShan, M. L. Kessler, G. M. Matrone, J. D. Lewis, and T. A. Weaver. A computer-controlled conformal radiotherapy system. I: Overview. *International Journal of Radiation Oncology Biology Physics*, 33(5):1139–1157, 1995.

[6] Susan Gerhart, Dan Craigen, and Ted Ralston. Experience with formal methods in critical systems. *IEEE Software*, 11(1):21–28, Jan. 1994.

[7] Jonathan Jacky. *The Way of Z: Practical Programming with Formal Methods.* Cambridge University Press, 1997.

[8] Jonathan Jacky. Analyzing a real-time program with Z. In Jonathan Bowen, Andreas Fett, and Michael Hinchey, editors, *ZUM '98: The Z Formal Specification Notation.* Eleventh International Conference of Z Users, Springer-Verlag, 1998. Lecture Notes in Computer Science.

[9] Jonathan Jacky. Clinical neutron therapy system installation and operations. Technical Report 99-08-01, Department of Radiation Oncology, University of Washington, Box 356043, Seattle, Washington 98195-6043, USA, August 1999.

[10] Jonathan Jacky. Lessons from the formal development of a radiation therapy machine control program. In Michael G. Hinchey and Jonathan P. Bowen, editors, *Industrial-Strength Formal Methods in Practice*, pages 185–206. Springer-Verlag, 1999.

[11] Jonathan Jacky. Clinical neutron therapy system implementation. Technical Report 2001-03-01, Department of Radiation Oncology, University of Washington, Box 356043, Seattle, Washington 98195-6043, USA, March 2001.

[12] Jonathan Jacky, Michael Patrick, and Ruedi Risler. Clinical neutron therapy system, control system specification, Part III: Therapy console internals. Technical Report 95-08-03, Radiation Oncology Department, University of Washington, Seattle, WA, August 1995.

[13] Jonathan Jacky, Michael Patrick, and Jonathan Unger. Formal specification of control software for a radiation therapy machine. Technical Report 95-12-01, Radiation Oncology Department, University of Washington, Seattle, WA, December 1995.

[14] Jonathan Jacky and Ruedi Risler. Clinical neutron therapy system reference manual. Technical Report 99-10-01, Department of Radiation Oncology, University of Washington, Box 356043, Seattle, Washington 98195-6043, USA, October 1999.

[15] Jonathan Jacky and Ruedi Risler. Clinical neutron therapy system therapist's guide. Technical Report 99-07-01, Department of Radiation Oncology, University of Washington, Box 356043, Seattle, Washington 98195-6043, USA, July 1999.

[16] 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.

[17] 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.

[18] 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.

[19] Jonathan Jacky, Jonathan Unger, Michael Patrick, David Reid, and Ruedi Risler. Experience with Z developing a control program for a radiation therapy machine. In Jonathan P. Bowen, Michael G. Hinchey, and David Till, editors, *ZUM '97: The Z Formal Specification Notation*, pages 317 – 328. Tenth International Conference of Z Users, Springer-Verlag, 1997. Lecture Notes in Computer Science 1212.

[20] Jonathan Jacky and Cheryl P. White. Testing a 3-D radiation therapy planning program. *International Journal of Radiation Oncology, Biology and Physics*, 18:253–261, January 1990.

[21] Ira J. Kalet, Jonathan P. Jacky, Mary M. Austin-Seymour, Sharon M. Hummel, Kevin J. Sullivan, and Jonathan M. Unger. Prism: A new approach to radiotherapy planning software. *International Journal of Radiation Oncology, Biology and Physics*, 36(2):451–461, 1996.

[22] Ira J. Kalet, Jonathan P. Jacky, Ruedi Risler, Solveig Rohlin, and Peter Wootton. Integration of radiotherapy planning systems and radiotherapy treatment equipment: 11 years experience. *International Journal of Radiation Oncology, Biology and Physics*, 38(1):213–221, 1997.

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

[24] Brian W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall, second edition, 1988.

[25] G. E. Laramore, J. M. Krall, T. W. Griffin, W. Duncan, M. P. Richter, K. R. Saroja, M. H. Maor, and L. W. Davis. Neutron versus photon irradiation for unresectable salivary gland tumors: final report of an RTOG-MRC randomized clinical trial. *International Journal of Radiation Oncology, Biology and Physics*, 27(2):235–240, 1993.

[26] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[27] Nancy G. Leveson and Clark S. Turner. An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7):18–41, July 1993.

[28] Yu-cheng Liu. *The M68000 Microprocessor Family: Fundamentals of Assembly Language Programming and Interface Design*. Prentice-Hall, 1991.

[29] G. S. Mageras, Z. Fuks, J. O'Brien, L. J. Brewster, C. Burman, C. S. Chui, S. A. Leibel, C. C. Ling, M. E. Masterson, R. Mohan, and G. J. Kutcher. Initial clinical experience with computer-controlled conformal radiotherapy of the prostate using a 50-MeV medical microtron. *International Journal of Radiation Oncology Biology Physics*, 30(4):971–978, 1994.

[30] G. S. Mageras, K. C. Podmaniczky, and R. Mohan. A model for computer-controlled delivery of 3-d conformal treatments. *Medical Physics*, 19(4):945–953, 1992.

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

[32] Eros Pedroni, Reinhard Bacher, Hans Blattmann, Terence Bohringer, Adolf Coray, Antony Lomax, Shixiong Lin, Gudrun Munkel, Stefan Schweib, Uwe Schneider, and Alexander Tourovsky. The 200-MeV proton therapy project at the Paul Scherrer institute: Conceptual design and practical realization. *Medical Physics*, 22(1):37–53, January 1995.

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

[34] Mark Saaltink. The Z/EVES system. In Jonathan P. Bowen, Michael G. Hinchey, and David Till, editors, *ZUM '97: The Z Formal Specification Notation*, pages 72 – 85. Tenth International Conference of Z Users, Springer-Verlag, 1997. Lecture Notes in Computer Science 1212.

[35] R. W. Scheifler and J. Gettys. The X window system. *ACM Transactions on Graphics*, 5(2):79–109, 1986.

[36] J. M. Slater, D. W. Miller, and J. O. Archambeau. Development of a hospital-based proton beam treatment center. *International Journal of Radiation Oncology, Biology and Physics*, 14(4):761–775, 1988.

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

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

[39] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols.* Addison Wesley, 1994.

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

[41] Wind River Systems, Inc., Alameda, California. *VxWorks Programmer's Guide 5.3.1*, 1997.

[42] Wind River Systems, Inc., Alameda, California. *VxWorks Reference Manual 5.3.1*, 1997. Edition 1.

(drawing goes here)

Figure 3: Neutron therapy machine

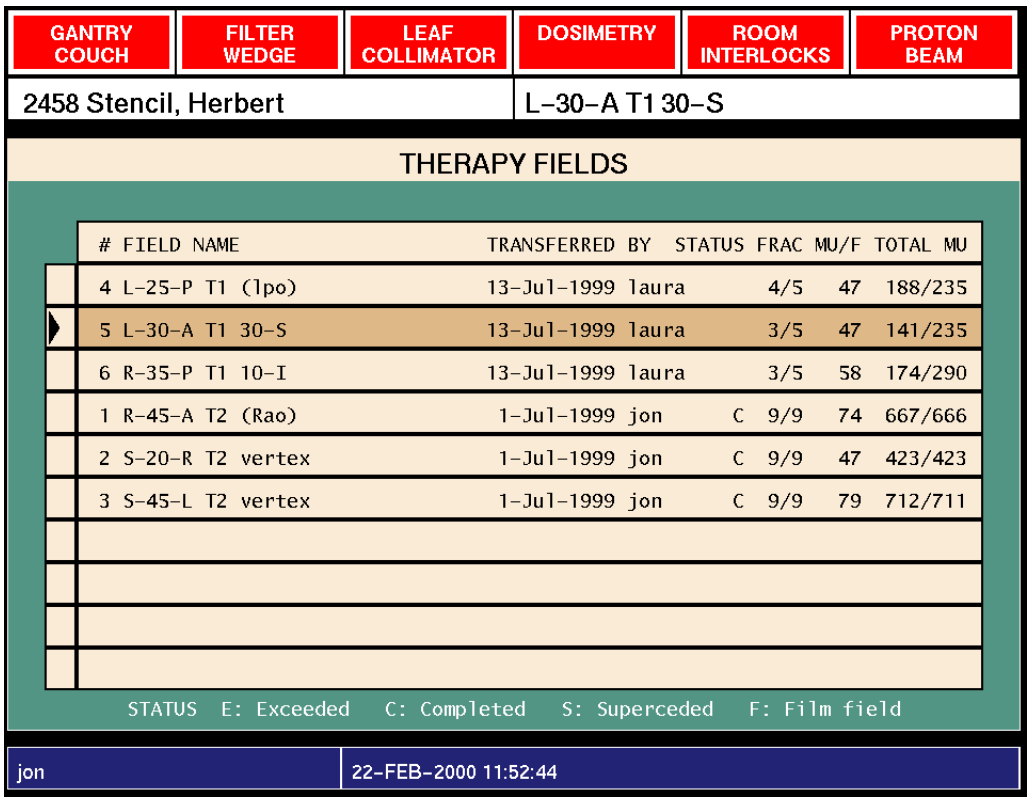

Figure 4: Field list screen

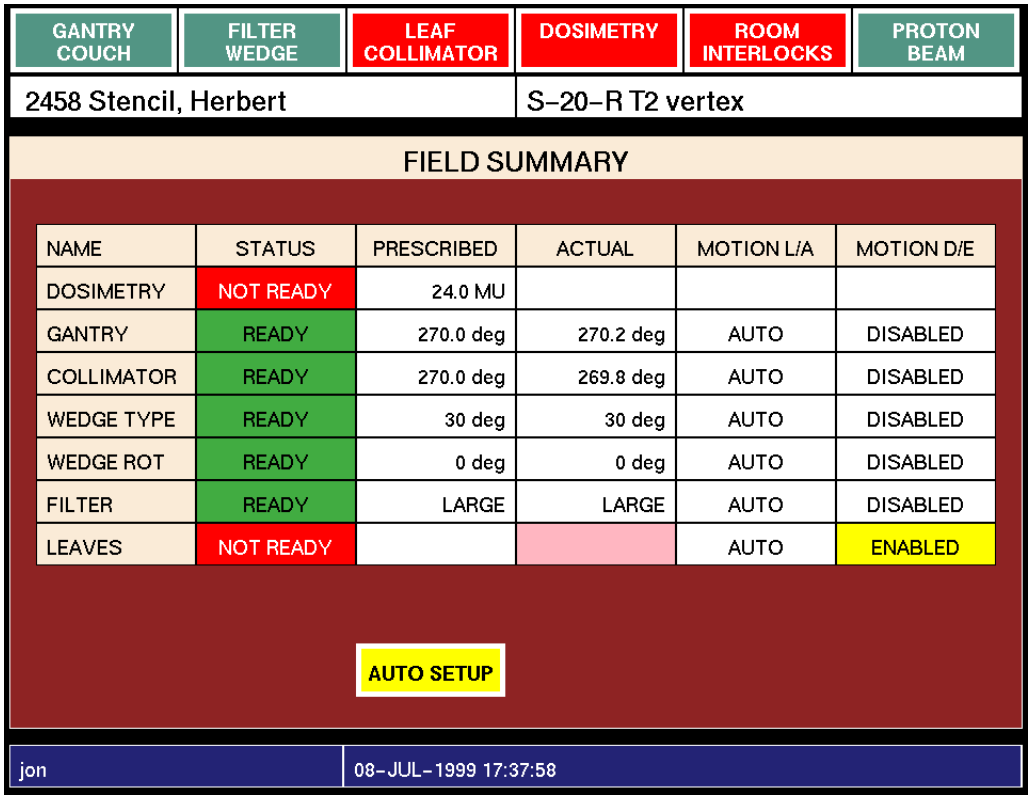

Figure 5: Summary screen

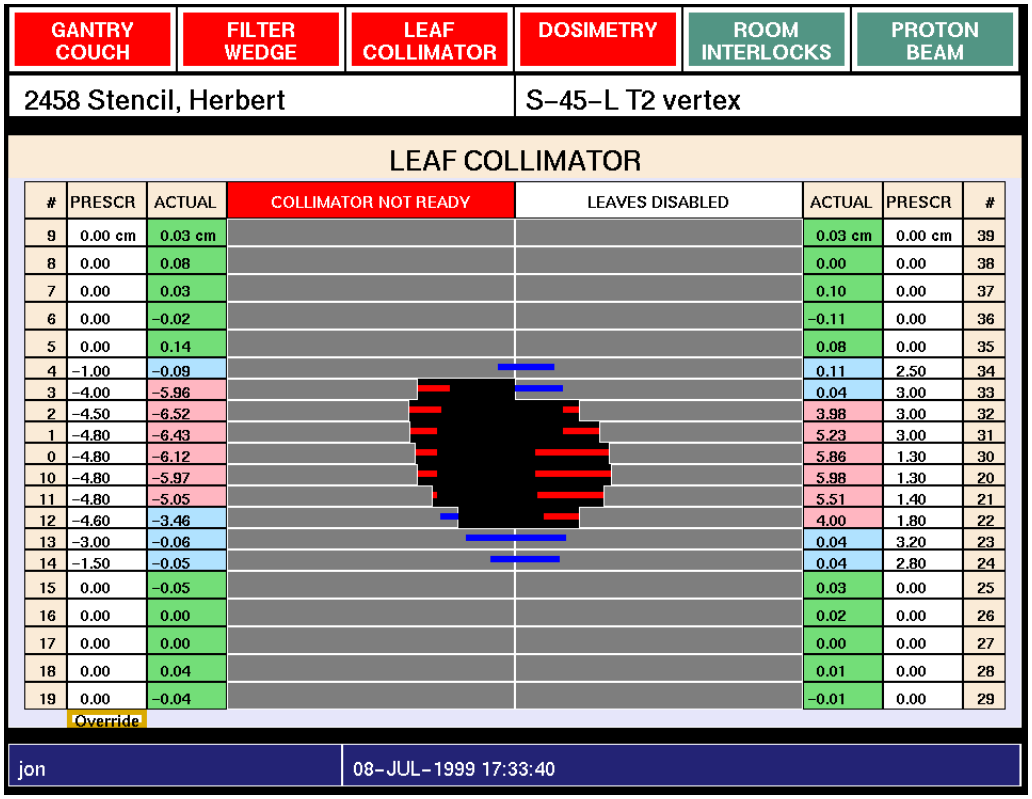

Figure 6: Leaf collimator screen

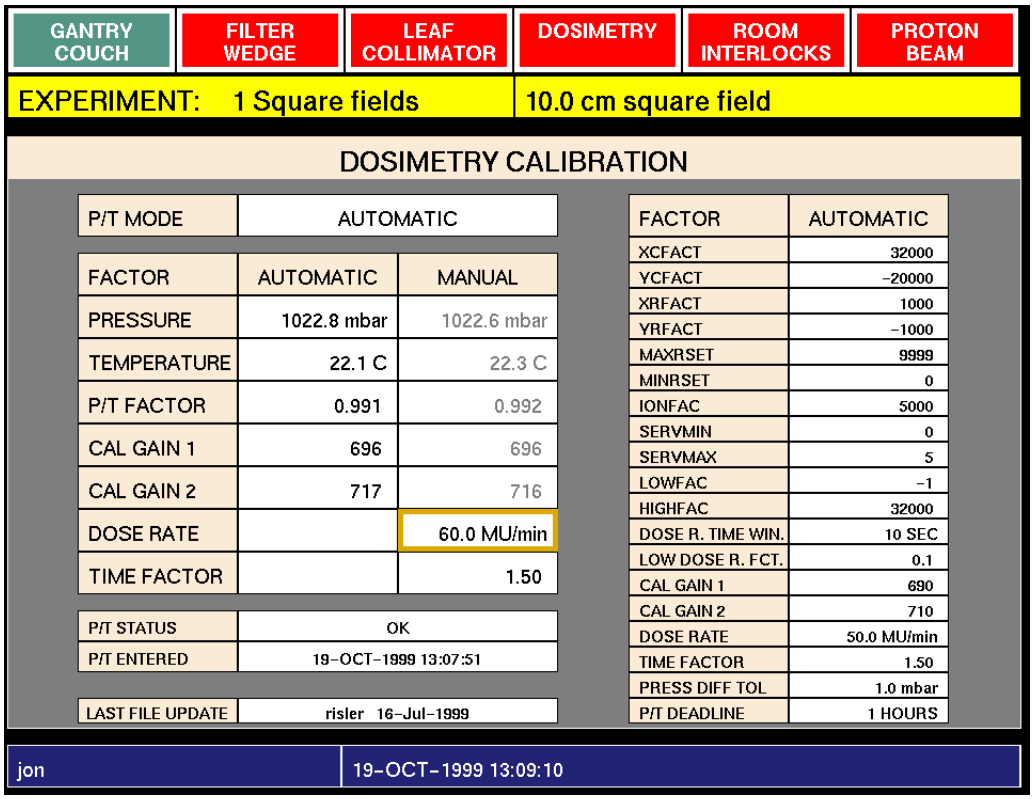

Figure 7: Dose calibration screen