Implementation of a Proton Therapy Supervisory System for iThemba LABS

by

Sehlabaka Qhobosheane

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Engineering at Stellenbosch University

Supervisors:

Dr. Mike Blanckenberg
Department of Electrical and Electronic Engineering

Dr. Neil Muller
iThemba LABS

December 2012

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

December 2012

Abstract

The Proton Therapy facility at iThemba LABS employs the passive double scattering method to modify the narrow proton beam from the accelerator into a broad treatment beam. The facility uses various safety and control systems to ensure that the beam is correctly configured and functional, that the patient is correctly positioned for treatment, that the treatment room is cleared and armed for treatment, and that the treatment is terminated when the required dose is administered to the patient. These systems collectively form the Proton Therapy Control System. One such control system is the Supervisory System which coordinates all other subsystems by supplying them with beam parameters and other treatment information needed to configure the entire treatment therapy unit for the correct irradiation of a patient. It allows for management of other control systems and is the primary user-interface to the proton therapy system.

This control system was developed on Scientific Linux 5.5 operating system with Kernel 2.6.18 using Qt/C++ widgets set for the user interface. The Supervisory System also uses the Eagle technology EDRE driver in order to use the Eagle Technology 848C PCI DAQ card to interface to the Therapy Safety Bus so as to access various safety and fail-safety lines.

The treatment parameters are parsed from the treatment planning files located on the Radiotherapy Database server using Boost.Spirit template meta-programming techniques. These beam parameters and treatment information are then send to other control systems using the Open Network Computing Remote Procedure Calls (ONC-RPC).

Opsomming

Die Protonterapie behandeling fasiliteit by iThemba LABS maak gebruik van die passiewe dubbel verstrooiings metode om 'n dun proton bundel komende van die versneller te transformeer na 'n breë proton bundel wat geskik is vir behandeling. Verskeie beheer-en veiligheidsstelsels werk saam om te verseker dat die pasiênt reg geposisioneer is en dat die behandelings stelsel korrek gekonfigureer en funksioneel is. Die stelsels verseker ook dat die behandelingskamer voorbery en ontruim is voor behandeling en dat die behandeling getermineer word wanneer die verlangde dosis bestraling toegedien is. Die beheer-en veiligheidsstelsels staan gesamentlik bekend as die protonterapie beheerstel. Een van die substelsels van die protonterapie beheerstelsel is die behandelings bestuurstelsel. Die behandelings bestuurstelsel koördieneer al die ander substelsels deur hulle te voorsien van bundel parameters en ander informasie wat nodig is vir die konfigurasie van die hele behandelings eenheid om sodoende te verseker dat 'n pasiënt reg bestraal word.
Die behandelings bestuurtelsel is die primêre gebruikers koppelvlak van die protonterapie stelsel en dit verskaf ook 'n koppelvlak met die ander beheerstelsels in die protonterapie fasiliteit. Scientific Linux 5.5 asook Qt/C++ widgets vir die gebruikers koppelvlak is gebruik vir die ontwikkeling van die behandelings bestuurstelsel sagteware. Die beheerstelsel sagteware maak ook gebruik van die "Eagle Technology" EDRE sagteware drywer om die "Eagle Technology" 848C PCI DAQ data-versamel bord aan te spreek. Die bord word gebruik vir die monitor en beheer van die Terapie Veiligheid Bus wat bestaan uit verskeie beheer en status lyne.
Die behandelings parameters word verkry vanaf die beplannings lêer op die radioterapie databasis bediener met behulp van "Boost.Spirit" meta-programmerings tegnieke. Die bundel parameters en behandelings informasie word dan gestuur na ander beheerstelsels met behulp van "Remote Procedure Calls" (ONC-RPC).

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations who have contributed to this work;

- My supervisors Dr. Neil Muller and Dr. Mike Blanckenberg for their guidance, ideas, motivation, knowledge and patience
- iThemba LABS and the University of Stellenbosch for giving me the opportunity to do research at their facilities and for providing me with the funding for my studies
- Mr Lebina Ts'epe for believing in me and for providing me with the initial funding through his company CalCommunications
- Mr. Evan de kock, Mr. Christo van Tubbergh, Mr. Jan van der Merwe, Mr. Nolan Klaasen and Mr. Casey Callaghan for their help, knowledge and support
- Mr. Mike Hogan, Mr. Lebelo Serutla, Dr. John Pilcher and Dr. Gillian Arendse for believing in me and for their support
- The Open Source community for their philosophies and excellent free software
- All my friends and family for their support and motivation
- My wife 'Makatleho Qhobosheane (Ngoanez) for her love, support and inspiration
- *Chess.com* for providing the excellent pastime and for keeping me sane
- And finally, mother Nature for Life and the Universe to explore.

Contents

List of Figures

List of Tables

Nomenclature

Subscripts

i	The i-th field.
j	The j-th fraction.

Acronyms and Abbreviations

BSC	-	Beam steering controller
CCD	-	Charged-coupled device
CT	-	Computed tomography
DMC	-	Dose monitor controller
DRR	-	Digitally reconstructed radiograph
EBNF	-	Extended Backus-Naur Form
EDC	-	Energy degrader controller
FC	-	Faraday cup
FCC	-	Faraday-cups controller
FPGA	-	Field-programmable gate array device
HIU	-	Hardware interlock unit (subunit of the safety interlock system)
HV	-	High voltage
HV-PSU	-	High-voltage power supply unit
I/L	-	Interlock (with specific reference to the safety interlock system)
LED	-	Light emitting diode
MCIS	-	Main cyclotron interlock system
NIC	-	Network Interface Card
N/C	-	A switch or relay whose contacts are normally closed

N/O	-	A switch or relay whose contacts are normally open
PC	-	Personal Computer
PCI	-	Peripheral Component Interconnect Local Bus
PT	-	Proton therapy
PT1	-	The existing proton therapy facility that utilizes the treatment vault BG1
PPS	-	Patient positioning system
PSU	-	Power supply unit
RCS	-	Range control system
RMP	-	Range modulator propeller
RMD	-	Range monitoring detector
RMU	-	Range monitoring unit
RT	-	Radiotherapy
SABUS	-	South African (communication) bus
SIS	-	Safety interlock system (subsystem of the TSCS)
SOBP	-	Spread-out Bragg peak
SPC	-	Solid-pole cyclotron
SPG	-	Stereophotogrammetric
SCU	-	Signal conversion unit (subunit of the TSCS)
SIC	-	Safety interlock computer (subunit of the SIS)
SIU	-	Signal interface unit (subunit of the HIU)
SS	-	Supervisory system
SSC	-	Separated-sector cyclotron
TCS	-	Therapy control system
TSCS	-	Therapy safety control system
TSB	-	Therapy safety bus
TTL	-	Transistor-transistor logic
USB	-	Universal Serial Bus
VLU	-	Voting logic unit (subunit of the HIU)

Chapter 1

Introduction

iThemba Laboratories for Accelerator-Based Sciences (iThemba LABS) is a multidisciplinary research and educational center focusing on accelerator physics, radiotherapy, and production of radio-isotopes for research and medical use. It is administered by the National Research Foundation (NRF) and has two operational sites; one in Gauteng Province, and another in the Western Cape.

The Medical Radiation Group is situated in the Western Cape site where it offers both Neutron and Proton therapy for treatment of lesions. Both types of therapy make use of the variable-energy k-200 Separated Sector Cyclotron, and while the Neutron Therapy Unit was designed abroad, the Proton Therapy Unit was designed locally. The proton therapy facility makes use of a combination of various devices to modify the characteristics of the 200 MeV proton beam from the accelerator into a broad treatment beam. It also uses various safety and control systems collectively called the Proton Therapy Control System to ensure the correct and safe irradiation of the patient, and to ensure safety of the personnel during treatment.

The existing proton therapy control system consists of old and difficult-to-maintain subsystems, thus a number of projects aimed at expanding the facility and upgrading the hardware and software used in the proton therapy unit are ongoing. One such system is the Supervisory System which is the subject of the work done in this thesis.

The aim of this thesis is to implement the Supervisory System for the proton therapy facility at iThemba LABS, which configures and coordinates all other subsystems of the Proton Therapy Control System for the correct and safe irradiation of the patient during treatment.

1.1 Organization of the Thesis

The remainder of this thesis is laid out as follows: Chapter 2 gives a brief overview of the proton therapy facility at iThemba LABS and the current supervisory system. Chapter 3 covers the new Proton Therapy Control System

and shows how the supervisory system maps all other subsystems together. Chapter 4 gives an indepth coverage of the Therapy Safety Bus simulation rig which simulates safety lines monitored and triggered by the supervisory system during treatment. Chapter 5 provides the detailed functional specification of the supervisory system, its top-level design and how it was implemented. Chapters 6 and 7 conclude the thesis with a discussion of how the supervisory system was tested, the results obtained, and with recommendations for future work on the system.

Chapter 2

Proton Therapy at iThemba LABS

Proton therapy at iThemba first started in September 1993 using the 200 MeV beam. Proton therapy treatment offers a number of advantages over alternative radiation modalities. One of the most significant advantages is that higher doses of radiation can be used to control and manage lesions while significantly reducing damage to healthy tissue and vital organs [1].

In practice, it is relatively easy to modulate the physical properties of a proton beam than it is to change the beam direction with respect to the patient, which normally requires a complex and expensive gantry system. Instead it is more cost effective to position the patient correctly with respect to a fixed proton beam [2; 3; 4]. The proton therapy facility at iThemba LABS follows this simpler approach (of localized treatment) and uses a fixed horizontal beam delivery system that implements the passive double scattering method to produce a uniform dose, while an assembly of cylindrical block collimators shape the beam to match the target volume [5]. A collection of range modulator propellers (wheels) is used to vary the beam's energy thus producing a dose distribution with the required Spread out Bragg Peak (SOBP). Patient positioning is achieved by an intricate system consisting of a custom-made marker-carrier, stereophotogrametry (SPG) system and a motorized chair. The marker-carrier is fitted with diopaque and retro-reflexive markers which are detected by the SPG system through a number of charged coupled device (CCD) cameras in order to compute their respective positions in a 3D coordinate system [5; 6; 7; 8]. The motorized chair, on which the patient is placed, has an immobilization device which is used to fix (move) the marker-carrier, and thus the patient to the position required for treatment. Figure 2.1 shows the current patient positioning system highlighting the SPG system with its array of CCD cameras as well as the treatment chair with its immobilization device.

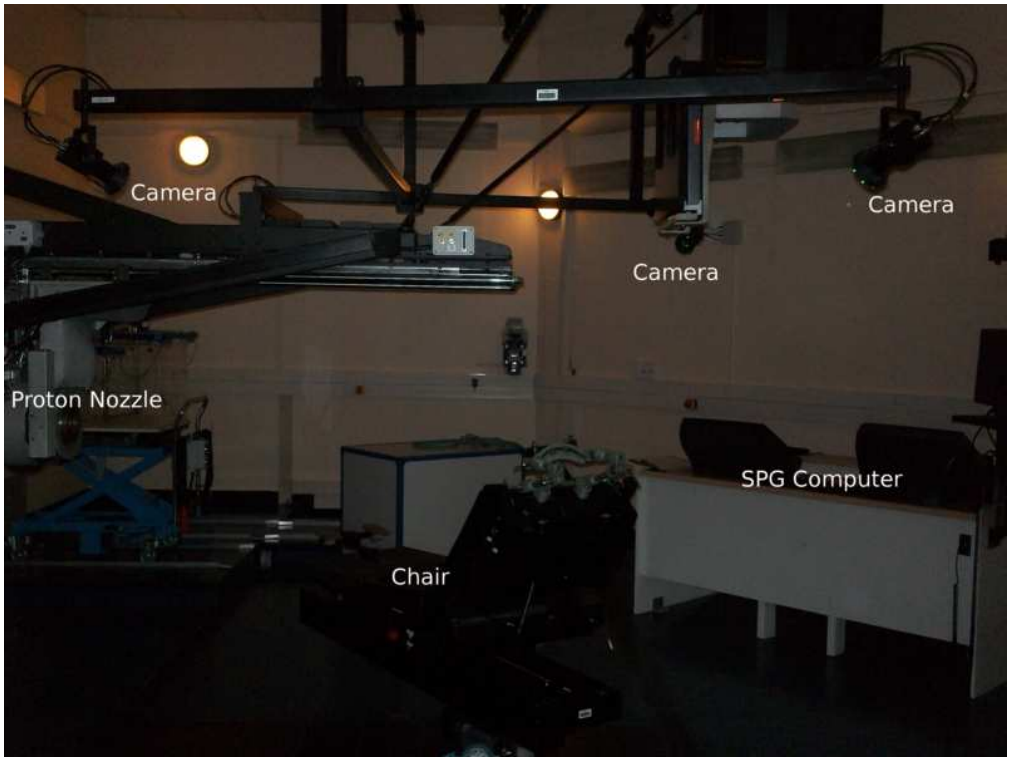

Figure 2.1: Patient Positioning System

2.1 The Current Therapy Control System

Together with the hardwired interlock system, the existing therapy control system consists of a distributed computer control system running on OS/2 operating system, and which has been in operation since 1990 [9].

Proton therapy at iThemba consists of five distinct phases [10]:

- Daily calibration of the SPG System: for accurate patient positioning.
- Dosimetry: which calculates proper radiation dose for treatment.
- Treatment planning and preparation of the patient: which is done once-off before patient treatment.
- Treatment simulation: done before actual treatment to test if all systems are ready for treatment, and to prepare the patient for the actual treatment.
- Treatment delivery to the patient.

The current Supervisory System provides an inteface to radiation therapists and radiographers for carrying-out basic treatment planning and preparation.

During actual treatment, it provides an interface for cross-checking beam parameters and patient information with the SPG computer and the treatment simulation form (sheet). It also provides instructions to signal when it is the right time to activate the barcode tracker for verifying patient-specific beam components. It is worth noting that the current supervisory system is not electronically interlocked and/or networked with the barcode tracker, SPG, DMC, and double-wedge systems, which all require independent checks during treatment delivery [10].

2.2 Other Radiation Therapy Supervisory Systems

Apart from the control system currently operational at iThemba LABS, there is a handful of other proton therapy facilities that have implemented supervisory systems to achieve similar functionalities.

The Indiana University's Proton Therapy Center, formerly Midwest Proton Therapy Institute (MPRI) has implemented a similar control system running on a Linux platform [11]. This system, called the Treatment Room Controller, employs KDE/Qt widgets for its user-interface, allows for management of other control systems, and is the primary user interface to the proton therapy system. Eventhough the system has changed significantly over the past years since its initial version in 2003, its overall design and purpose remains unchanged. It allows users to [11]:

- select patient treatment plans;
- download treatment and beam parameters of the selected field to other subsystems of the Therapy System;
- check and verify that all systems are ready for treatment;
- stay informed of the status of other subsystems;
- start and stop treatment;
- record and store treatment results in the Treatment Planning database;

Figure 2.2 depicts the software achitecture of the former MPRI Treatment Room Control System (TRCS). It is divided into five different groups; the patient data group which deals with patient and treatment information, the communications group for handling treatment requests to and from other Proton Therapy Systems (PTS), X-Ray System, Beam Delivery System (BDS) and Dose Deliver System (DDS) using TCP/IP sockets, the treatment management group for dealing with therapy treatment commands from the Radiotherapy Technologist (RTT), the maintenance group for testing and configuring the

system, and the data acquisition (DAQ) group for handling analog and digital input/output signals to and from the Kicker Enable System (KES) and the MPRI Interlock and Radiation System (MIRS) together with other control logic [11].

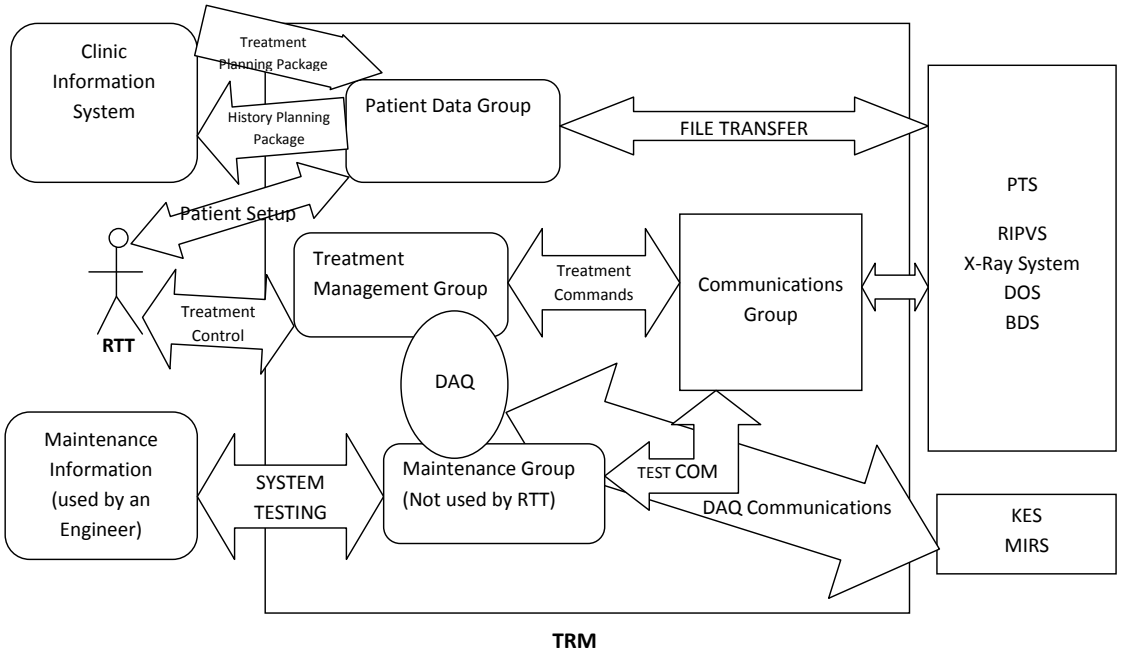

Figure 2.2: MPRI Treatment Room Manager

The TRIUMF Proton Therapy facility, through its Treatment Control System, has also achieved the same functionality. This system provides for monitoring of patient safety, controlling patient dose, and operator control [12]. The Treatment Control System is based on a VAX computer, Mitsubishi PLC, CAMAC PROM-based controller, standard NIM and CAMAC modules, and numerous custom-made devices.

Just like proton therapy at iThemba, TRIUMF has two operating modes for the controls: the 'Normal Mode' (almost analogous to `Physics Mode` at iThemba) which is responsible for development, calibration, and testing, and the 'Patient Mode' (analogous to iThemba's `Clinic Mode`) used for treatment.

Chapter 3

The New Proton Therapy Control System

The Proton Therapy Control System consists of a number of subsystems which collectively ensure that the beam delivery system is properly configured and functional, that the patient is well positioned for treatment that the treatment room is cleared and armed for treatment and that the beam is properly terminated after treatment.

This chapter briefly discusses all the subsystems which form the Proton Therapy Control System (as illustrated in Figure 3.1) and outlines how each is configured by the supervisory system.

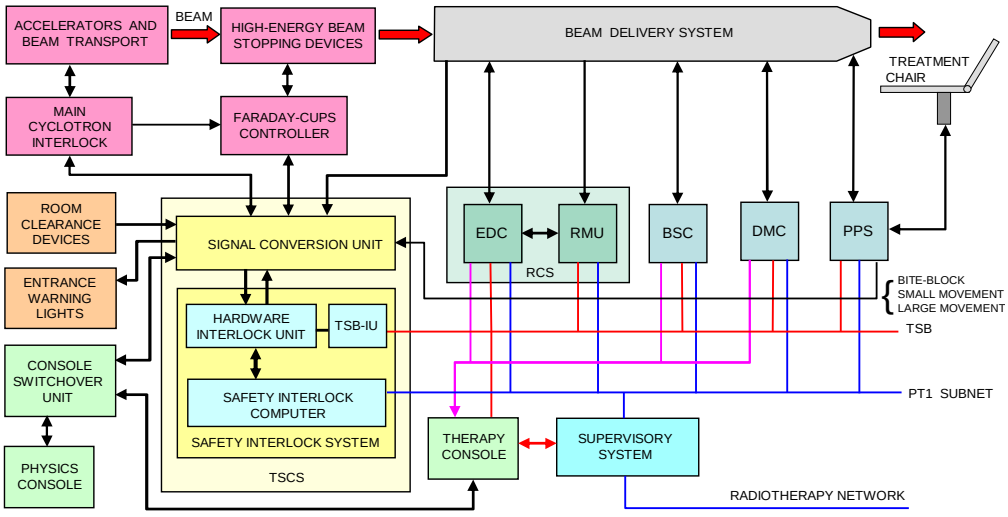

Figure 3.1: Schematic layout of the proton therapy control system

Section 3.1 covers the console system and shows how the supervisory system is interfaced to the Therapy Safety Bus. Section 3.2 discusses how safety is enforced in the Proton Therapy Control System through the Therapy Safety

Control System. The Range Control System follows in section 3 followed by the Beam steering controller in section 4. Section 5 discusses the Dose Monitor Controller and the chapter ends with section 6 which covers the Patient Positioning System.

3.1 Console System (CS)

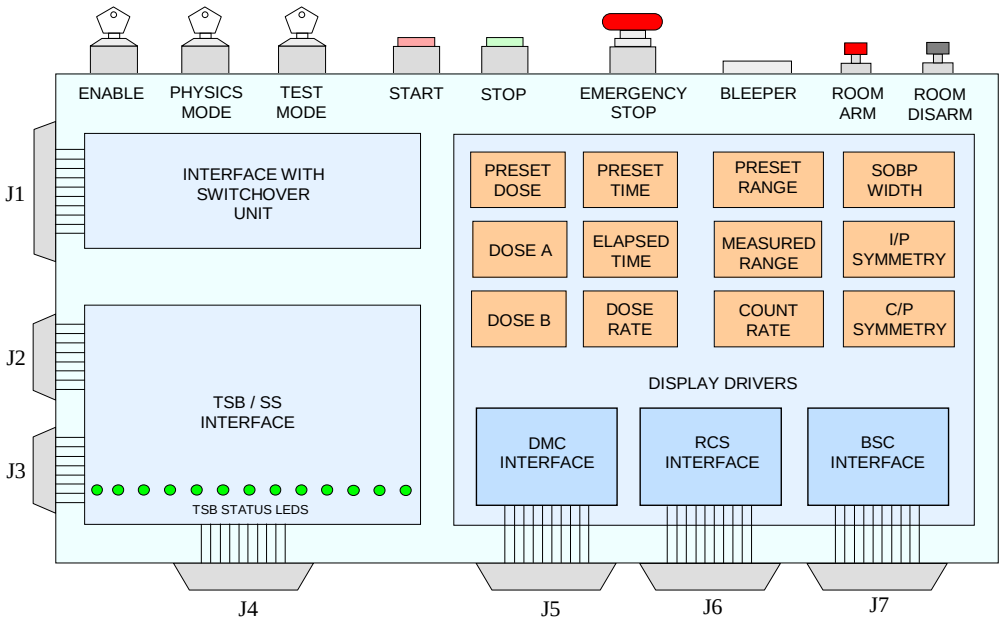

Figure 3.2: Therapy Console

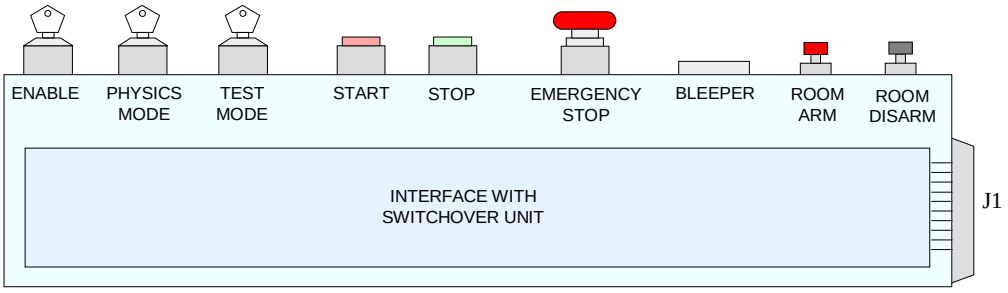

Figure 3.3: Physics Console

This system consists of the Therapy and Physics Consoles (Figures 3.2 and 3.3) which are used to manually start and stop treatment. Both consoles present an interface for inputs from the operator through a number of switches described below [5; 13]:

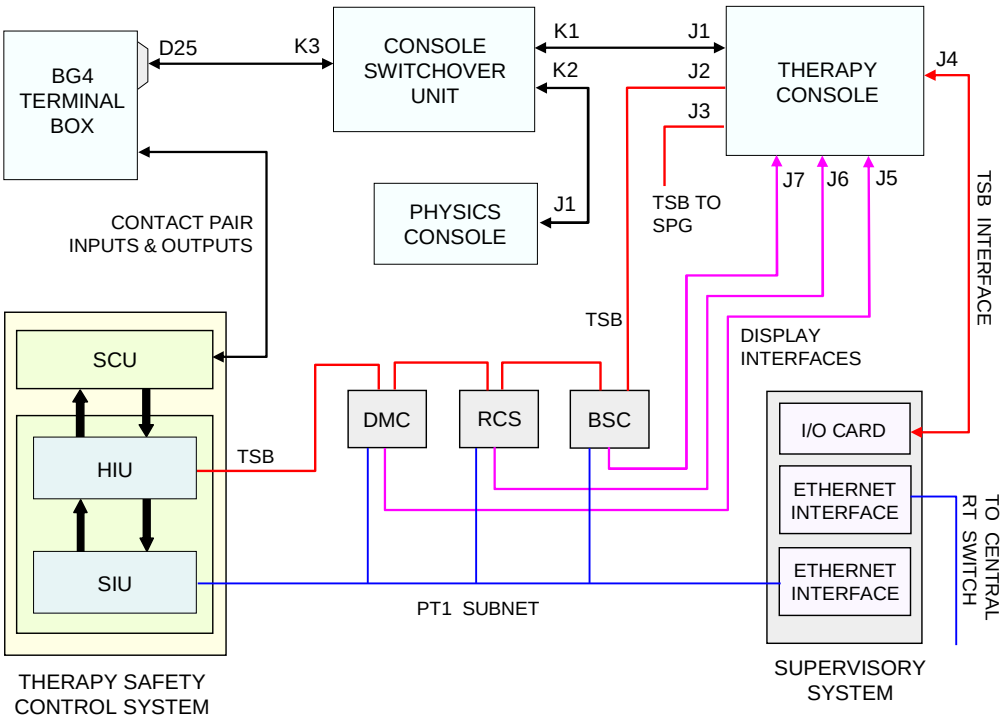

Figure 3.4: Console Switchover Unit

1. *Console enable* which activates the console (makes it the active console driving the TSB lines). This is an `ON/OFF` switch which is mutually exclusive between the Therapy and Physics consoles i.e. only one console can be switched ON at any given time, not both. To accomplish this both consoles have an identical lock for the *console enable* switch and only one key is used, which can only be removed when in the off position [13].

2. *Emergency stop* for switching off the irradiation and enforcing a complete shutdown of the acceleration of the beam. It causes an abrupt halt of the beam thus providing a more drastic method of stopping the beam and should only be used for emergencies. It is a red mushroom-type, latching push-button which is *normally closed* until the button is pressed. The switch mechanically latches in the open state once the button is pressed.

3. *Physics mode* for toggling the therapy control system between physics and clinical modes of operation. It is selectable from both the Therapy and Physics consoles and only one key is used between both consoles.

4. *Room disarm* for disarming the vault.

5. *Room arm* for arming the vault. This button switches on the BEAM ON warning light at the vault entrance to indicate that treatment is in process and no unauthorised entry.

6. *Start* which switches the irradiation on (if permissible). It is a non-latching push button which is illuminated as long as the beam is switched on.

7. *Stop* for switching off the irradiation. Just like the *Start* button, it is also a non-latching push button. The *Stop* button causes a gradual stop of the beam by inserting the Faraday cups and beam shutter into the beam.

The outputs of the switches of the consoles are connected to the therapy safety control system through the console switch-over unit. The purpose of the switch-over unit is to ensure that the outputs of the consoles are mutually exclusive, viz. only one console may be activated at a time [5]. Besides providing a means to start and stop treatments, the therapy console is also capable of displaying real-time information about the dose delivery and beam characteristics to the operator.

Most importantly, the therapy console provides an interface between the supervisory system and the therapy safety bus.

3.2 Therapy Safety Control System (TSCS)

This is arguably the most important subsystem of the Proton Therapy Control System. Its primary objective is to ensure the radiation safety of the patients and personnel and to switch the beam on and off. This system consists of the Safety Interlock System(SIS) and the Signal Conversion Unit(SCU). The Signal Conversion Unit links the SIS to a number of external devices (Figure 3.5) [5; 13];

- Interlock and safety devices such as numerous interlock switches in the beam delivery system and elsewhere in the treatment vault, as well as room clearance, arm and disarm switches and many other devices in the proton therapy facility.

- Consoles through the console Switch-over Unit.

- Faraday-cups controller which is an electronic crate controlling the high-energy beam stopping devices in the last section of the beam line to the treatment vault.

- Main cyclotron interlock system which is the interlock system for the accelerator equipment and beam transport lines.

- Other subsystems of the therapy control system which are mostly connected to the TSCS via the Therapy Safety Bus (TSB), while others are connected through hardware interfaces.

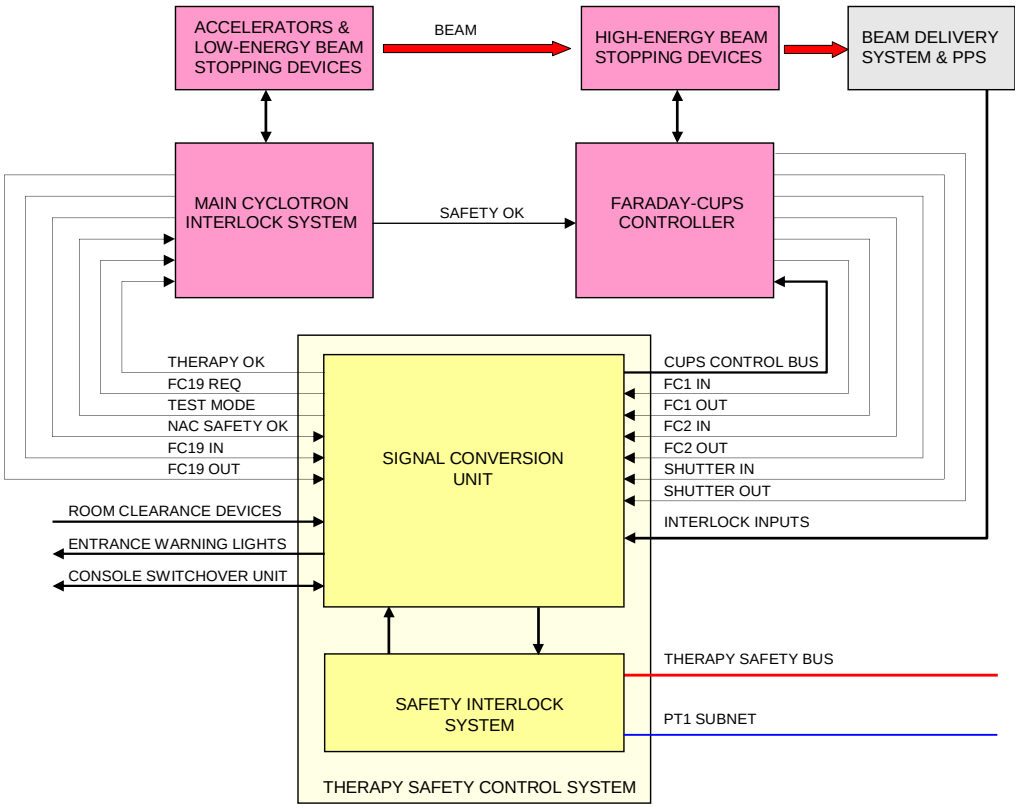

Figure 3.5: Therapy Control System Interfaces

Figure 3.7 illustrates the physical implementation of the TSCS. Both the SCU and SIU consist of 19 inch rack-mountable crates that house the specialised electronic components for these units. The SIS is implemented as a 19 inch rack-mountable SABUS system (). The hardware components of these units may briefly be summarized as follows:

- The SCU is equipped with nine line driver cards that are used to convert contact-pair signals to TTL signals, or vice versa. Six of these cards are 'input' cards, while the other three are 'output' cards. Each card provides eight line drivers, or signal conversion channels.

- The SIU is equipped with a 5V and 24V power supply unit, and a printed circuit board that implements the TSB current source unit, the voting logic unit and the cups control unit.

- The SIS is equipped with an ETX computer module and four SABUS I/O cards. Three of these I/O cards are configured as input cards, while the fourth is used as an output card. Each I/O card provides 24 digital input or output channels that are grouped into three ports of eight bits each.

As the name suggests, the Signal Conversion Unit provides signal converters that convert the contact-pair inputs from the external devices to single-ended TTL input signals for the safety interlock system and vice-versa as shown in Figures 3.5, 3.6, 3.7 and 3.8.

The safety interlock system consists of the Safety interlock computer attached to the hardware interlock unit. The safety interlock computer is an embedded computer module connected to the PT1 subnet whereas the HIU is connected to both the SCU and TSB.

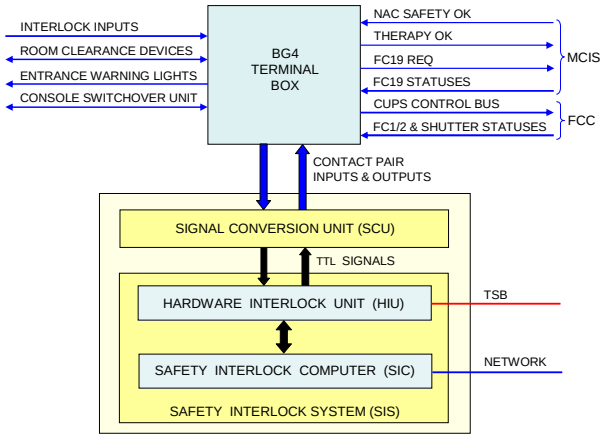

Figure 3.6: Therapy Safety Control System - Layout

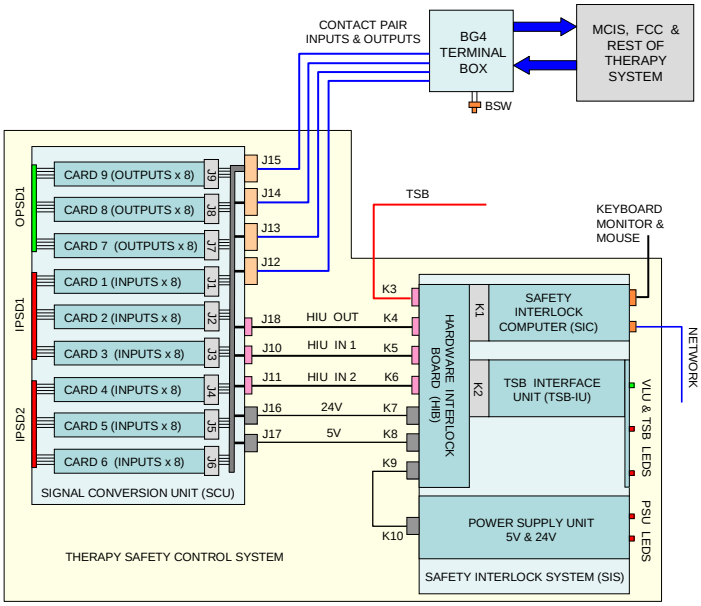

Figure 3.7: Therapy Safety Control System - Crates

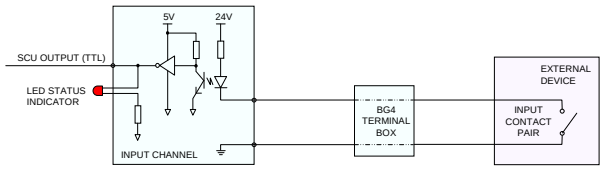

Figure 3.8: Signal Conversion Unit Input Channel

Table 3.1 depicts the hardware interlock unit with a description of its subunits.

Sub Unit	Description
Signal Interface Unit	An I/O interface device equipped with a number of TTL compatible I/O ports that are addressable by the safety interlock computer. The I/O ports are connected to some of the other subunits of the hardware interlock unit and most of the TTL data lines of the SCU thus allowing the safety interlock computer to interface to these subunits and the SCU.
TSB Current Source Unit	Provides the current sources that drive different lines of the TSB. Each of these circuits drives an LED that indicates the status of the line. This unit also converts the status of the TSB lines into TTL input signals to the safety interlock computer.
Voting Logic Unit	Its primary objective is to generate the output signals that cause the beam to be switched on and to ensure that the beam is automatically stopped when any interlock, functional or hardware failure is indicated by the TSB, the cyclotron interlock system or the safety interlock computer. It consists of the logical circuits that combine certain signals from the cyclotron interlock system and the faraday cups controller to determine whether the beam should be switched on and whether it is safe to do so.
Cups Control Unit	Is a microcontroller responsible for extraction or insertion of the high-energy beam stopping devices based on a single request signal from the voting logic unit thus simplifying the communication between the safety interlock computer and the faraday cups controller.

Table 3.1: Hardware Interlock Unit

The safety interlock computer uses software to process and draw logical conclusions from the input signals and to derive the necessary output signals

from these conclusions. Through the PT1 subnet, the safety interlock computer may at any time be prompted by the supervisory system to provide an instantaneous copy of its state (as determined by the input and output signals).

3.3 Range Control system(RCS)

This system consists of the energy degrader controller, double-wedge system and range monitoring unit which operate together to ensure that the beam has the required range during treatment.

3.3.1 Energy Degrader Controller (EDC)

This system consists of two synthetic graphite wedges mounted back-to-back on a drive mechanism (Figure 3.9). The wedges are driven in or out of the beam by a stepper-motor controlled by the EDC. The EDC uses input from the different sensors on the drive mechanism of the wedges to determine the relative position of the wedges, and through the use of a calibration curve, it relates the R_{50} range of the beam to the position of the wedges [5]. Through the use of the EDC sofware an operator can create or renew the calibration curve by entering the measured R_{50} ranges corresponding to different wedge positions.
The EDC obtains the required beam type, SOBP width and beam range for a particular treatment field from the supervisory system through the RPC network interface. Connection to the range monitoring unit is accomplished through a serial interface which is used to send the required beam type, SOBP width and beam range to the range monitoring unit before treatment starts. After treatment has started, the EDC uses the serial interface to receive real-time measurements of the beam range, which enables it to adjust the wedges so that the measured range continually matches the required range. Thus, the energy degrader controller, double-wedge system and range monitoring unit operate together as a closed-loop range control system to ensure that the beam has the required range during treatment (Figure 3.9).

3.3.2 Range Monitoring Unit (RMU)

The range monitoring unit is connected to the multi-layer Faraday cup (MLFC) of the new range monitoring detector and also to the rotation sensing unit of the range modulator assembly [5]. The MLFC is a measuring device consisting of a series of copper plates alternating with insulating layers of Lexan, which are concentric to the optical axis of the beam and have a small circular aperture in the middle. The MLFC is part of a cylindrical vacuum chamber located in the new range monitoring detector, and its copper plates are connected by

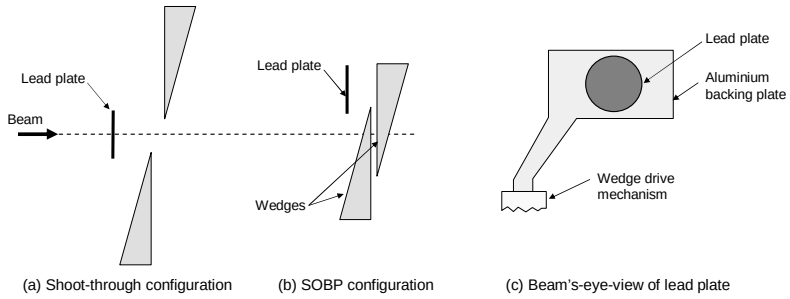

Figure 3.9: Double-Wedge System with scattering lead plate

thin insulated wires to a vacuum-tight multi-pin feedthrough connector that is mounted on the side of the vacuum chamber thus allowing the charges collecting on these copper plates to be conducted by low-noise signal cables to the multi-channel integrator circuits of the range monitoring unit. The range monitoring unit uses these signals from the MLFC to determine or measure the R_{50} range of the beam. The range monitoring unit uses the serial interface with the energy degrader controller to receive the required beam type, SOBP width and beam range before treatment is started, and to transmit the measured beam range during treatment. For SOBP beams it samples the beam range for each revolution of the range modulator wheel (see Figure 3.10). It uses the signals from the rotation sensing unit to determine when each revolution starts and stops. For shoot-through beams it uses an internal clock to determine the rate at which to sample the beam range.

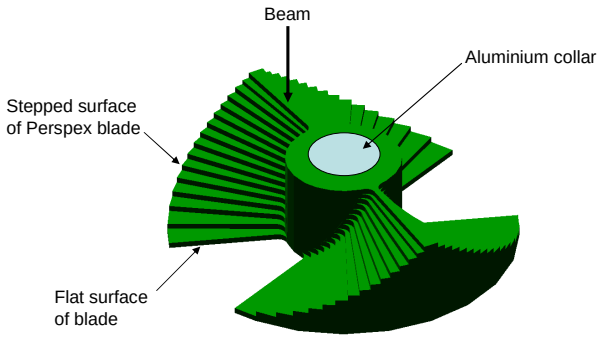

Figure 3.10: Range-Modulator wheel

3.4 Beam Steering Controller

This system has a network interface to the accelerator control network which allows it to keep the beam aligned with the optical axis of the beam delivery system. It achieves the beam alignment by using signals from the segmented

ionization chambers to determine the position and symmetry of the beam and then using that information to regulate the current through the coils of the two steering magnets and hence keeping the beam in place.

It is equipped with a power supply unit that provides the required high voltage to the segmented ionization chambers.The power supply uses a feedback circuit and a voltage comparator to verify the continuity of the circuit that supplies high voltage to the chambers. If the continuity of this circuit is interrupted, the beam steering controller uses the TSB to stop or prohibit the treatment.

This system uses multiplicative calibration factors to adjust the gain of each segment of the ionization chambers. These factors are used in the software of the system to increase or decrease the measured reading of each segment and new calibration factors can be entered by the operator through the software while the beam is turned on [5]. The beam steering controller provides a real-time graphical display of the beam symmetries and detector count rate, and uses the TSB to stop the treatment if any of the symmetry ratios deviates from unity by more than the allowed tolerance value. It obtains the required beam type, SOBP width and beam range for a particular treatment field from the supervisory system through its network interface to the PT1 subnet.

3.5 Dose Monitor Controller (DMC)

The primary objective of the DMC is to monitor the dose rate as well as the total dose delivered to the patient using signals from the two ionization chambers. It consists of two dose monitoring modules, one per chamber, which have internal clocks to measure elapsed irradiation time, and it has non-volatile recorders for displaying integral dose delivered should there be a power failure [5]. Each module has a power supply unit providing the required high voltage to the chambers. To verify the continuity of the circuit that supplies the high voltage to the chamber, the power supply unit uses a feedback circuit and a voltage comparator. If the continuity is interrupted, the DMC uses the TSB to stop or prevent treatment.

The operational parameters of the DMC may be set using either of two methods; through the PT1 network interface by the supervisory system or through the local keyboard by an operator. The parameters include the required treatment dose, maximum allowed dose rate and treatment time, and the calibration factors that should be used for the dose monitoring chambers [5].

This system uses the TSB to stop treatment should the preset dose or treatment time be reached, or if the dose rate exceeds the preset value for the maximum allowed dose rate. To display some operational parameters such as the preset treatment time and dose measurements, the DMC uses a display interface module.

3.6 Patient Positioning System

Patient positioning at iThemba LABS makes use of a motorised treatment chair to which the patient is fixed to an immobilization device. A multicamera-stereophotogrammetry (SPG) system is then used to position the treatment chair (hence the patient) so as to align the treatment beam with the tumour. The SPG system makes use of a custom-made marker-carrier fitted with radiopaque and retro-reflexive markers which is worn by the patient during treatment planning and irradiation so as to acquire the position of the tumour with respect to the markers and hence be able to align the tumour to the treatment beam by moving the chair. The marker-carrier is securely attached to the patient with the help of the bite-block vacuum system [8].

The following subsections briefly discuss all the units of the patient positioning system.

3.6.1 Bite-block Vacuum System

This system is responsible for securely attaching the marker-carrier to the patient. It achieves this by evacuating the air between the patient palate and the bite-block into a vacuum pump. This effectively causes the marker-carrier (affixed to the bite-block system) to attach to the patient and hence respond to any of the patient's movements.

It consists of a gauge, pressure regulator, a three-way valve, and an electrical control unit [5]. The control unit connects to a solenoid which actuates the valve. When the solenoid is activated, the three-way valve connects the bite-block to the vacuum-pump thus causing the bite-block to attach to the patient's palate.

The control unit operates the system using a number of switches as illustrated below.

The *pump on* switch turns the power to the vacuum pump on or off while the *vacuum on* switch latches two relays in the control unit. The first relay (labelled 6 in Figure 3.11) activates the solenoid of the three-way valve thereby connecting the air line from the bite-block to the vacuum pump. The other relay closes a contact pair in the control unit which provides a *bite-block interlock signal* to the safety interlock system indicating the successful connection of the bite-block to the vacuum pump. The *vacuum release* is a hand-held switch with which the patient releases the relays so that the solenoid of the three-way valve is deactivated causing the bite-block to detach. The last switch, *interlock override*, suspends the input to the safety interlock system thereby permitting treatments to proceed even when the vacuum system is not operational. When the supervisory system and the therapy safety bus have been fully implemented, the interlock override switch will be permanently removed.

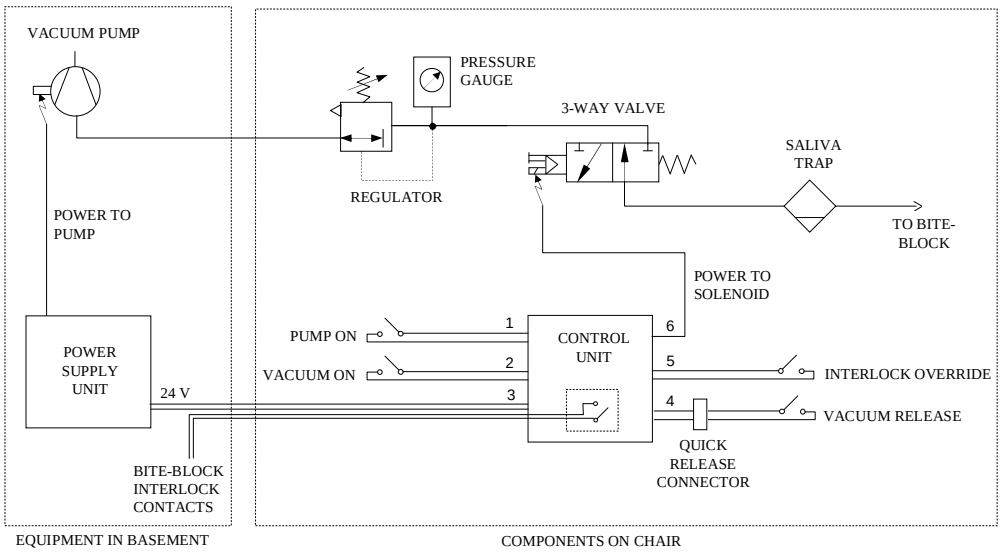

Figure 3.11: Bite-block Vacuum control system

3.6.2 Chair Control System

As the name suggests, this system controls the motorised treatment chair on which the patient is placed for treatment. It controls the five motors of the chair with stepper-motor control modules, one for each motor. The motors allow the chair to make translations in three orthogonal directions, a rotation about the vertical support pillar of the chair, and a backrest rotation, all totalling five degrees of freedom.

The chair control system is also equipped with a sixth stepper-motor control module which controls the motor that is responsible for the rotation of the treatment collimator. Each of these control modules is used to transmit low-voltage control signals to a translator driver, which translates these signals into the high-voltage pulses needed to drive the stepper-motor that is connected to it. The various limit switches of the chair are connected to the five chair control modules. These switches are used to determine the travel range of each axis of motion and to calibrate the chair [5; 7; 14; 15]. The angular position sensors of the treatment collimator assembly are connected to the collimator control module. These sensors are used to determine the angular position of the collimator and to calibrate the collimator rotations [5; 14].

The chair control system is also equipped with a hand-pendant that allows the operator to select between the different modes of operation of the system and to manually control the movements of the chair. When the system is switched to the manual mode, the hand-pendant may be used to issue specific chair movement commands to the system. When the system is switched to the SPG mode, it operates under the control of the SPG system through the network interface. The hand-pendant may also be used to align the chair, to lower it

beneath the floor of the treatment vault, or to raise it above the floor. The chair alignment procedure enables the system to locate the reference position for each axis of the chair (using the limit switches).
Furthermore, the chair control system provides an emergency stop function that may be used to stop the motions of the chair and collimator at any time, irrespective of the mode in which the system is operating. This function is immediately executed when any one of the two emergency stop switches on the chair, or the emergency stop button of the hand-pendant, is pressed [14]. It provides two main functions while operating in the SPG mode, namely the patient alignment function and the chair talk function [14]. In brief, the purpose of the patient alignment function is to use the input from the SPG system to calculate the chair motions and collimator rotation that will result in the correct treatment setup of the patient, and then to execute these motions when instructed to do so by the SPG system. The purpose of the chair talk function is to execute arbitrary chair and/or collimator motions as specified by the SPG system.

3.6.3 Stereophotogrammetric (SPG) System

It is a calibrated multi-camera system capable of acquiring and processing synchronized video images from any three of its nine CCD cameras. The cameras are calibrated by means of a special calibration cube that can be accurately positioned at the treatment isocenter. The SPG program allows the operator to select three suitable cameras to be used during treatment. The suitability of the cameras depends on the required direction of the treatment beam relative to the patient. The program uses the selected cameras to acquire video images of the marker carrier that is attached to the patient so as to use image processing and stereophotogrammetry techniques to derive the treatment room coordinates of those markers that are visible in the video images. These coordinates are then used to calculate the transformation matrix between the patient coordinate system and the beam coordinate system [16]. It then transmits this transformation (consisting of a translation matrix and a rotation matrix), as well as the basic parameters of the treatment beam, to the chair control system as part of the instruction to execute the patient alignment function using RPC communication [5; 14].

Through RPC network communication the SPG program can also instruct the portal radiographic system to acquire radiographs of the patient and to determine the possible errors in the treatment setup from these images. These errors are send back to the SPG program so that it may apply the necessary corrections to the treatment setup [5; 14]. During irradiation it uses video streams from the selected cameras to monitor the patient, and if the patient moves out of treatment position it stops the beam by sending the necessary signal to the safety interlock system using one of two relay contacts (small

movement and large movement relays). For the beam to be switched on both relays must be closed, and when a small patient movement occurs, the SPG program stops the beam by opening the small movement relay, whereas both the small movement and large movement relays are opened when a large movement occurs [5].

3.6.4 Portal Radiographic (PR) System

It consists of an x-ray imaging unit and an image registration system. The x-ray imaging unit is used to acquire portal radiographs of the patient, while the image registration system uses these images to estimate the errors in the patient treatment setup and hence to verify the correctness thereof [5; 16]. The major components of the x-ray imaging unit and their respective functions may be briefly summarized as follows:

- X-ray tube: provides the exposures needed for x-ray imaging. It is a component of the beam delivery system and can be driven pneumatically in and out of the beam path.
- X-ray generator: it is the high-voltage switching power supply for the x-ray tube. It is equipped with a control desk that allows the operator to set the required exposure parameters and to initiate the exposure sequence by pressing the button of a hand-held switch that is connected to the desk.
- Flat-panel detector: serves as the electronic portal imaging device (EPID).
- Image acquisition computer: accepts the raw image data from the EPID and applies various corrections to the raw images to produce the portal radiographs.
- Synchronization interface module: it is a special module of the x-ray generator that allows the operation of the EPID to be synchronized with the exposure sequence of the generator so that the image accumulation period properly overlaps with the x-ray exposure period.
- X-ray hoist mechanism: allows the EPID to be inserted into the beam path downstream from the patient or to be retracted from this position.
- X-ray control unit: drives the stepper-motor of the hoist mechanism. It also serves as an interlock for the x-ray imaging unit to prohibit any x-ray exposures when the EPID and the x-ray tube are not correctly positioned for the acquistion of the portal radiographs.

The image acquisition computer is equipped with the following software and hardware [5]:

- Image acquisition software that provides the functions needed to acquire the raw images and to apply corrections to the raw images to compensate for line noise, detector pixel defects, and variations in the bias offsets and gains of the detector pixels. It also allows for the acquisition of the calibration images and defect maps that are needed for this purpose, and to save this calibration data on the system.

- Software support for an application programming interface (API) that allows a remote computer to call, via a network link, the image acquisition and calibration functions and to transfer the corrected images (i.e.,the portal radiographs) to the remote computer.

- A data acquisition card that provides the data interface with the EPID. This card sends the necessary control signals to the EPID and allows the image data to be downloaded to the image acquisition computer.

- An I/O card that provides the input and output channels needed to synchronize the operation of the EPID with the exposure sequence of the x-ray generator. The synchronization interface module provides the hardware interface between the x-ray generator and the I/O card, while the image acquisition software provides a software interface between the I/O card and the data aquisition card to which the EPID is connected. The link between the synchronization module and the I/O module passes through the x-ray control unit to provide the necessary interlock function as described above.

The image registration system is a multi-processor computer equipped with specialized software to perform the following tasks [5]:

- Communicate with the image acquisition computer and the SPG system via the network interface.

- Calibrate the x-ray imaging unit. These calibration functions are used to determine the transformation matrix between the coordinate system of the fudicials (special calibration shapes on the calibration cube) of the x-ray imaging unit and the coordinate system of the beam line, which is needed for the proper and accurate positioning of a patient.

- Construct pre-operatively the 'light slab' data that are used during treatment to efficiently generate digitally reconstructed radiographs (DDRs) of the patient in slightly different treatment positions. The light slabs are constructed from the CT data of the patient. A separate light slab is created for each field in the treatment plan.

- Upon instruction by the SPG system, acquire portal radiographs of the patient via calls to the image acquisition computer.

- Estimate the errors in the treatment setup for the given treatment field and communicate these errors to the SPG system. The errors in the treatment position of the patient are estimated by registering DRRs for different positions of the patient against the portal radiograph showing the observed position and searching for the DRR (and hence the position) that gives the optimal match. The differences between this optimal position and the required position express the errors in the treatment position of the patient. The errors in the orientation of the treatment collimator are estimated by using a contour matching algorithm to calculate the difference between the observed and required orientations of the collimator aperture [5].

Chapter 4

Therapy Safety Bus Simulation Rig

TSB Simulation Rig is a proof-of-concept prototype for the actual therapy safety bus, which has been implemented so as to simulate all interlock lines for the proton therapy system in order to test the TSB configuration functionality of the supervisory system. It is a custom-made 'box' equipped with control switches, connectors and status LEDs as will be described in the following sections. Section 4.2 describes the operational specifications of the rig followed by section 4.3 which describes all the rig's physical components.

4.1 Design

Figure 4.1 depicts the schematic layout of the TSB rig showing how all the connectors, switches and status LEDs are connected together, and how the entire box is interfaced to a PC. An Eagle Technology PCI 848 I/O card is used to exchange information between the PC and the rig thus enabling the PC (supervisory system) to sense and control interlock lines. With regard to the supervisory system, the SHV and BNC connectors are irrelevant hence will not be mentioned in this discussion.

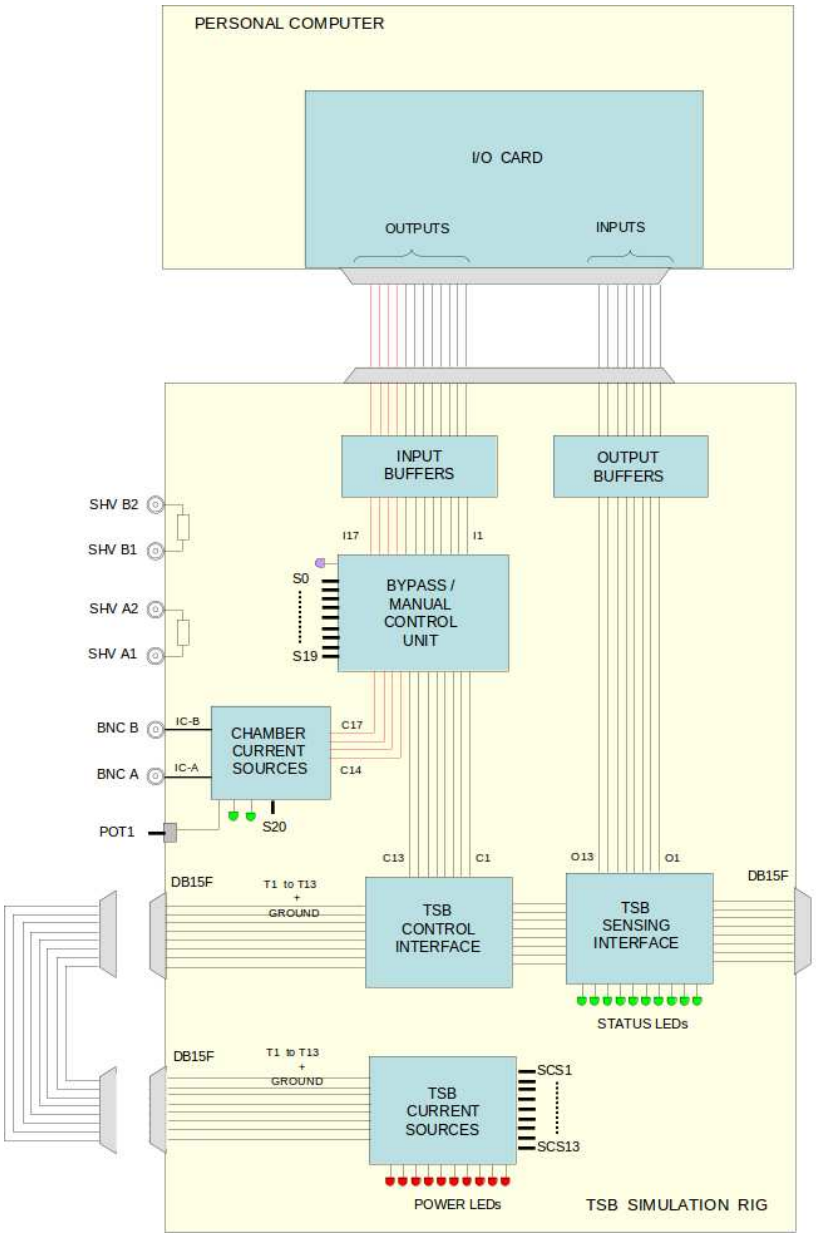

Figure 4.1: Therapy Safety Bus Simulation Rig

4.2 Operational Specifications

The TSB simulation rig consists of thirteen (13) current source lines of 15mA each, two (2) chamber current source lines of 200nA each whose current can also double to 400nA, chamber leakage current of 100pA per chamber, a PC I/O interface, and a High Voltage (HV) feed-through.

4.3 Description

The box consists of four panels; front, rear, left and right panels. The front panel, which is the most populated panel, hosts 13 switches for the 13 current sources for each of the TSB lines (CS 1−13). It also has 13 red LEDs to indicate the status of each CS line, whereby an On state of the LED denotes that the line is active. Also present on the front panel are 13 switches which control each TSB line [CON $1-13$] and 13 green LEDs to indicate the status of each control line. There are also 3 switches for the chamber current sources [CHAM, A_EN, B_EN] together with 3 green LEDs to indicate their status. 3 switches for the chamber current x2 Control [x2_EN, Ax2, Bx2] with 3 accompanying green LEDs to indicate their status can also be found on the front panel. The chamber current x2 control switches are for doubling the current of either chamber current source or both to 400nA. Sensing of the TSB lines [SENSE $1-13$] is indicated by yet 13 more green LEDs, and lastly 1 amber LED indicates whether the front panel is selected [BP], whereby On means front panel is selected while Off denotes PC selection (for I/O operations).

The rear panel only consists of 1 power connector including a switch and fuse, and 1 PC interface connector (DB25).

The left panel is dedicated to treatment simulations and consists of 1 TSB line CS output connector con1 for simulation purposes, 1 TSB line input connector con2 from simulation current sources, or from system, and 1 function select switch to select between front panel or PC simulation. When front panel simulation is selected, the BP LED on the front panel lights up.

Lastly, the right panel hosts 1 TSB CS output connector CON3 to System, 2 BNC connectors for chamber simulation, and 4 SHV connectors for HV simulation.

Chapter 5

Supervisory System Design and Implementation

This chapter is about the design and implementation of the new supervisory system for the proton therapy facility at iThemba LABS. It starts off with section 5.1 which outlines functional and non-functional requirements of the supervisory system followed by section 5.2 which discusses the design of the system. The chapter ends with a discussion of how the first version of the supervisory system was implemented.

5.1 Requirements Analysis

5.1.1 User Requirements

The Supervisory System is the central system of the proton therapy treatment unit that coordinates and configures all other systems with beam parameters and treatment information needed to ensure a correct and safe irradiation of a patient during treatment. It also collects, records and verifies all the necessary treatment information after any successful or otherwise irradiation session. Furthermore, it acts as a gateway/proxy between all other subsystems of the proton therapy control system and the radiotherapy data-store server connected to the central switch of the radiotherapy network [5; 17]. Any system which requires patient-specific information from the data-store server does so by issuing a request to the supervisory system which in turn authenticates the system and retrieves the data from the server on the requesting system's behalf. This way, access to shared data is well managed, there is a high immunity against deadlocks and there is a reduced possibility of ending up with incorrect data in the database [18].

5.1.2 System Requirements

The functional requirements of the new supervisory system are outlined below [19];

- User and attribute-based access control: each and every user of the system (medical physicists, radiation therapists and treatment planning engineers) will be authenticated and granted access to only the functionality they are authorised to use.
- Use of a bar-code scanner to identify treatment components in order to minimize the possibility of human error during the treatment process.
- Communication with the patient data-store to retrieve treatment files and records so as to load patient treatment information. Also to retrieve patient-specific files on behalf of other subsystems of the proton therapy control system. With the exception of the treatment record, all files are generated externally from the supervisory system (i.e. are created by other systems such as the treatment planning software), thus no editing of patient data will be allowed by the supervisory system except for updates on treatment records.
- Communication with the DMC to send configuration information such as CVolt factors, and temperature and pressure readings.
- Communication with the SPG system to send configuration information and act as a proxy between the SPG and data-store.
- Communication with the PR (X-ray) system to send configuration information and act as a proxy between the PR system and data-store.
- Communication with the EDC to send configuration information.
- Communication with the BSC to send configuration information.
- Configure the Therapy Safety Bus so as to signal whether it is safe to administer treatment and whether other systems have been configured successfully.
- Communication with the SIS so as to act as its slave monitor thus displaying statii of interlock lines.

Figure 5.1 illustrates interaction of the supervisory system with its major actors through a UML use-case diagram.

The non-functional requirements of the supervisory system are the organizational requirements .vis those requirements specific to iThemba LABS Medical Radiation and constraints on its software projects. These requirements involve implementation-specific issues as well as interoperability issues with

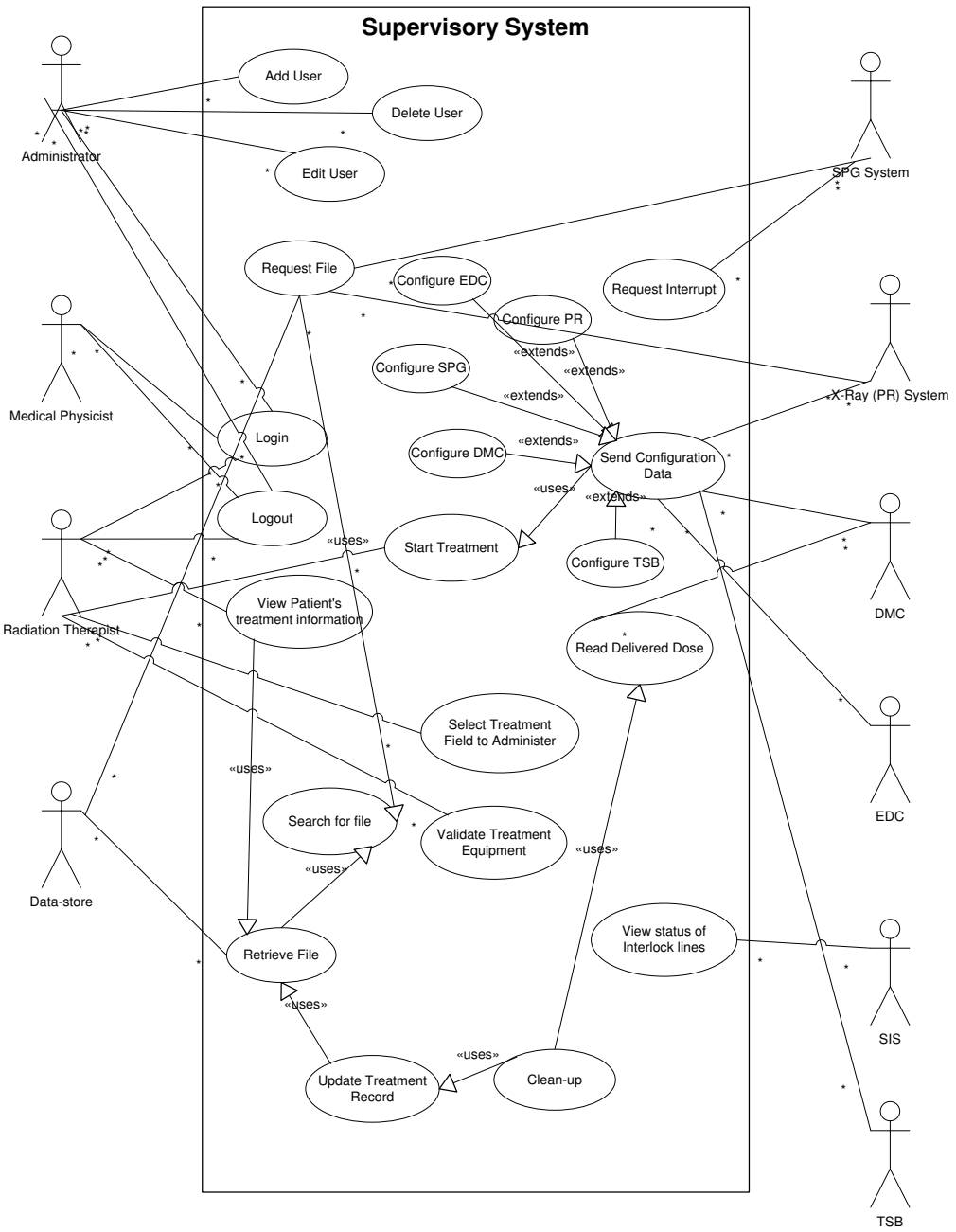

Figure 5.1: UML Use-case Diagram - Supervisory System

existing software [20; 21]. Table 5.1 outlines the major non-functional requirements of the supervisory system [20].

Non-functional Requirements
The GUI of the supervisory system shall be implemented using Qt/C++ widgets set. This is because Qt provides signal/slot mechanism which allows for simple inter-process and event-driven communications. All functionality of the system will be through menu items and user-centric forms. Also, Qt/C++ is the prefered developement environment because it provides open source application building tools and can easily be integrated into the entire proton therapy control system.
The network communication with other subsystems shall be through ONC-RPC method calls. This allows for a well distributed system using TCP/IP socket programming as well as a robust client-server architecture.
Configuration of the Therapy Safety Bus shall be via a PCI interface on the supervisory system computer connected to an ETX-SABUS system. The plug-in board of the PCI interface should have opto-isolated outputs to allow for interfacing capability with the TSB.

Table 5.1: Non-functional Requirements

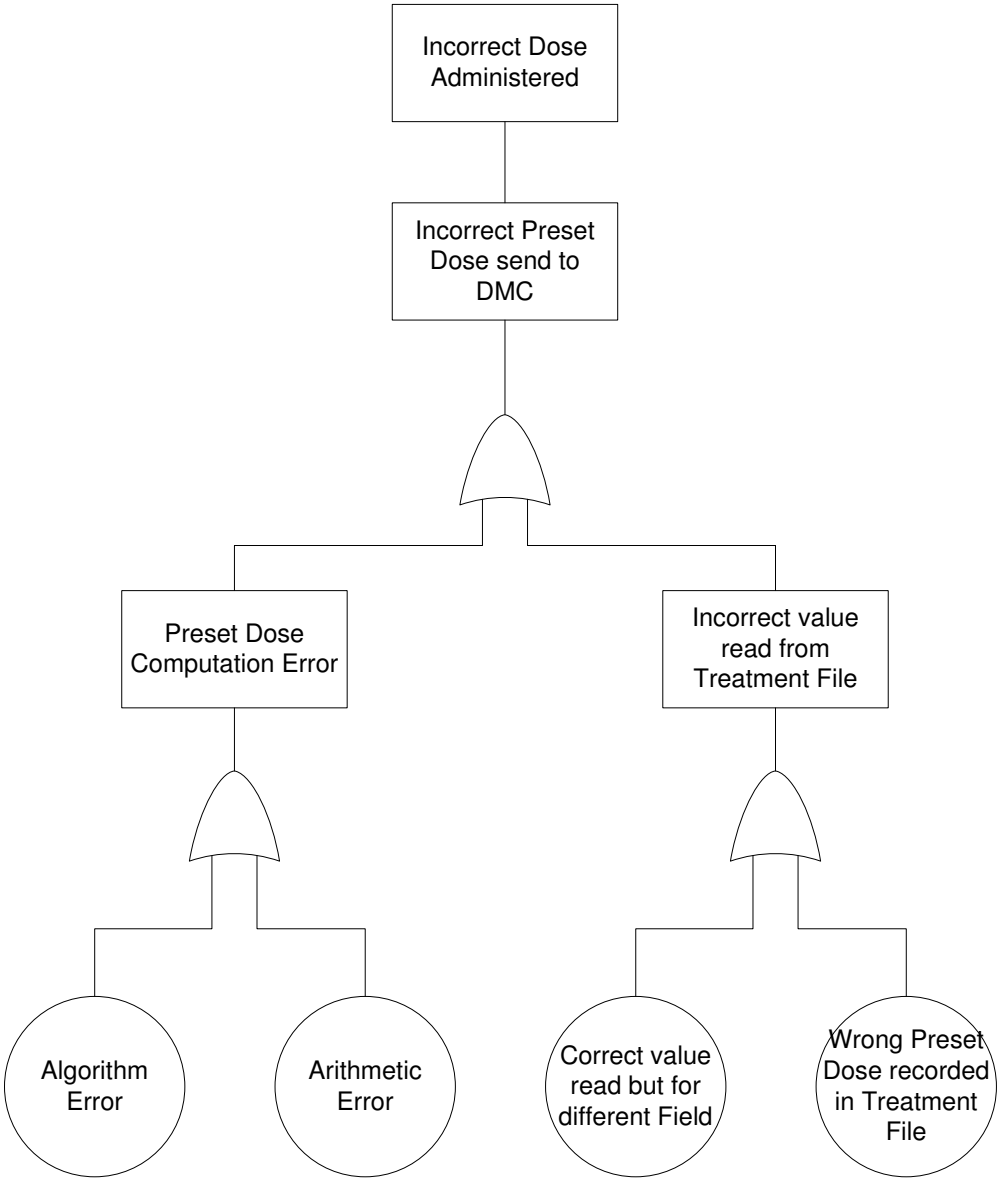

Figure 5.2: Fault tree Diagram - Supervisory System

Fig 5.2 highlights some of the unwanted hazards/risks pertaining to the supervisory system in the form of a fault tree analysis. Proton underdose and overdose represent a single hazard, namely, 'incorrect dose administered', and a single fault tree follows. States that can lead to the 'incorrect dose' hazard are then linked with 'or' symbols to denote that any combination of the risks can lead to the hazard.

5.1.3 Domain Requirements

These requirements are derived from the application domain of the supervisory system, namely, proton therapy. In proton therapy control systems, dosimetry, or the exact measure (and control thereof) of the amount of radiation administered to a patient is of paramount importance. If the dosimetric quantities are defined as follows [1; 17; 22; 23; 24];

N_p = The number of a treatment plan. A given patient can have more than one treatment plan. Thus, the numbers N_p are needed to differentiate between these plans. The plan numbers always lies in the interval $N_p \in [101, 999]$.

N = Total number of treatment fields for a given treatment plan, with $N \leq 99$.

N_u = Number of treatment fields from a given treatment plan that should be delivered by a specific treatment unit. Thus, $N = \sum_u N_u$ and $N_u \leq 99$.

M_u = Number of treatment fractions for *all* those fields from a given treatment plan that should be delivered by a specific treatment unit, with $M_u \leq 99$.

α = The field index $\alpha \in [1, N_u]$ that sequentially enumerates the treatment fields as they appear in a treatment file.

B_α = The beam number of a treatment field having the field index α in a treatment file, with $B_\alpha \in [1, 99]$. The numbers B_α do not necessarily start at one and do not necessarily form a contiguous or even ordered sequence as a function of the field index α. The beam number for a specific field in the treatment plan is unique, thus no two fields in the treatment plan can have the same beam number. Thus, for a particular treatment file, the relationship between B_α and α is one-to-one and therefore unique.

κ = The index that specifies the treatment fraction under consideration for a particular treatment unit. This number is always restricted to the interval $\kappa \in [0, M_u]$. The index $\kappa = 0$ always represents the patient simulation fraction during which no real treatment is done (i.e., no doses are administered). The real treatment fractions have indices $\kappa = 1, \ldots, M_u$.

D_α = Total prescribed dose to be administered by the treatment field with beam number B_α and field index $\alpha \in [1, N_u]$.

d_α = Prescribed dose per fraction for the field α, with $d_\alpha = D_\alpha / M_u$ given in Monitor Units (MU).

Δt_α = Prescribed treatment time per fraction for the field α. This is the maximum time, measured in minutes, that the beam is allowed to be switched on in order to deliver the dose d_α.

$d^p_{\alpha,\kappa}$ = Preset dose for the field α during an irradiation session of the treatment fraction κ, with $\kappa \in [1, M_u]$. It is the number of MU sent by the supervisory system to the DMC, via the `IN SETDOSE` command, to set the dose that should be delivered by the field α during a particular irradiation session of the treatment fraction κ.

$\Delta t^p_{\alpha,\kappa}$ = Preset treatment time for the field α during an irradiation session of the treatment fraction κ, with $\kappa \in [1, M_u]$. It is the time, in minutes, that is sent by the supervisory system to the DMC, via the `IN SETTIME` command, to set the time that the beam is allowed to be switched on to dilver the dose $d^p_{\alpha,\kappa}$. It is calculated as $\Delta t^p_{\alpha,\kappa} = \mathrm{Max}((d^p_{\alpha,\kappa}/d_\alpha)\ \Delta t_\alpha,\ \Delta t_{\min})$, where the constant $\Delta t_{\min}$ is defined below. The function $\mathrm{Max}(x, y)$ returns the largest value of its two arguments x and y.

$\Delta t_{\mathbf{min}}$ = The smallest value for the treatment time, given in minutes, that may be sent via the `IN SETTIME` command to the DMC.

$d^r_{\alpha,\kappa}$ = Actual dose delivered by the field α during the irradiation session for which the preset dose was given by $d^p_{\alpha,\kappa}$. It is the dose measured by the DMC *i*mmediately after the irradiation session is terminated. This termination may occur long before the preset dose $d^p_{\alpha,\kappa}$ is reached, whereupon $d^r_{\alpha,\kappa} \ll d^p_{\alpha,\kappa}$. Upon normal termination of the irradiation session, $d^r_{\alpha,\kappa} = d^p_{\alpha,\kappa} + \Delta d_{\alpha,\kappa}$, with $\Delta d_{\alpha,\kappa} \gtrsim 0$.

$d^m_{\alpha,\kappa}$ = Approximate value for $d^r_{\alpha,\kappa}$ as given by the mechanical counters of the DMC.

$d^s_{\alpha,\kappa}$ = The cumulative value of the actual doses delivered over all the irradiation sessions needed to complete the administering of field α for the treatment fraction κ.

$\Delta d_{\alpha,\kappa}$ = Dose overrun during the *n*ormal termination of an irradiation session for which the preset dose was given by $d^r_{\alpha,\kappa}$.

Δd^{th}_α = A theoretical estimate for the dose overrun $\Delta d_{\alpha,\kappa}$. This estimate is the same for all treatment fractions $\kappa \in [1, M_u]$ and only depends on nominal dose-rate that is required for the field α.

Γ = A predefined fraction that is used to specify whether the dose difference $d^p_{\alpha,\kappa} - d^r_{\alpha,\kappa} > 0$ should be eliminated by resuming the current fraction κ, or by distributing this difference evenly over the remaining fractions $\kappa + 1, \ldots, M_u$. Upon a premature termination of the beam, the dose difference $d^p_{\alpha,\kappa} - d^r_{\alpha,\kappa}$ should only be allowed to be evenly distributed over the remaining fractions when $d^p_{\alpha,\kappa} - d^r_{\alpha,\kappa} \leq \Gamma\, d_\alpha$, otherwise this difference should be eliminated by completing the the fraction κ.

$\dot{D}^{\mathbf{nom}}_{\mathbf{ref}}$ = Nominal dose rate for the reference field, given in MU/minute. The reference field is that field that is used for absolute dosimetry measurements. The nominal dose rate $\dot{D}^{\text{nom}}_{\text{ref}}$ expresses the ideal beam current in terms of a dose-rate measurement.

$\dot{D}^{\mathbf{max}}_{\mathbf{ref}}$ = Maximum dose rate allowed for the reference field, given in MU/minute.

O_α = Output factor for the field α. This is the ratio of the dose rate for the field α to the dose rate for the reference field under identical beam current conditions.

$I^{\mathbf{nom}}_\alpha$ = Ideal beam current required for the field α during any irradiation session, given in units of ηA.

$\dot{D}^{\mathbf{nom}}_\alpha$ = Ideal dose rate required for the field α during any irradiation session, given in MU/minute. It can be calculated as $\dot{D}^{\text{nom}}_\alpha = O_\alpha\, \dot{D}^{\text{nom}}_{\text{ref}}$.

$\dot{D}^{\mathbf{max}}_\alpha$ = Maximum dose rate required for the field α during any irradiation session, given in MU/minute. It can be calculated as $\dot{D}^{\text{max}}_\alpha = O_\alpha\, \dot{D}^{\text{max}}_{\text{ref}}$. The value of $\dot{D}^{\text{max}}_\alpha$ defines the maximum dose rate that will be tolerated by the DMC before it will terminate the beam. This value is sent by the supervisory system to the DMC via the `IN MAXRATE` command.

$\%\Delta_{\mathbf{AB}}$ = The maximum allowed dose difference, expressed as percentage, that the DMC will between the readings from the dose monitors A and B before it will terminate the beam.

$\Delta_{\mathbf{AB}}$ = The maximum allowed dose difference, expressed in terms of MU, that the DMC will tolerate between the readings from the dose monitors A and B before it will terminate the beam. This dose difference, calculated as $\Delta_{\text{AB}} = \text{Max}(\%\Delta_{AB}\, d^p_{\alpha,\kappa}/100,\, \Delta^{\text{min}}_{AB})$, is sent by the supervisory system to the DMC via the `IN DOSD` command. The constant $\Delta^{\text{min}}_{\text{AB}}$ is defined below.

$\Delta d^{\mathbf{min}}$ = The smallest value for the maximum allowed dose difference, given in MU, that may be sent via the `IN DOSD` command to the DMC.

CV_1 This is the CVOLT factor that adjusts the gain of the dose monitor A to compensate for the daily air-density changes in the monitor ionization

chambers that is caused by variations in temperate and pressure. It sent by the supervisory system to the DMC via the `IN CVOLT` command.

CV_2 Same as for CV_1, but for dose monitor B. It is sent by the supervisory system to the DMC via the `IN CVOLT2` command.

$\Delta t_{\mathbf{sys}}$ = The time taken by the beam delivery system to stop the beam when the preset dose is reached. It is given in minutes and includes the response time of the DMC to transmit the beam stop request after the preset dose is reached, as well as the time needed by the the Faraday-cup control system to accomplish this request. It is safe to assume that Δt_{sys} is independent of the beam current. See Addendum A for the measurement of Δt_{sys}.

Then the supervisory system's dose algorithm should aim to achieve the following two goals for each treatment field α [17; 19]:

- Minimize the differences $|d^s_{\alpha,\,i} - d_\alpha|$ separately for each fraction $i = 1, \ldots, N$, but allowing for the dose difference $d^p_{\alpha,\,i} - d^r_{\alpha,\,i} > 0$ to be distributed over the remaining fractions when $d^p_{\alpha,\,i} - d^r_{\alpha,\,i} \leq f\, d_\alpha$.

- Minimize the difference $|D_\alpha - \sum_{i=1}^{N} d^s_{\alpha,\,i}|$.

5.2 Supervisory System Design

This section outlines the design of the new proton supervisory system highlighting the execution flow the program will follow when performing its tasks. It is organized as follows; the section begins with a discussion of the architectural design of the system in section 5.2.1. The top-level flow of the proton supervisory system is illustrated in Figure 5.4 and explained in section 5.2.2 showing the role played by the supervisory system in the processes followed for the proton treatment of a patient.

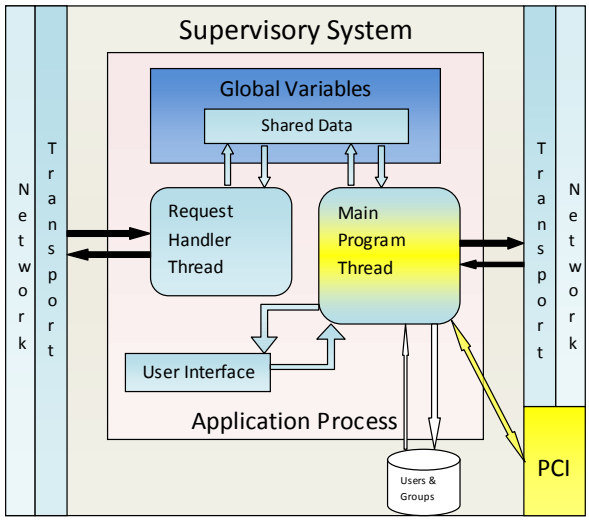

Figure 5.3: Architectural Design

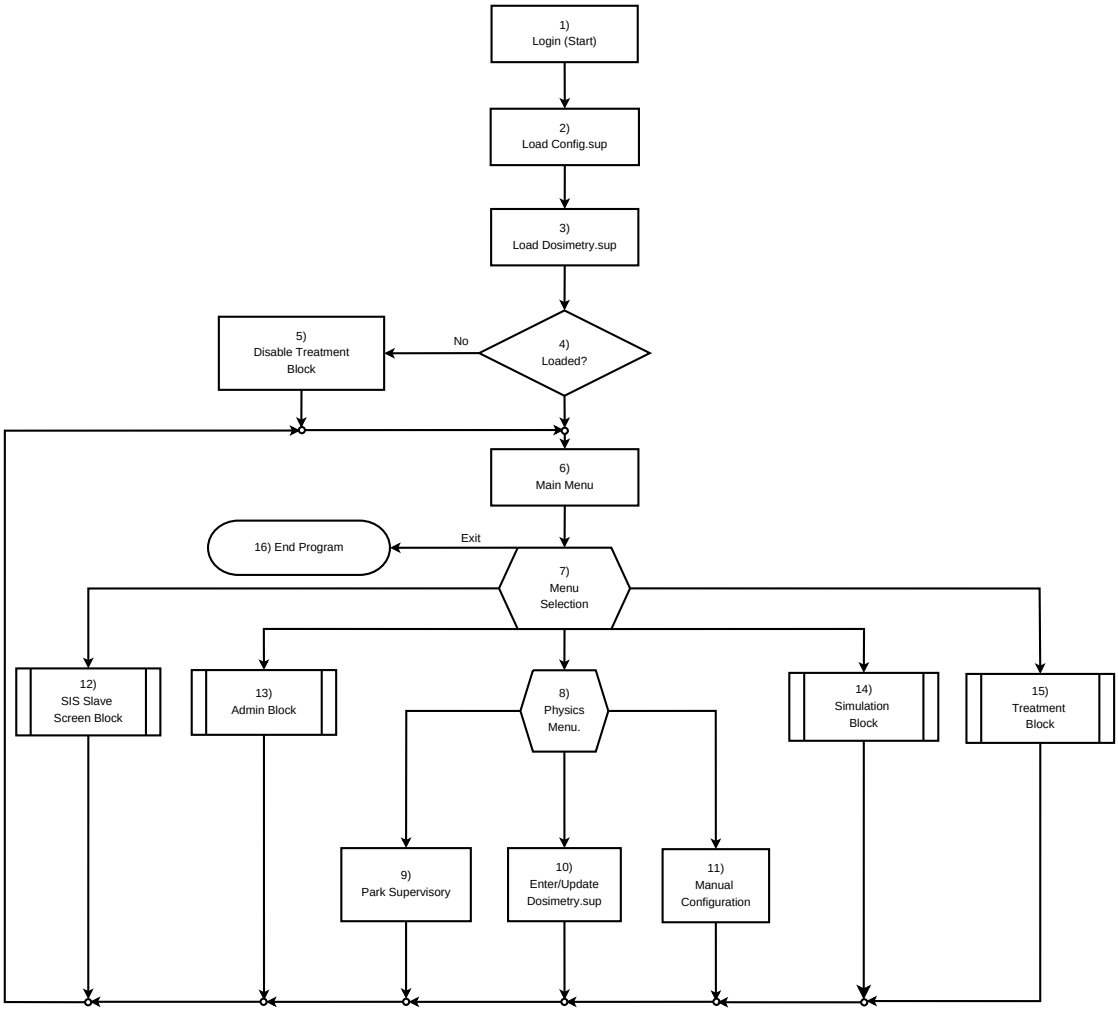

Figure 5.4: Supervisory System Top-Level Program Flow

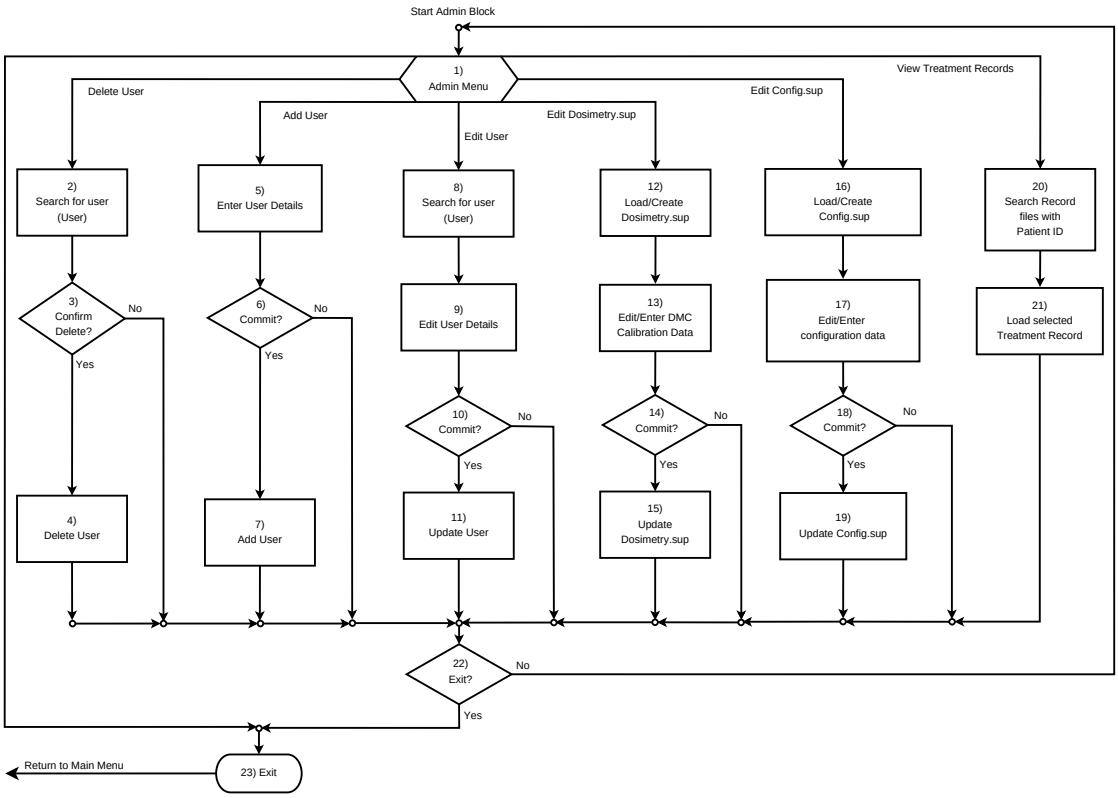

Figure 5.5: Supervisory System Admin Block

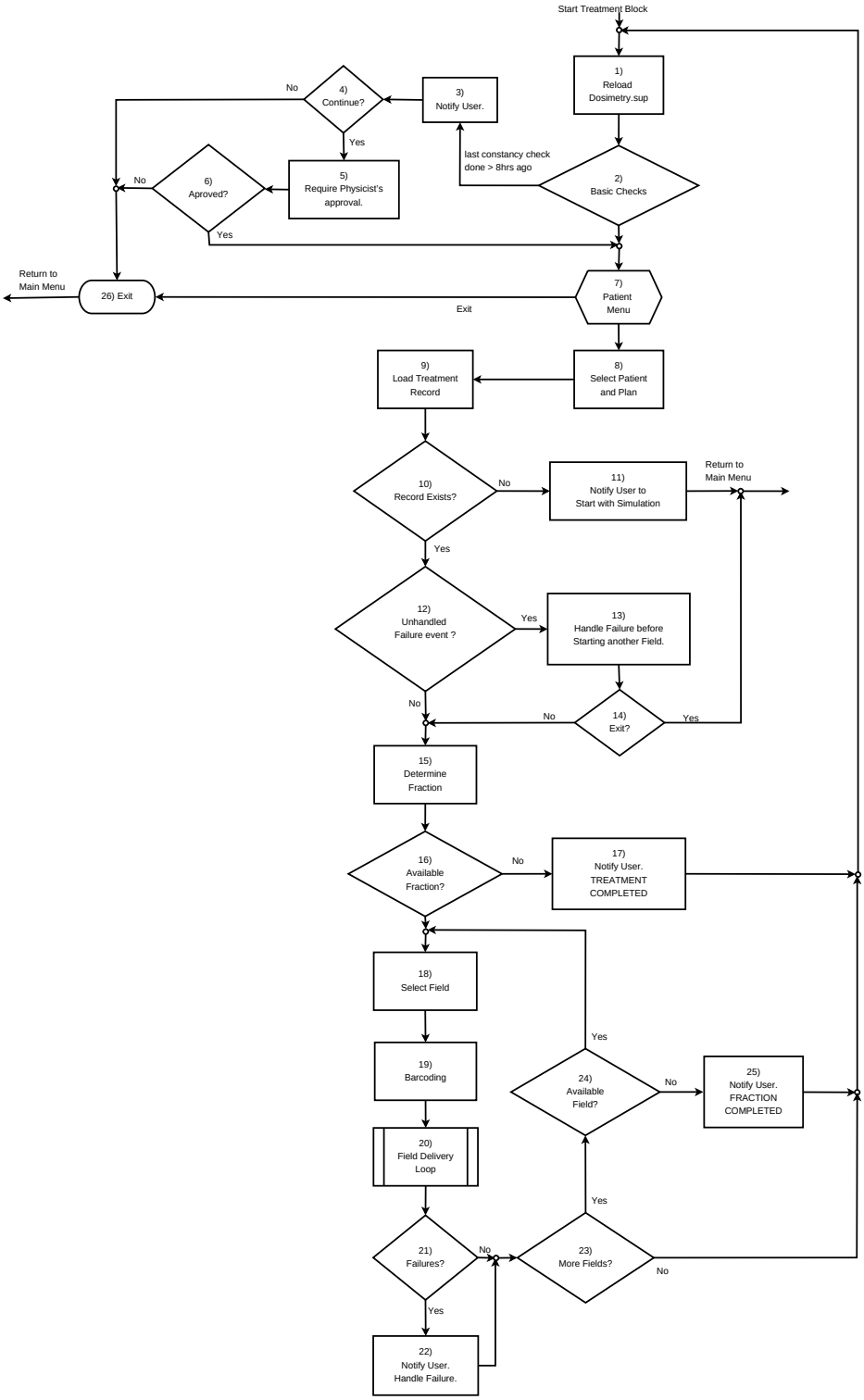

Figure 5.6: Supervisory System Treatment Block

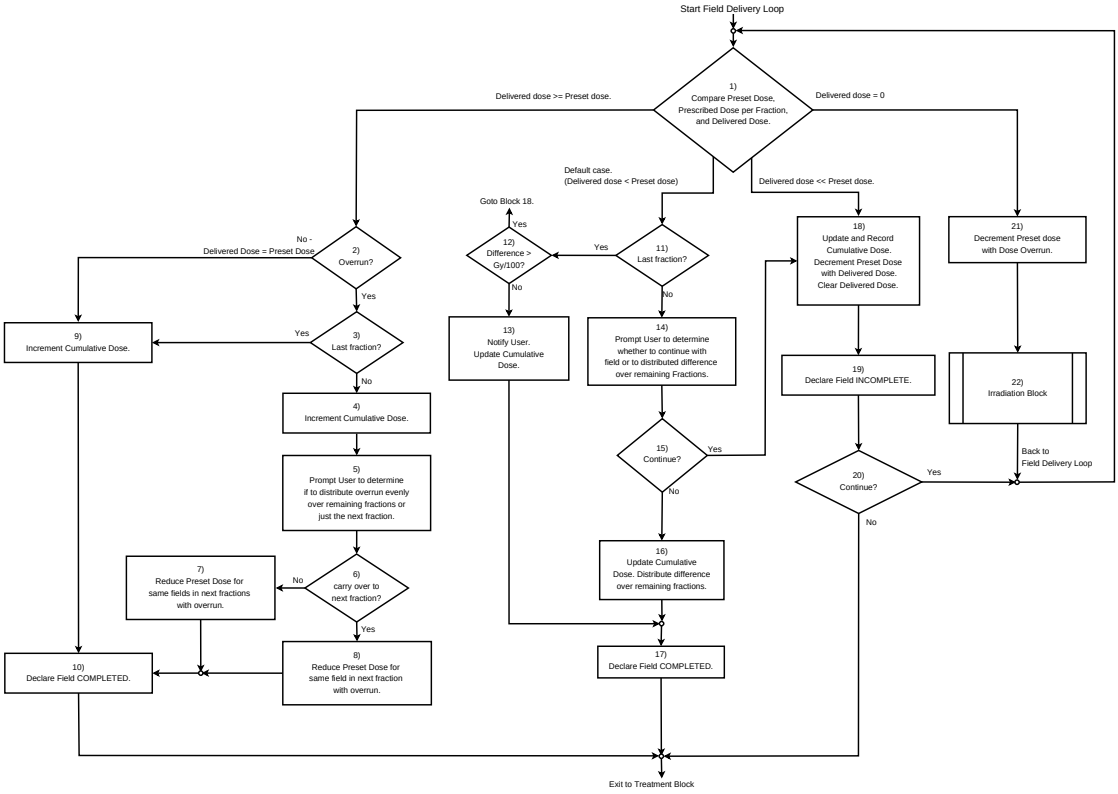

Figure 5.7: Supervisory System Field Delivery Loop

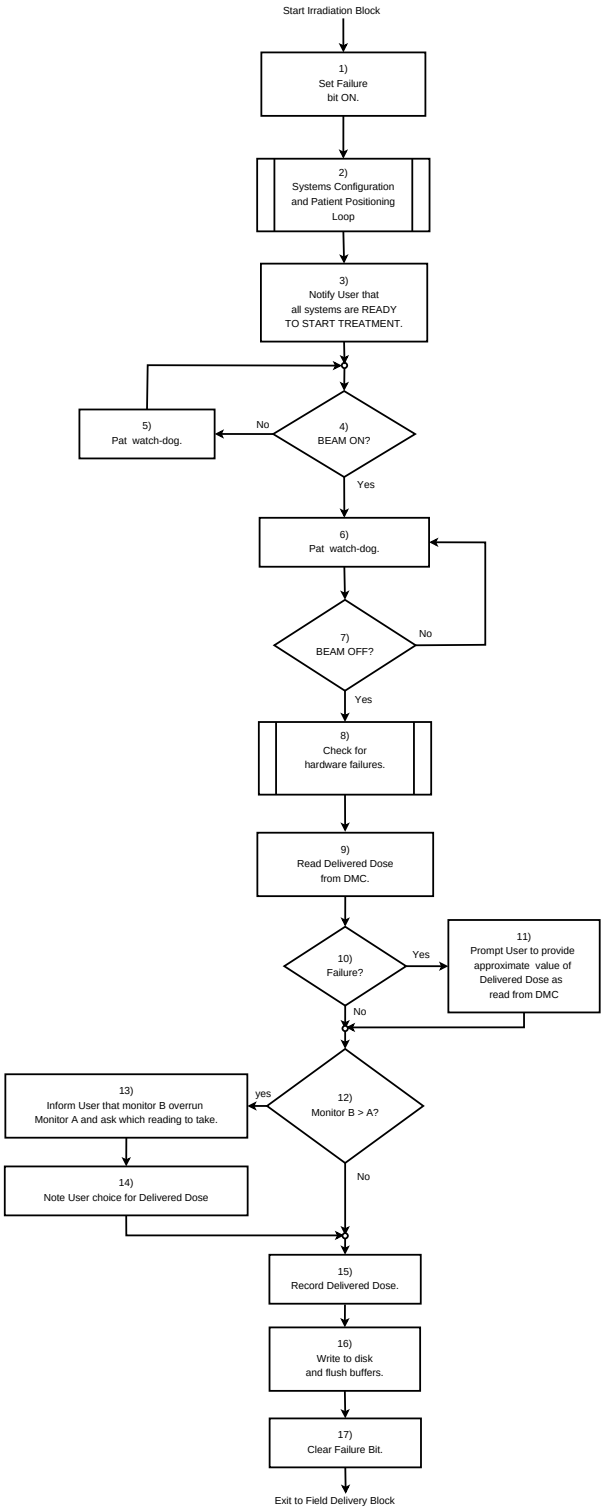

Figure 5.8: Supervisory System Irradiation Block

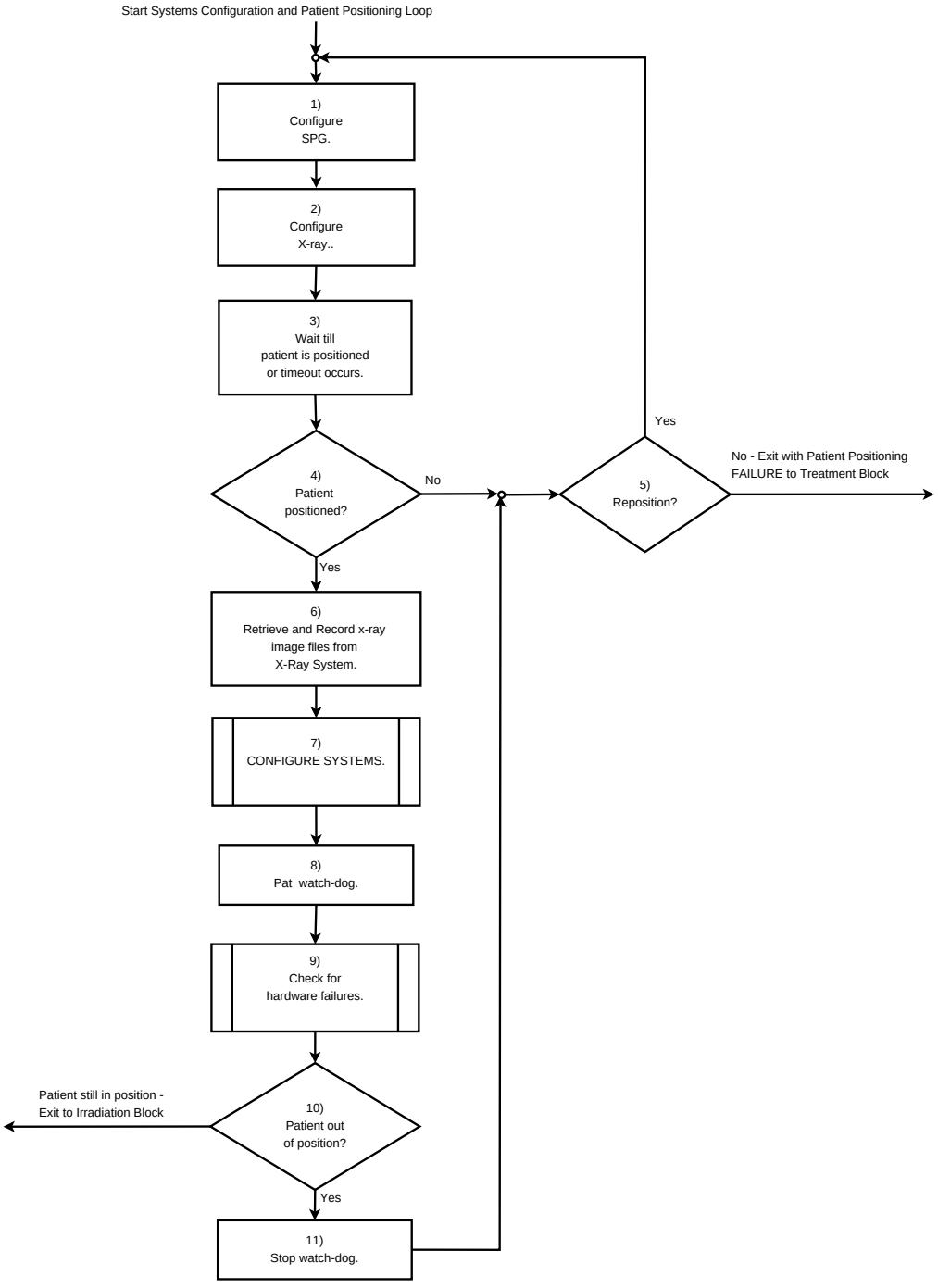

Figure 5.9: Supervisory System - Systems Configuration and Patient Positioning Loop

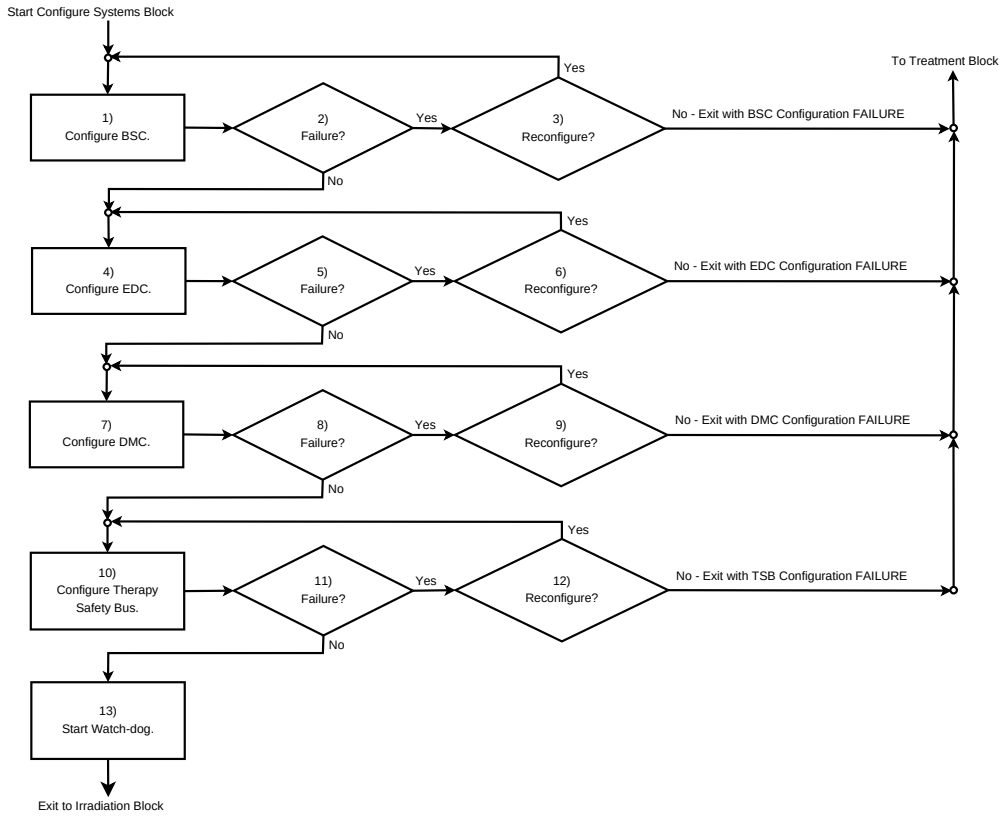

Figure 5.10: Supervisory System - Systems Configuration Block

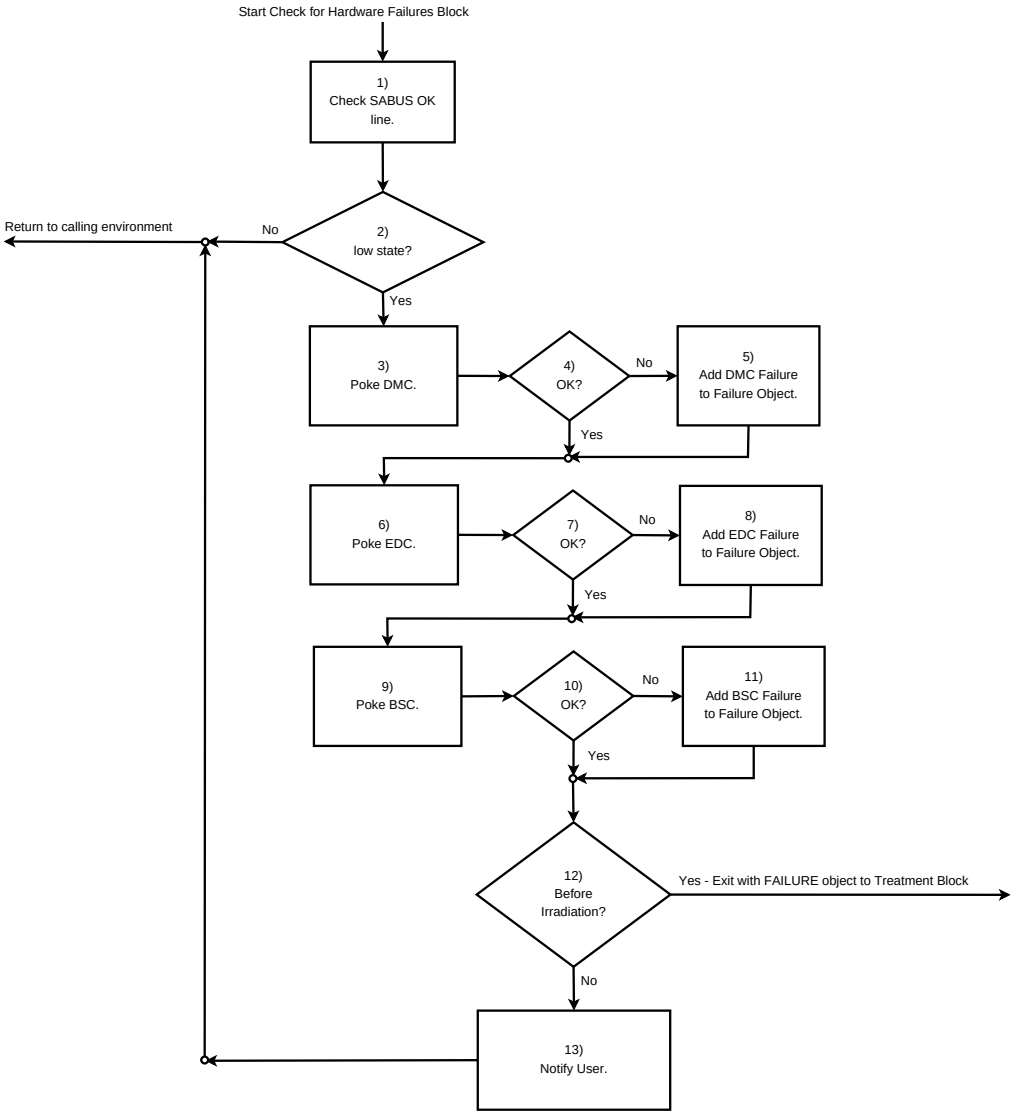

Figure 5.11: Supervisory System - Check for Hardware Failures Block

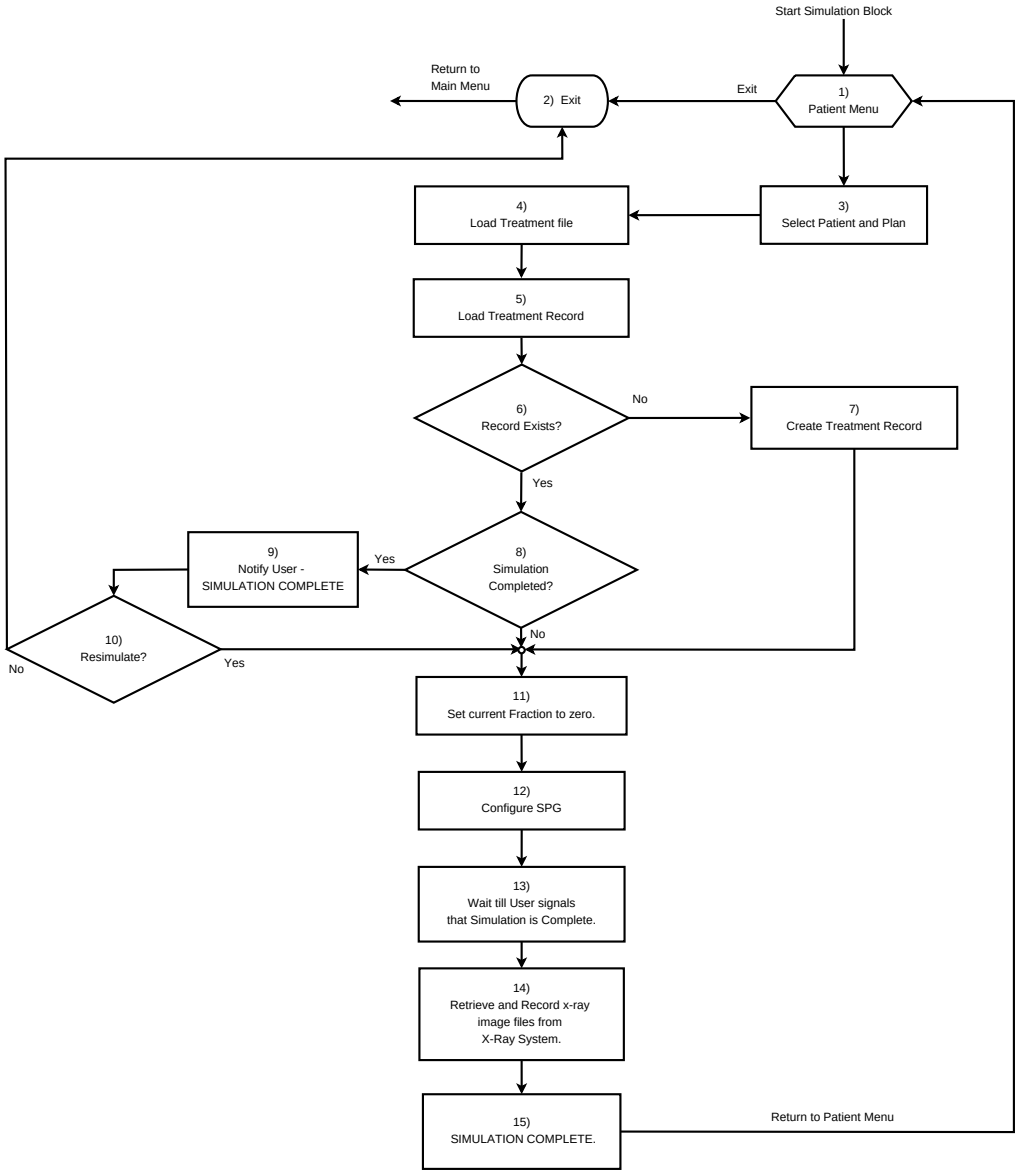

Figure 5.12: Supervisory System - Treatment Simulation Block

5.2.1 Architectural Design

Figure 5.3 demonstrates the proposed architectural design of the system. The supervisory system application process consists of two loosely coupled main threads; the main program thread and the request handler thread. The main program thread allows the intended users to interact with the system through the GUI, thus it provides most of the functionality of the supervisory system. Configuration of the Therapy Safety Bus is accomplished through the PCI interface and is handled by the main program thread. The system will also consist of a MySQL database into which the system users and their associated groups will be stored in order to provide user-based access control of the system. The proxy functionality of the system will be handled by the request handler thread which will publish RPC services/methods that other systems can access remotely (e.g. file transfer requests by SPG or PR systems). Configuration of other systems will be handled by the main program thread using RPC through the network and transport layers. Lastly, to achieve inter-thread communication, the system will employ global variables which could contain shared data between the two main threads.

5.2.2 Top Level Program Flow

As previously discussed in Chapter 2, proton therapy at iThemba LABS consists of five distinct steps; calibration of the patient positioning system, dosimetry, treatment planning and preparation of the patient, treatment simulation, and treatment delivery to the patient. The following subsections describe the proton therapy treatment process highlighting the proposed solution for the new proton supervisory system in each step.

5.2.2.1 System Calibration

The 200MeV beam is only available for proton therapy on Mondays and Fridays from 08h00 to 17h00 and from 08h00 to 15h00 respectively. Thus Mondays and Fridays constitute treatment days for proton therapy. Every treatment day, the CCD cameras of the SPG system are calibrated using a small calibration cube on which sides radiopaque and retroreflective markers are attached. Calibrating the SPG entails using this cube to determine the transformation matrix between the coordinate system of the markers and that of the beam line. It is this transformation matrix that the X-Ray Imaging system (Portal Radiographic system) uses to check whether the patient has moved out of position so as to signal the SPG to stop the beam and/or position the patient correctly.

This step of the proton therapy process does not require any input from the supervisory system.

5.2.2.2 Dosimetry

This step involves daily constancy checks on the absolute calibration of the dose monitors. The dose constancy should be ± 2%, and the calibration equipment should be positioned at the isocenter when the check is performed. The data from the constancy check includes the temperature and pressure conditions during the check, as well as the CVolt factors CV1 and CV2 , together with the date and time when the constancy check was performed, and the name and HPCSA registration number of the medical physicist who performed the check.

All these data is entered using either the Physics form/screen (in Figure 5.4 block 10) or the Admin form (in Figure 5.5 blocks 12 to 15). The output of the dosimetry step/process is the dosimetry.sup file which is loaded by the supervisory program after successful login by a user (refer to Figure 5.4 block 3). No patient treatments are allowed if the dosimetry file does not exist or fails to load or if the time elapsed since the last constancy check, as specified by the date and time in the dosimetry file, is more than eight hours. This restriction does not apply to treatment simulations.

5.2.2.3 Treatment planning and preparation of patient

Treatment planning and preparation starts when a patient has been accepted for proton therapy at iThemba LABS. A patient-specific marker carrier is manufactured and fitted with radiopaque and retroreflective markers. A scanning and planning schedule is agreed on and the patient undergoes CT scans. The SPG-CT Scanner system identifies the location of the tumour with respect to the reflective markers and ensures that the patient does not move out of position when the scans are being taken. The scans are then used by the treatment planning software to construct cubes (Hounsfield units) which aid the oncologist in delineating sensitive structures as well as the target volume on the images. After the radiation oncologist has prescribed the dose and fractionation, the planning radiographer plans treatment conforming the dose to the target volume while limiting it to critical structures [10; 25].

Once the plan has been approved by the radiographer, medical physicist and oncologist, collimators are manufactured by in-house mechanical engineers. These collimators are cylindrical blocks with the required aperture needed to shape the beam for each field in the treatment plan. Planning radiographer then goes on to create light-slabs from the CT data using treatment planning software, which will then be used to generate digitally reconstructed radiographs (DRRs) by the X-Ray Imaging system during patient positioning.

Also, administrative functions and some of physics-related functions of the supervisory system could be included in this step since they are part

of treatment preparation. As seen from Figure 5.4, after a successful login by the user the program loads all initialization information from the files `config.sup` and `dosimetry.sup` and then exposes the user with the Main Menu where the user can choose to `view slave screen for the Safety Interlock System (SIS)` (block 12), `perform administrative functions` (block 13), `perform physics functions` (blocks 8 to 11), `simulate patient treatment` (block 14), or to `deliver actual treatment to the patient` (block 15). `Config.sup` file contains all the data needed for the initialization of the supervisory system, namely the IP addresses of the SPG system and the x-ray imaging system, the treatment unit controlled by the supervisory system, and a list of strings that specify the radiation types and treatment techniques that are supported by the treatment unit. It also includes the dosimetric constants: Γ, Δt_{min}, Δt_{sys}, $\%\Delta_{AB}$, Δ_{AB}^{min}, $\dot{D}_{ref}^{nom}$ and $\dot{D}_{ref}^{max}$ for every radiation type [5]. To perform administrative functions the user will select block 13. Access to this option is only granted to users who belong to the admin group. These admin functions include user administration tasks such as add, delete or edit users, updating or creating configuration files `Config.sup` and `Dosimetry.sup`, as well as viewing treatment records of existing patients (refer to Figure 5.5).

Physics related functions are only available to users in the physics or administrator groups. The following are available options;

- Park Supervisory;
- Enter or Update
- Manual Configuration.

`Under Park Supervisory`, the user is allowed to configure TSB lines without going though the 'normal' beam-on initialization steps followed during patient treatment. Thus enabling this functionality will cause the supervisory system to drop FC19 and Beam-On HOLD lines on the TSB as well as 'pause' its monitoring function.
`Enter or Update Dosimetry.sup` allows the physics user to input calibration data for the DMC during the daily constancy check. The name of the medical physicist inputting the data, as well as the date and time when the constancy check is being made, are recorded in the dosimetry.sup file together with the calibration data. This function is usually performed during the dosimetry step of the proton therapy process (refer to section 5.2.2.2). In `Manual Configuration`, the medical physicist is given control to set beam parameters so as to carry out physics experiments.

It is in this step that medical physicists conduct scheduled checks of the proton therapy equipment to ensure that everything still functions correctly.

5.2.2.4 Treatment Simulation

During treatment simulation radiation therapists simulate treatment to decide upon the optimal order of the treatment fields, optimal choice of camera combinations for the SPG system, as well as optimal positions for the patient and treatment chair associated with each field. Treatment simulation also helps uncover any potential problems with the treatment plan which will result in the radiographer drawing up a new plan. This step also familiarizes the patient with the treatment procedure and prepares them for actual treatment.

When simulating treatment, the radiation therapist selects block 14 from the Main Menu (Figure 5.4) of the supervisory system and is presented with the simulation form (Figure 5.12). This screen allows the user to enter a patient ID and plan number search string which prompts the program to load relevant patient plan and treatment record files. If the current simulation is the first for that patient the program creates the record file and initializes it with data from the plan file (Figure 5.12 blocks 3 to 7). However, if the record file already exists, the supervisory program checks whether simulation has been completed for that patient. Simulation will have been completed if all fields under fraction 0 (the simulation fraction) have been marked off as complete. The supervisory system then configures the SPG and X-Ray imaging systems and then waits until the user indicates that simulation has been completed. The signal from the user causes the program to query the X-Ray imaging system for x-ray image files that were used in positioning the patient. The names of the files are recorded in the treatment record file, and the files are send to the Radiotherapy Database Server. The program returns to the start of the simulation block to allow the user to simulate another patient's treatment or to return to the Main Menu.

5.2.2.5 Treatment Delivery

Before actual treatment commenses, medical physicists perform scheduled checks of proton therapy equipment (bite block, collimators, double-wedges, treatment chair, dose monitors, entire beam-line, and SPG's CCD cameras) together with their control software. They also monitor the beam profile (range, flatness and symmetry) and liase with accelerator engineers to adjust the beam until it is acceptable for treatment. The patient is then placed onto the treatment chair and correctly positioned with respect to the isocenter with the aid of the SPG system. The therapist then selects the treatment block in the Main Menu of the supervisory program (Figure 5.4 block 15) to be presented with the treatment screen (Figure 5.6). The system reloads the `Dosimetry.sup` file parsing it to check when the last constancy check was done on the file. If more than 8 hours have elapsed since the last check, the user is notified and asked whether they wish to continue with treatment regardless of the failed check

on the dosimetry file. If the user wishes to continue, the system prompts for authorization from a physicist by requiring a physicist's username and password. If the credentials are approved, the incident is logged to the patient record file detailing the failed constancy check and the physicist's approval to still continue with treatment (name and staff ID will be logged next to the incident). Program execusion then proceeds to Figure 5.6 block 7, same as if the last constancy check was performed not more than 8 hours ago (i.e. the check succeeded). The user is then allowed to select and administer a specific treatment field from the patient treatment plan. This step is reached if there is an incomplete fraction available, and the user is only allowed to select a field α for which $F_{\alpha,i} = 0$ (refer to Addendum A). When a field has been selected the program displays all its parameters to the user and then proceeds to the barcoding section where it allows the user to verify that all equipment needed for that particular field are available and are the correct ones being used for treatment. This is done by displaying names of all required equipment with unmarked check-boxes next to them. The user is then prompted to scan-in barcodes of each equipment. A check mark is made on a check-box next to every successfully scanned equipment. The program waits until all required equipment have been scanned before proceeding to the field delivery loop. The field delivery loop implements the core functinality of the proton supervisory system. It implements the dose algorithm which guarantees that only the planned dose as indicated in the treatment plan file is administered to the patient by properly handling dose overruns that may occur during sessions, as well as recording each and every dose that is delivered to the patient, even during treatment failures.

After the barcoding block (Figure 5.6 block 19), the program loads the following dosimetric quantities from the treatment record and/or treatment plan files: `preset dose`, `prescribed dose per fraction`, `delivered dose` and `cumulative dose` (if they are not already loaded). On entering the field delivery loop, these quantities are compared against each other to determine the appropriate execusion flow that the program needs to take. The following options are available [20]: If,

- $(DeliveredDose > PresetDose$ and $currentfraction = lastfraction)$ OR $DeliveredDose = PresetDose$: The program increments the cumulative dose with delivered dose, declares the field complete and exits to treatment block so that user can select another field or can return to the main menu.
- $(DeliveredDose > PresetDose$ and $currentfraction \neq lastfraction)$: The program increments the cumulative dose with delivered dose. It then prompts the user to determine if to distribute overrun evenly over remaining fractions or just the next fraction. If user wishes to 'carry over' the difference to the next fraction, then the program reduces the

preset dose for same field in next fraction with overrun, and declares field completed. Otherwise, it divides the overrun by the number of remaining fractions and reduces that number from the preset doses of each of the remaining fractions.

- *(DeliveredDose* $\ll$ *PresetDose)*: The program increments the cumulative dose with delivered dose, decrements preset dose with delivered dose, sets delivered dose to zero, and declares the field incomplete. It then prompts the user whether to readminister the field.

- *(DeliveredDose* $<$ *PresetDose* and $currentfraction = lastfraction$*)*: If difference between delivered dose and preset dose is greater than $Gy/100$ (i.e. 100th of a Gray), then it is handled similar to the *deliveredDose* $\ll$ *presetDose* condition, otherwise if the difference is less than 100th of a Gray then the program increments cumulative dose with delivered dose, notifies the user about the negligible difference, declares the field complete, and exits to treatment block.

- *(DeliveredDose* $<$ *PresetDose)* and $(currentfraction \neq lastfraction)$: The program asks the user whether to continue with the field or to distribute the difference over ramaining fractions. If the user wishes to continue with field then it is handled same as the *delivereddose* $\ll$ *presetdose* condition above, otherwise if user wishes to distribute the difference, then program increments cumulative dose and increases preset doses of the same field in remaining fractions with the difference (underrun) divided by number of remaining fractions. It then declares the field complete and exits to the treatment block.

- *DeliveredDose* $= 0$: If expected dose overrun (due to delay taken by the beam-stopping devices to stop the beam after beam stop signal has been issued) was not taken into account in the treatment plan, then program first reduces the preset dose with the expected overrun before entering the Irradiation Block (Figure 5.7 block 22 and Figure 5.8), otherwise it simply enters the irradiation block and administers treatment.

The irradiation block (Figure 5.8) is where the actual irradiation of the patient occurs. The program only gets to execute this block if and only if the delivered dose parameter in the program memory is set to zero (for that particular treatment session). Being a critical section of the program, the first thing that the program does when entering this block is initialize the failure BIT/OBJECT and keep updating it whenever the program performs a distinctive task so that when a failure occurs it is easy to recover from, as it will be known exactly what the program was doing before it failed. After initializing the failure object, the program then configures other subsystems of the Proton Therapy Control System and confirms that the patient has been correctly

positioned by the Patient Positioning System (refer to System Configuration and Patient Positioning Loop Figure 5.8 block 2 and Figure 5.9). If the configuration was successful, the user is notified that all systems are READY to START TREATMENT. The program then enters into an active BUSY WAITING stage of continuously checking if the Beam ON condition has occurred or not, at the same time patting the watchdog (which is initialized in the Systems Configuration and Patient Positioning Loop) so that it doesn't time out. When the Beam ON condition has been met (i.e. Beam OFF signal low) then the program keeps on patting the watchdog while busy polling the Beam OFF signal to see if it has been raised or not.
When the Beam OFF condition is met (i.e. Beam OFF signal high), the supervisory program first checks for hardware failures to determine what stopped the beam (see Figure 5.8 block 8 and Figure 5.11) and then attempts to read back the delivered dose from the DMC. If the program fails to read back the dose (e.g. if the DMC has failed or there is a communication failure), the user is prompted to provide the delivered dose as read from the mechanical counters of the DMC (both monitor A and B). The program then compares the values from the two dose monitors (see Figure 5.8 block 12) and if monitor A is slightly greater than or equal to monitor B then the value for the delivered dose is taken as monitor A reading, otherwise (i.e. monitor B is greater than A) the user is informed that monitor B has overrun monitor A and is asked which reading should be taken as the approximate value for the delivered dose. The program then updates the *delivered dose* parameter in the program memory and commits the changes to disk (i.e. writes to the treatment record file, flushes buffers and closes the file handle). The program then clears the failure object before exiting the irradiation block back to the start of the field delivery loop to check how the delivered dose compares to the preset dose in order to determine if the field has been completed or not.

The systems configuration and patient positioning is where the patient positioning system (SPG and X-Ray imaging systems) is configured (sent configuration parameters). The program waits until it receives a signal from the X-Ray imaging system indicating that the patient has been placed in position, or until a timeout occurs (if there is a communication failure between the Supervisory system and the X-Ray imaging System). If a timeout occurs, or the patient is not properly positioned, the supervisory system notifies the user and prompts if they wish to reposition (see Figure 5.9 blocks 1 to 5). The program exits to the treatment block with a patient positioning failure if the patient is not in position and the user does not wish to reposition the patient. If however, the patient has been properly positioned, as indicated by the signal from the X-Ray imaging system, the supervisory system program then queries the X-Ray imaging system for the final x-ray image files used when positioning the patient. After successfully retrieving and recording the x-ray image files, the program configures all other remaining subsystems of the

Proton Therapy Control System, namely the DMC, BSC, EDC and the TSB, as well as starting the watchdog timer (see Figure 5.9 block 7 and Figure 5.10).

After all systems have been successfully configured, the program then enters the watchdog section. This is the section entered into by the program after successfully starting the watchdog timer to when the timer is intentionally allowed to timeout and the watchdog object cleared from the program memory. It is the responsibility of the supervisory system to satisfy or pat the watchdog during this section so that a watchdog timeout can quickly alert the user that the supervisory system has failed thus rendering it incapable of performing its task, before or during irradiation. After configuring systems and patting the watchdog, the program double-checks that all systems are still functional before irradiation can start. This is done by checking if any of the configured systems experienced a hardware failure, as indicated by the `SABUS OK` line (see Figure 5.11). Similarly, after irradiation, when the Beam OFF signal goes high, the program checks if it was a hardware failure of another system that stopped the beam, and records that information in the treatment record file. Under normal operational modes, any subsystem of the proton therapy control system should drop the SABUS OK TSB line when it incurs a hardware failure. Thus a dropped SABUS OK line is an indication that one or more of the control subsystems experienced a hardware failure. To determine which subsystem(s) failed, the supervisory system individually pokes every system (i.e. DMC, EDC and BSC) and updates its failure object accordingly. The poke command is a simple RPC call which works more like the traditional PING command. After poking all relevent systems and updating the failure object, the program exits to the treatment block if the hardware failure(s) occurred before irradiation, otherwise it notifies the user and then returns to the calling environment. This is because if a hardware failure occurs before irradiation, treatment is prohibited, hence why the program exits with a hardware failure result to the treatment block. On the other hand, if the failure occurred during treatment and thus caused the beam to be switched off, the program determines what caused the failure, notifies the user, records the incidence, and then attempts to read the delivered dose from the DMC which is why it returns to its calling environment. If no hardware failures occurred, the program confirms that the patient is still in position before returning to the irradiation block and proceeding with treatment. However, if the patient is no longer in position, it stops the watchdog and prompts the user whether to reposition. The program then loops to the start of the systems configuration and patient positioning loop if a positive response is issued by the user, otherwise it exits to the treatment block with a patient positioning failure. The `systems configuration` block deals with the configuration that the supervisory system performs to each and every other subsystem of the Proton Therapy Control System that should be configured before treatment. The order followed when configuring systems is to first configure those systems

involved in beam alignment and steering (BSC, and EDC) then move on to the DMC and end with the TSB before starting up the watchdog timer. All configuration is done through RPC calls to the specific system with configuration data sent as structures with appropriate attributes. When in this block, the program first sends the required configuration data to the BSC and waits for an acknowlegdement of receipt. If a timeout occurs or the BSC sends a negative response, the program attempts reconfiguration for atleast two times before notifying the user. The user is then prompted whether to reconfigure the system or to exit to the treatment block with a BSC configuration failure. Only after the BSC has been successfully configured will the program proceed to configure the EDC. The same configuration process follows and the program exits to the treatment block with an EDC configuration failure if it is unable to successfully configure the EDC. Successful configuration of the EDC allows the program to then proceed to DMC configuration process. Similary, the same configuration process applies (just like for BSC or EDC) and the program exits to the treatment block with a DMC configuration failure if unable to configure the DMC. It then raises the SS OK line on the TSB signalling that it has successfully configured other systems and is ready to supervise patient treatment. Repeated failures to raise the SS OK line cause the program to exit to the treatment block with a TSB Configuration failure. As the last action before exiting the systems configuration block and returning to the systems configuration and patient positioning loop the program starts up the watchdog timer and thus enters the watchdog section.

Upon exiting the field delivery loop, the program checks to see if any failures occurred while in the delivery loop and if so, it summons the failure recovery block (section 5.2.2.6). If there are incomplete fields available the program informs the user and waits for the user to select another field to administer, or if the user wishes so, to return to the start of the treatment block. If all fields have been completed, the program declares that fraction complete, notifies the user, and returns to the start of the treatment block.

5.2.2.6 Error Handling

Numerous errors and failures can occur during treatment delivery to the patient and it is the responsibility of the supervisory system to accurately record such failures and gracefully recover from them. Failures of interest include [20]:

- Configuration failures that prevent irradiation from commencing since one or more systems cannot be correctly set up;
- Hardware failures (power failure or hardware malfunction e.t.c) that can occur after systems have been configured, but before irradiation starts;

- Hardware failures that can occur to other systems during irradiation (treatment delivery);
- Power failure of the supervisory system during `Beam ON` condition i.e. after all systems have been properly configured and irradiation has started (`BEAM OFF` signal went low);

If a power failure (of the supervisory system), or any unhandled failure, occurred in the previous execution of the program when treating a certain patient, it will be picked up in the treatment record of that particular patient through the unhandled failure flag (see Figure 5.6 block 13). The flag will also highlight the cause of the failure by stating what the program was doing when the failure occurred. Before attempting another session, the program notifies the clinician about the unhandled failure, and if irradiation had started when the failure occurred, it prompts the user to enter the delivered dose as read from the mechanical counters of the DMC. However, if a failure occurs under normal execution of the program, such as when configuring systems, or one of the systems experiences a hardware failure, the program quits the field delivery loop and jumps to blocks 21 and 22 of Figure 5.6. Similarly, if irradiation had started when the failure occurred, it prompts the user to enter the delivered dose as read from the mechanical counters of the DMC, otherwise it simply notifies the user about the failure, reads the delivered dose from the DMC and records the incident in the patient treatment record file. The ability to recover from failures relies on the creation and proper updating of the failure object parameter as well as timely updating of the patient record file. Thus it is of paramount importance that all file-write buffers are timeously flushed to commit every change to disk.

5.3 Supervisory System Implementation

This section describes how the software system was developed starting with the development methodology followed in section 5.3.1. Section 5.3.2 discusses the choice of programming language used as well as the development hardware for the system. Being a file-intensive system, section 5.3.3 highlights the Boost.Spirit template metaprogramming techniques adopted for parsing the numerous files used by the supervisory system. The section ends with a discussion of the low-level module of the system which interfaces to the Therapy Safety Bus through an 848C Eagle Technology PCI card.

5.3.1 Methodology

E**X**treme **P**rogramming (XP) agile software development methodology was followed when implementing the supervisory system. XP was chosen because of its rich practices and approaches to reliable and robust software development,

and because it is the most heavily documented, tried and tested agile methodology in software development. In [26] Abrahamsson et. al describe agile methodologies as software development approaches that are flexible, adaptable and allow developers to make late changes in the specifications, hence making the requirements of the software project not to be locked-in and frozen before the design and software development commences. They (Abrahamsson et al) also identify focal values honored by all agile methodologies as;

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

It is these central values, which accommodate the volatility and competitiveness of today's business community, that no doubt have put agile methods in the forefront of software development approaches, and that have invariably led us to the choice of XP as our preferred software development approach. Figure 5.13 depicts the life-cycle of the XP process [26] and how it was applied in implementing the supervisory system.

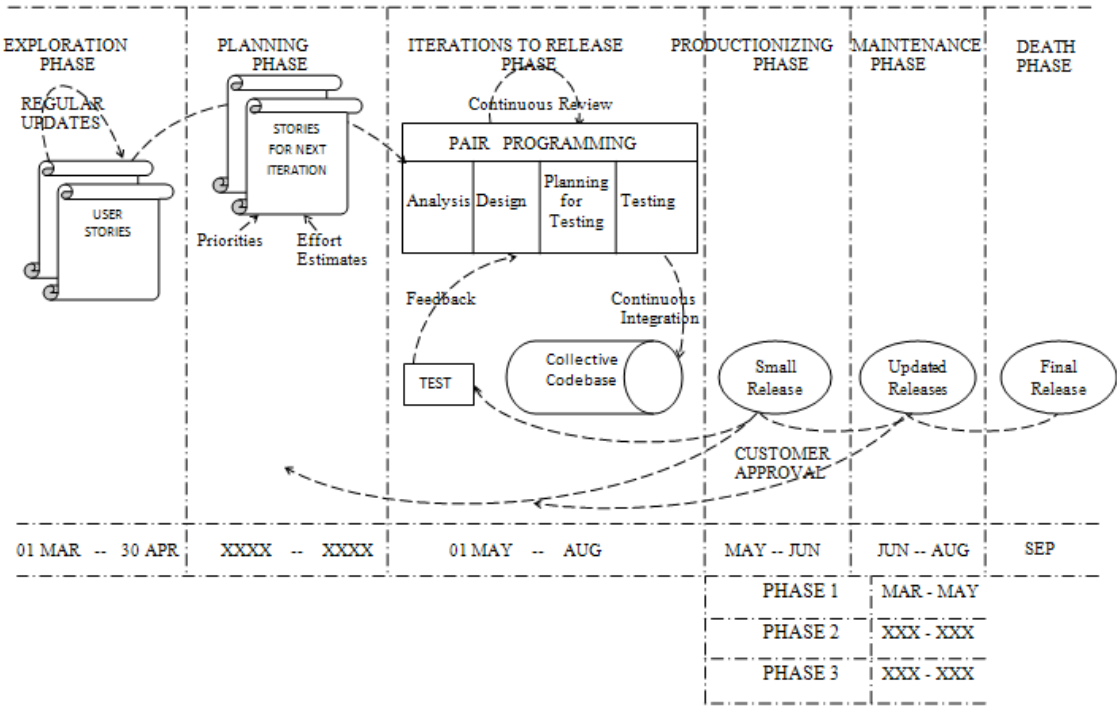

Figure 5.13: Software Development Methodology

In the **Exploration Phase**, the tools, technologies and practices to be employed in developing the project, were studied while at the same time users wrote out 'story cards' containing the features they wished to be included in the first release. The tools and technologies that were proposed to be used included;

- Qt4 C++ for GUI programming.
- Qt Creator IDE for development and Unit tests.
- Barcode Scanner.
- RPC method invocation. (with possible upgrade to other technologies such as RMI, SOAP, CORBA, e.t.c)
- Eagle Technologies 848C PCI DAQ card.
- SABUS

In the **Planning Phase**, an agreement of the contents of the first release was made, and the priority order for the user stories were set. The stories which enforced building the overall structure for the whole system were selected to form the contents of the first release.

It is in the **Iterations to Release Phase**, where the system underwent several iterations before the first release. These iterations were determined from the stories' priority order set in the Planning Phase. The functional tests created by the customer (or users) were then run at the end of every iteration. Eventhough the Supervisory System was to be implemented in three phases, each phase was implemented in increments which allowed early testing and debugging before the system became too complex.

At the **Productionizing Phase**, extra testing and performance analysis was carried out before the system could be delivered to the users.

In the **Maintenance Phase**, more effort was on customer/user support tasks as well as some corrective and preventive maintenance on the system.

The **Death Phase** will be archived when the system meets all the functional requirements set in the Planning phase and it's operation is satisfactory to the users in all respects (such as performance and reliability). The XP practices that were employed in this project include [21; 26];

- Tactical Planning Game (between developer and users)
- Small/Short Releases

- Simple Design
- Exhaustive Testing (Test-first practice)
- Refactoring
- Relentless Integration
- On-site Customer/User
- Coding Standards
- And, a 40-hour Working Week

5.3.2 Development hardware and software

The system was developed on an Intel Core 2 Quad Q9000 series desktop PC with four processing cores that can achieve up to 12MB of L2 cache and 1333MHz Front Side Bus [27]. The desktop PC also had 2x 2GB of RAM and an Intel PRO/1000 GT Desktop Adaptor NIC for fast and reliable network connectivity. Due to the non-functional requirement of using a Linux operating system for development, the supervisory system was developed on Scientific Linux 5.5 running kernel 2.6.18. Qt-Creator 1.3.1 based on Qt 4.6.2 was the IDE of choice for all the programming work, with C++ and C as programming languages. A Zebex Z-3010/USB barcode scanner was used in identifying treatment equipment so as to minimize the possibility of human error during the treatment process, while attribute-based access control for users was afforded through the use of an internal MySQL database, version 14.22 distribution 5.0.77. Lastly, the system used an Eagle Technology 848C PCI card in order to interface to the custom-made TSB-Simulation rig discussed in chapter 4.

5.3.3 Template Metaprogramming techniques for file parsing

The supervisory system employed Boost.Spirit libraries to develop generic parsers for reading in all the different files used by the system.

> *"Boost Spirit is an object-oriented, recursive-descent parser and output generation library for C++. It allows you to write grammars and format descriptions using a format similar to Extended Backus Naur Form (EBNF) directly in C++. These inline grammar specifications can mix freely with other C++ code and, thanks to the generative power of C++ templates, are immediately executable. In retrospect, conventional compiler-compilers or parser-generators have to perform an additional translation step from the source EBNF code to C or C++ code."* [28]

This means that using Spirit, one is able to create parser grammars by specifying required tokens used for parsing using C++ templates, thus generating generic parsers that can handle different data types without recompilation of the source code.

Below is a snippet of the `config.sup` file parser which reads in the file from disk and stores it as attribute-value pairs in a map data strucure in program memory.

```
-----------------------------
...

namespace parsers
{
    namespace qi = boost::spirit::qi;
    namespace ascii = boost::spirit::ascii;

    typedef std::multimap<std::string,std::string> pairs_type;

    using qi::lit;
    using ascii::char_;
    using qi::eol;

    ////////////////////////////////////////////////////////////////
    // Our Supervisory Configuration file (Config.sup) parser
    ////////////////////////////////////////////////////////////////

   template <typename Iterator>
      struct config : qi::grammar<Iterator, pairs_type()>
    {
      config() : config::base_type(configStart)
      {
       configStart %= configPair >> *(+eol >> configPair);
       configPair %= configKey >> -(-lit('=') >> configValue);
       configKey %= *(char_ - (+lit('=') | +eol));
       configValue %= +(char_ - eol);
      }

  qi::rule<Iterator, pairs_type()> configStart;
 qi::rule<Iterator, std::pair<std::string, std::string>()> configPair;
  qi::rule<Iterator, std::string()> configKey, configValue;

 };

}
```

The program defines a namespace called *parsers* which contains the different parsers for each specific file used by the supervisory system. The parser illustrated is for the configuration file `config.sup` which stores all the initialization information needed by the supervisory program. Using EBNF, the above grammer can simply be translated into;

configStart ::= configPair (endOflineCharacter+ configPair)∗;
configPair ::= configKey − (−(‘=’) configValue);
configKey ::= (char_ − ((‘=’)+ / endOflineCharacter+))∗;
configValue ::= (char_ − endOflineCharacter)+;

Where ∗, +, − and / are the regular-expression Kleene star, plus, difference and alternatives operators respectively. Note that Spirit.Qi uses prefix star and plus operators since there is no postfix star or plus in $C++$. The Kleene star, plus, difference and alternatives operators are defined as;

a∗ = Match zero or more of *a*.

a+ = Match one or more of *a*.

a − **b** = Match *a* but not *b*.

a / **b** = Try to match *a*. If *a* succeeds, success, otherwise try to match *b*.

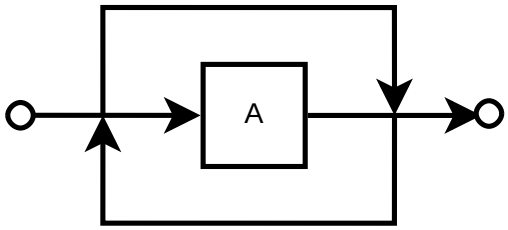

Figure 5.14: Kleene Star

It follows therefore that the parser simply parses the `config.sup` file one line at a time forming attribute-value pairs from strings before and after the ‘=’ regular expression, and storing the result into a multimap data structure of strings.

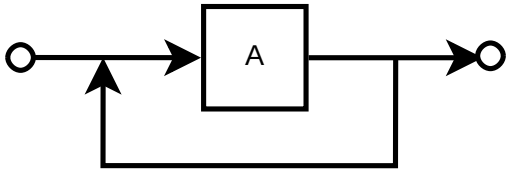

Figure 5.15: Plus operator

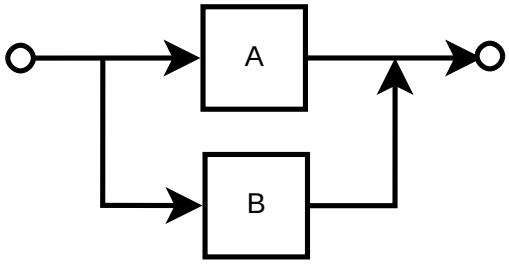

Figure 5.16: Alternatives - Ordered Choice

Chapter 6

Experimental Results

This chapter is about the tests conducted and results obtained in determining whether the developed supervisory system meets its intended objectives. It begins with section 6 which describes the functional tests conducted and the results obtained. Section 6 ends the chapter with a discussion of some of the non-functional tests carried out on the system.

Functional Tests

Communication with other subsystems

These tests represent both unit and beta (integration) testing of the supervisory system against its functional requirements. The tests were conducted at the component level using the “happy path" (inputs within expected range) and “unhappy path" style of testing. Since the development of most of the subsystems forming the new Proton Therapy Control System is still underway, the bench testing methodology was followed. That is to say, all tests described in this chapter were carried out with emulator systems whose interfaces mimicked those of actual systems currently under development. This means that RPC server programs with well-defined methods/functions that could be remotely invoked were developed to simulate each of all systems to be configured by the supervisory system through the PT1 subnet. These programs were launched on a separate PC from the one hosting the supervisory system, but in the same network so as to provide identical network communications with those expected for the actual systems. In the case of the SPG and PR (x-ray) emulators, file request modules were implemented to simulate actual file request commands expected from such systems, and to test the proxy functionality of the supervisory system.

Figure 6.1 illustrates the test setup consisting of a Surecom EP-808X 8 port 10/100M Ethernet Mini Switch, supervisory system computer, and the test computer hosting the simulation programs. Each system is represented by

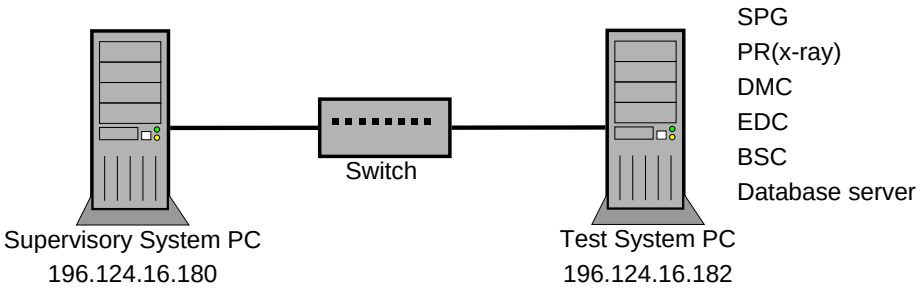

Figure 6.1: Simple Test setup

a unique process running on the test PC with a unique RPC program number. Figure 6.2 shows some of the configuration information send by the supervisory system to respective systems to be configured. A custom struct datastructure is used to hold the configuration information send by the supervisory system to each and every subsystem while a simple return string is expected from all subsystems to indicate whether the configuration was successful or not.
To test how the supervisory system reacted to network failures amid configuration or file transfer requests, the network cable was intentionally removed for 15 seconds during a file transfer between the supervisory system and the test SPG system, and then inserted back into the NIC of the supervisory PC. As was expected, after the program detected the network failure, it waited for 5 seconds before attempting to resend the file. If no response was received in the following 5 seconds, the program displayed a `network failure` error message to the user notifying him or her of the incidence. The tests were repeated for time periods less than 10 seconds whereby the user did not receive any error notifications since the program issued resubmissions which went through successfully, and for time periods more than 10 seconds which resulted in the program terminating the transfer and notifying the user of the failure.

Communication with the database server (datastore) consisted of file transfers to and from a remote directory structure. This is because implementation details of the database server have not been fully specified as such a remote directory structure has been assumed. The module responsible for such communication was well designed as an independent black-box subsystem that could easily be replaced while still maintaining the overall functionality of the supervisory system. It consists of one main method/function that takes as inputs, a pointer to the database server's RPC handle, the name of the file, and a string specifying whether to retrieve or store the file. The method then returns a string detailing the transfer result. Figure 6.3 outlines the algorithm adopted when transferring files between systems. It consists of three distinct steps;

1: The client queries the server if the specified file is available. The file

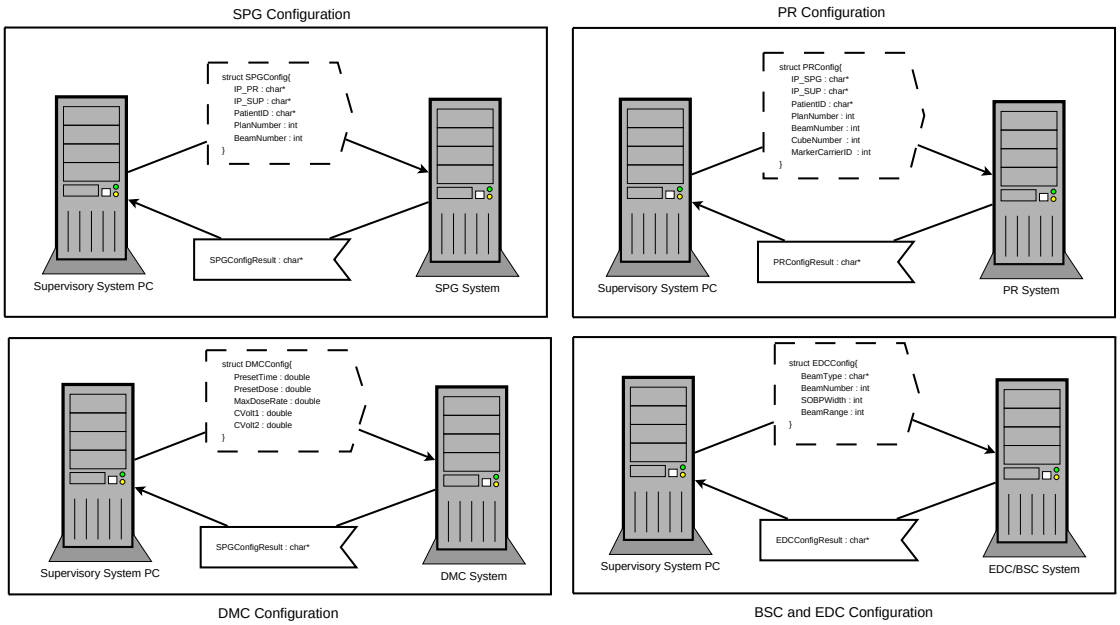

Figure 6.2: Systems Configuration data

name is send as input to the server.

2: The server responds by sending a file header datastructure consisting of the following information: the name of the file being queried, an MD5 checksum of the file, a timestamp value of the time and date the file was last modified, an integer specifying the size of the file, and a file-exists flag which is set to 1 if the file exists or 0 if not. If the file does not exist on the server then only the flag and the filename parameters are set. Other parameters are left as null values.

3: If the file-exist flag was raised in the header structure, the actual file transfer begins. The transfer consists of the client and server exchanging chunks of data constituting the file. The data is organised into a fileBody datastructure comprising of a variable size data block of up to 10Mega bytes if large image files are being exchanged, the number of bytes send in that particular chunk, and the position in the file where the data was last read from or written to.

All file transfers between the supervisory system and the datastore server were successful when tested with different kinds of files, from text files to image files as well as compressed files.

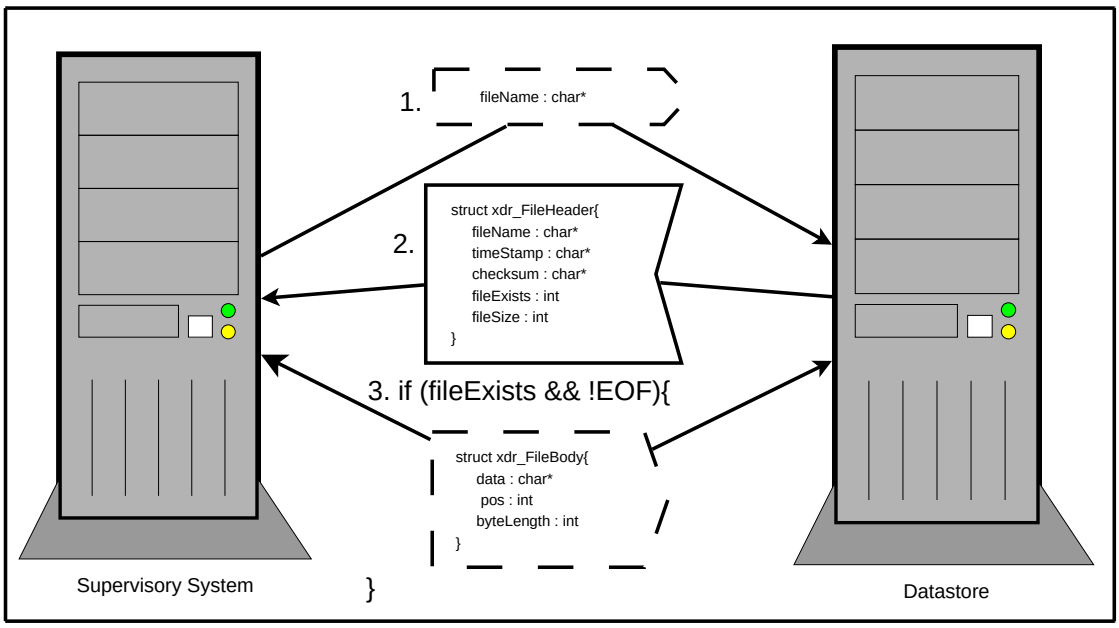

Figure 6.3: File Transfer Algorithm

File Transfer Proxy

When acting as a proxy between other systems and the datastore, the supervisory system performs a dual role of being a server (to other systems) and a client (to the datastore). The proxy algorithm works as follows:

1: The client queries the supervisory system if the specified file is available. The file name is send as input to the supervisory system.

2: The supervisory program queries the datastore whether the requested file is available. The server responds by sending a file header datastructure consisting of the name of the file being queried, an MD5 checksum of the file, a timestamp value of the time and date the file was last modified, an integer specifying the size of the file, and a file-exists flag which is set to 1 if the file exists or 0 if not. If the file does not exist on the server then only the flag and the filename parameters are set. Other parameters are left as null values.

3: The supervisory system forwards the header structure to the calling client.

4: If the file-exist flag was raised in the header structure, the actual file transfer begins. The transfer consists of the client, the supervisory system and datastore server exchanging the fileBody datastructure comprising of a variable size data block of up to 10Mega bytes if large image files

are being exchanged, the number of bytes send in that particular chunk, and the position in the file where the data was last read from or written to. The role of the supervisory system in this step is to receive and forward the file-body structure between the datastore and the requesting client program.

File transfers between the client program (SPG or PR), supervisory system and the datastore server were successful when tested with different kinds of files, from text files to image files as well as compressed files.

Attribute-based Access-control

The program has an internal MySQL database of all registered users to be granted access to the system. Each user has a unique username and password used to gain access to the system. The users have also been grouped into different groups, *clinic*, *physics* and *admin*, which determine what views and functionality of the system they can access.

When a user logs into the system, the program runs a check on the provided username and password against the entire database. If a match is found, the program retrieves the group ID associated with that user and then loads the appropriate view/display for that group. Different users registered under all groups were used to test the attribute-view functionality of the supervisory system with successful results.

Use of barcode scanner

The Zebex Z-3010USB barcode scanner used to test the system was also a simple device to interface to. This is because the scanner inputs data to the system as if it were a computer keyboard. Thus if an input widget is highlighted on the program's display and a barcode is scanned, the text-equivalent of that barcode is immediately entered into the widget as if the input came from a computer keyboard device. Block 19 of Figure 5.6 represents the barcoding section of the program. A list of patient-specific treatment equipment with check-boxes next to them is displayed to the user. At the top center of the display window is a tiny text-edit widget which is highlighted immediately after the dialog box is presented to the user. As the user scans the treatment equipment being used, the text representation of the barcode is inputted into the text-edit wigdet as if it were being typed-in through a computer keyboard. Using Qt's signals and slots mechanism, the program takes the string representation of the barcode from the text-edit widget (after the *hasChanged flag* of the text-edit item has been raised) and compares it with barcode strings of equipment to be used when treating that specific patient, which were parsed

into program memory from the treatment plan file. If a match is found, the program places a check on the check-box next to that device on the display. The barcoding section is completed only when all devices on the display list have been checked out, or the user decides to quit the program thus terminating treatment and returning to the Main Menu.

Proper use of the barcode scanner was tested with custom-made barcode strings that represented different patient-specific devices to be used during patient treatment. The strings were stored in the patient treatment record file which was then parsed into program memory during treatment test procedure. Only after scanning all required equipment did the program allow the user to continue with patient treatment.

Electronically Interlocked with entire Proton Therapy Control System

Proper support of the TSB simulator rig demonstrates that the supervisory system is electronically interlocked with the rest of the proton control system. When conducting the test procedure for patient treatment, the following lines were properly raised;

TSB Line	Raised/Droppen when?
SS OK	Raised after properly configuring other subsystems. Block 10 of Figure 5.10.
SABUS OK	Dropped at any time if the system incurs a hardware failure after the watchdog timer has been started during treatment i.e. anywhere between blocks 2 and 9 of Figure 5.8.

Table 6.1: TSB Lines controlled by the supervisory system

The *SS OK* line was properly raised during normal program execution when carrying out the test procedure. To test functionality of the SABUS OK line, the system was intentionally powered off during treatment procedure after the watchdog timer had been started. This caused the program to stop patting the watchdog thus causing it to time-out, which then dropped the *SABUS OK* line.

Non-Functional Tests

As was required, the system has been implemented using Qt C++ widget sets, and uses the PCI technology to inteface to the Therapy Safety Bus. The pro-

gram also uses the Sun-RPC client-server model to configure other subsystems of the Proton Control System. Also, due to the intensive failure recovery measures inherent in the system's design, the reliability and dependability of the system can be well assumed eventhough it is only time and proper testing that can rightly determine such qualities for a system.

Chapter 7

Conclusion

The first version of the proton therapy supervisory system for iThemba LABS has been designed, implemented and tested with custom-made dummy systems representing the Proton Control System. The implemented system has been shown to meet some of its requirements of coordinating and configuring all other systems with beam parameters and treatment information needed to ensure a correct and safe irradiation of a patient during treatment. Also, it has been outlined how the system collects, records and verifies all the necessary treatment information after any successful or otherwise irradiation session, and how it acts as a gateway/proxy between all other subsystems of the proton therapy control system and the radiotherapy data-store server.

Recommendations for Future Work

However satisfactory the current version of the supervisory system may be, there is still more work to be done before the system can be considered complete and ready for patient use.

Firstly, the current version still does not entirely support the proposed dose algorithm. Currently, the program can only increment or decrement the preset dose of the same field in the next fraction if there was a dose overrun or under-dose during treatment delivery. Thus the current version does not support blocks 7 and 18 of Figure 5.7 (Field Delivery loop). To support such functionality would require the program to loop through all remaining fractions locating the same field as the current one been delivered, and updating the preset dose parameter accordingly. The solution is, in principle quite simple though it depends on the structure and format of the treatment record file. The current format of the treatment record is a simple list of attribute-value pairs which are parsed into a multimap datastructure in program memory. Iterating through such a datastructure and easily locating separate fractions and corresponding fields is not as intuitive as it should be. Thus the format

of the treatment record file needs to be looked into and designed to make it easy to parse into a custom-made memory object which can seamlessly be iterated over to locate separate fractions and fields. One format that could be employed is the eXtensible Mark-up Language (XML) format. Using this format the treatment record would be organized into XML tags or elements representing different parameters of the treatment record file, with associated values. Such a format would be well-formed hence it would be easy to parse the record into a user-defined object belonging to a well-defined class that could easily be iterated over. Developing a Boost.Spirit parser for an XML document should also prove to be quite simple due to the well-formed nature of XML and extensive documentation and support provided by Boost.

Bibliography

[1] NAPT: How proton therapy works, July 2011.
Available at: `http://www.proton-therapy.org/howit.htm`

[2] Flanz, J., Delaney, T., Kooy, H., Rosenthal, S., Titt, U. and MGH-NPTC: Treating patients with the nptc accelerator based proton treatment facility. *Proceedings of the 2003 Particle Accelerator Conference*, 2008.

[3] Giordanengo, S.: The cnao system to monitor and control hadron beams for therapy. *2008 IEEE Nuclear Science Symposium Conference Record.*, 2008.

[4] Rapoo, B.: Development of a patient set-up verification (psvd) for radiation therapy treatment. *Bioengineering Conference, Proceedings of the IEEE 28th Annual Northeast*, 2002.

[5] de Kock, E.: Hardware specifications and functional design of the proton therapy control system. Tech. Rep., iThemba LABS, 2010.

[6] Carstens, J.: *Fast generation of digitally reconstructed radiographs for use in 2D-3D image registration.* MSc, University of Stellenbosch, 2008.

[7] Wagener, D.W.: *Feature Tracking and Pattern Registration.* MScEng, University of Stellenbosch, November 2003.

[8] van Wyk, B.-M.M.: *Verifying Stereo Vision using Structure from Motion.* MScEng, University of Stellenbosch, 2008.

[9] Ts'oeu, M.: *Proton Beam Steering Control System for High Precision Radiotherapy at iThemba LABS: An Investigation on Actuator Saturation Constraints.* Master's thesis, University of Cape Town, 2008.

[10] Schroeder, S.: *Proton Therapy Operations and Treatment Procedure manual.* iThemba LABS, 2002.

[11] J.Katuin: Proton therapy treatment room controls using a linux control system. In: *Proceedings of the 2003 Particle Accelerator Conference.* IEEE, http://ieeexplore.ieee.org/, 2003.

[12] Blackmore, E., Evans, B. and Mouat, M.: Operation of the triumf proton therapy facility. *Proceedings of the 1997 Particle Accelerator Conference*, vol. 3, 1997.

[13] de Kock, E. and van Tubbergh, C.: Conceptual design of the proton beam treatment facility at ithemba labs. Tech. Rep., iThemba LABS, June 2005.

[14] de Kock, E.: The mechanical and mathematical aspects related to the control system for the proton therapy chair and treatment collimator. Tech. Rep., iThemba LABS, August 2008.

[15] van der Bijl, L.: *Verification of patient position for proton therapy using Portal X-Rays and Digitally Reconstructed Radiographs*. Master's thesis, University of Stellenbosch, 2006.

[16] Callaghan, C.: Communication between the spg and the portal x-ray systems during configuration. Tech. Rep., iThemba LABS, April 2009.

[17] de Kock, E. and Muller, N.: Treatment and simulation algorithms for the new proton supervisory system. Tech. Rep., iThemba LABS, March 2010.

[18] de Kock, E. and Carstens, C.: Command structure of the network communications in a distributed proton therapy control system. Tech. Rep., iThemba LABS, 2007.

[19] Carstens, C. and Muller, N.: The new supervisory system. Tech. Rep., iThemba LABS, March 2010.

[20] Qhobosheane, S.: Design document of the new proton supervisory system. Tech. Rep., iThemba LABS, November 2011.

[21] Sommerville, I.: *Software Engineering*. 8th edn. Addison-Wesley, 2007.

[22] Coutrakon, G., Slater, J. and Ghebremedhin, A.: Design considerations for medical proton accelerators. *Proceedings of the 1999 Particle Accelerator Conference*, 1999.

[23] M, S.: Control system for the neutron therapy facility at fermilab. *Particle Accelerator Conference. Accelerator Science and Technology., Proceedings of the 1989 IEEE*, 1989.

[24] Sepulchre, R.: Pencil beam scanning: a dynamical approach to proton therapy. *IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, 2000.

[25] Swanepoel, M.: Process for the proton treatment of a patient. Tech. Rep., iThemba LABS, 2005.

[26] Abrahamsson, P., Salo, O. and Ronkainen, J.: *Agile software development methods: Review and analysis*. 1st edn. VVT Electronics, 2002.

[27] Intel: Intel core 2 quad processors.
Available at: `http://www.intel.com/products/processor/core2quad/index.htm`
`Last Accessed: November 2012`

[28] de Guzman, J. and Kaizer, H.: Boost.spirit 2.5 libraries: Documentation.
Available at: `http://www.boost.org/doc/libs/1_47_0/libs/spirit/doc/html/spirit/introduc`
`Last Accessed: November 2012`

Addendum A

Use of RPC Communication by the Supervisory System

RPC (Remote Procedure Call) is a powerful technique for constructing distributed, client-server based applications. With RPC, the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems in a network.

RPC makes the client/server model of computing more powerful and easier to program since it adopts the notion of conventional procedure calls, and hides the complex part of distributed communication from the programmer (this applies if the programmer employs the high-level RPC interface and not the detailed low-level interface).

Just like in a function call, when an RPC is made, the calling arguments are passed to the remote procedure and the client blocks, waiting for a response to be returned from the remote procedure. When the request arrives, the server calls a dispatch routine that performs the requested service, and sends the reply to the client. After the RPC call is completed, the client program continues. The following pieces of code demonstrate use of RPC by the supervisory system when configuring the dummy DMC system.

Shared Data Set

```
#ifndef SSSHAREDDATA_H
#define SSSHAREDDATA_H

#define SPGSERVER_NUM 0x31234568
#define SPGSERVER_VERSION 1

#define PRSERVER_NUM 0x31234566
#define PRSERVER_VERSION 1

#define DMCSERVER_NUM 0x31234565
#define DMCSERVER_VERSION 1
```

```
#define SERVER_NUM 0x31234567
#define SERVER_VERSION 1

#define DBSERVER_NUM 0x31234569
#define DBSERVER_VERSION 1

#define SWITCH 1
#define IMP_FILE 3
#define LIST_FILES 31
#define LIST_FILES_SIZES 32
#define IMP_FILE_CONT 33
#define EXP_FILE 4
#define DEL_FILE 5
#define GET_CONFIG 6
#define SET_CONFIG 7
#define LOCK 8
#define UNLOCK 9
#define SPGSENDCONFIG 10
#define SPGGETCONFIG 101
#define PRSENDCONFIG 11
#define PRGETCONFIG 111
#define DMCSENDCONFIG 12
#define MONITORA 21
#define MONITORB 22
#define INCVOLT 23
#define INCVOLT2 24
#define INSETDOSE 25
#define INSETTIME 26
#define INMAXRATE 27
#define INDOSD 28
#define CONTES 29
#define CONSTOPONA 30
#define CONSTOPONB 230
#define OUTCVOLT 231
#define OUTCVOLT2 241
#define OUTSETDOSE 251
#define OUTFCDOSE1 252
#define OUTFCDOSE2 253
#define OUTDOSE1 254
#define OUTDOSE2 255
#define OUTSETTIME 261
#define OUTMAXRATE 271
#define OUTMAXARATE 272
```

```
#define OUTMAXBRATE 273
#define OUTARATE 274
#define OUTBRATE 275
#define OUTDOSD 281
#define EXIT 100
#define STATUS 411

#define ONE_MEG 1048576
#define TEN_MEG 10485760

#include <iostream>
#include <QString>
#include <QDateTime>
#include <QCryptographicHash>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <rpc/rpc.h>
#include <unistd.h>
#include <time.h>
#include <iomanip>

using namespace std;

struct SPGConfig {
        char* IPSUP;
        char* IPPR;
        char* patientID;
        int planNum;
        int beamNum;
};
typedef struct SPGConfig SPGConfig;

struct PRConfig{
    char *IPSUP;
    char *IPSPG;
    char *patientID;
    int planNum;
    int beamNum;
    int cubeNum;
    int markerCarrierID;
};
typedef struct PRConfig PRConfig;
```

```
struct fileHeader {
        int fileExists;
        int fileSize;
        char *fileName;
        char *timestamp;
        char *checksum;
};
typedef struct fileHeader fileHeader;

struct fileBody {
        int pos;
        int byteLength;
        char* data;
};
typedef struct fileBody fileBody;

struct monitoredDevices {
        unsigned long SABUS;
        unsigned long HV;
        unsigned long SS;
        unsigned long RCS;
        unsigned long BCS;
        unsigned long DMC;
        unsigned long PHYSICSMode;
        unsigned long SOBPMode;
        unsigned long BITEBLOCKMode;
        unsigned long BEAMON;
        unsigned long BEAMOFF;
        unsigned long FC19OUTREQ;
        unsigned long BEAMONHOLDREQ;
};
typedef struct monitoredDevices monitoredDevices;

extern struct monitoredDevices monDevs;

struct field{
    int number,
        leaf_pairs,
        misc_field_data,
        irregular_field_num;

    double weight,
         iso_distance,
```

```
        MLC_rotation,
        MLC_thickness,
        leaf_width,
        monitor_units,
        radiological_depth,
        output_factor,
        tissue_mass_ratio,
        tissue_phantom_ratio,
        source_skin_dist,
        wedge_factor,
        relative_dose,
        energy,
        gantry_angle,
        patient_support_angle,
        beam_lim_device_angle,
        field_width,
        field_height,
        wedge_angle;


    std::string name,
        type,
        irradiation_device,
        radiation_type,
        MLC_name,
        MLC_variable_leaf_width,
        beam_group_type,
        flags,
        dose_data,
        target_point,
        beam_lim_device_type,
        wedge_orientation,
        irregular_field_file,
        applicator_name;

};
typedef struct field field;

struct Plan{
    int number,
    fractionsNumber,
    fieldsNumber;

    double prescribedDose,
```

```
        normalizationDose;

    std::string patientName,
        patientID,
        therapist,
        doseCalcSource;

    field* treatmentFields;

};
typedef struct Plan Plan;

struct indicator
{
    int value;
    QDateTime date;
};
typedef struct indicator indicator;
typedef indicator timeInfo;
typedef indicator voltFactor;

struct error
{
    indicator error_indicator;
    QString reason;
};
typedef struct error error;

struct doseData
{
    double dose;
    QDateTime date;
};
typedef struct doseData doseData;

struct xrayData
{
    QString * fileName;
    QDateTime * date;
};
typedef struct xrayData xrayData;

extern bool_t xdr_PRConfig (XDR *xdrs, PRConfig *objp);
extern bool_t xdr_SPGConfig (XDR *xdrs, SPGConfig *objp);
```

```
extern bool_t xdr_fileHeader (XDR *xdrs, fileHeader *objp);
extern bool_t xdr_fileBody (XDR *xdrs, fileBody *objp);

extern int INTERRUPT_FLAG;
extern int STATE_TRANSITION_FLAG;
extern int STATE_IDENTIFY_FLAG;

#endif // SSSHAREDDATA_H
```

Supervisory System Client code

```
. . .
bool simulationwindow::initializeDMCClient()
{
    char Host[50];
    char Protocol[4];
    u_long Program, Version;

    strcpy(Host, "196.21.126.124");
    strcpy(Protocol, "tcp");
    Program = DMCSERVER_NUM;
    Version = DMCSERVER_VERSION;
    DMCClient = clnt_create(Host, Program, Version, Protocol);
    if(DMCClient != 0)
        return true;
    else
        return false;

}
   . . .

int simulationwindow::configureDMC(int cvolt, int cvolt2, double presetDose,
   int presetTime, int maxrate, double dosd)
{
    if(!initializeDMCClient())
        return -1;  // DMC RPC Client not initialized
    cmdINCVOLT(cvolt);
    if(cmdOUTCVOLT() != cvolt)
        return -2;  // CVOLT factor not set
    cmdINCVOLT2(cvolt2);
    if(cmdOUTCVOLT2() != cvolt2)
        return -3;  // CVOLT2 factor not set
    cmdINSETDOSE(presetDose);
    if(cmdOUTSETDOSE() != presetDose)
```

```
        return -4;  // Preset Dose not set
    cmdINSETTIME(presetTime);
    if(cmdOUTSETTIME() != presetTime)
        return -5;  // Preset Time not set
    cmdINMAXRATE(maxrate);
    if(cmdOUTMAXRATE() != maxrate)
        return -6;  // Maximum dose rate not set
    cmdINDOSD(dosd);
    if(cmdOUTDOSD() != dosd)
        return -7;  // Dose difference not set

    return 0;   // DMC Successfully configured!
}

void simulationwindow::cmdINCVOLT(int cvolt)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INCVOLT, (xdrproc_t) xdr_int, (char *) &cvolt,
      (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}

void simulationwindow::cmdINCVOLT2(int cvolt2)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INCVOLT2, (xdrproc_t) xdr_int, (char *) &cvolt2,
          (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}

void simulationwindow::cmdINSETDOSE(double dose)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INSETDOSE, (xdrproc_t) xdr_double, (char *) &dose,
      (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}
```

```
void simulationwindow::cmdINSETTIME(int time)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INSETTIME, (xdrproc_t) xdr_int, (char *) &time,
      (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}

void simulationwindow::cmdINMAXRATE(int maxrate)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INMAXRATE, (xdrproc_t) xdr_int, (char *) &maxrate,
      (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}

void simulationwindow::cmdINDOSD(double dosd)
{
    struct timeval Timeout;
    Timeout.tv_sec = 100;
    Timeout.tv_usec = 0;
    char* ParamOut = 0;
    clnt_call(DMCClient,INDOSD, (xdrproc_t) xdr_double, (char *) &dosd,
        (xdrproc_t) xdr_wrapstring, (char *) &ParamOut, Timeout);
}
. . .
```

DMC Server code

Header file: rpcrequesthandler.h

```
#ifndef RPCREQUESTHANDLER_H
#define RPCREQUESTHANDLER_H

#include <QThread>
#include "SSSHaredData.h"
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <rpc/rpc.h>
```

```
struct DMCData{
    int cvolt,
        cvolt2,
        time,
        maxrate,
        maxarate,
        maxbrate,
        arate,
        brate,
        fcdose1,
        fcdose2;
    double dose,
          dose1,
          dose2,
          dosd;
};
typedef struct DMCData DMCData;

class RPCRequestHandler : public QThread
{
public:
    RPCRequestHandler();
    ~RPCRequestHandler();
    void terminateServer();

protected:
    void run();
    void initializeServer();

private:
    SVCXPRT *Server;

};


extern void dispatch_me(struct svc_req *Request, SVCXPRT *Server);
extern DMCData *dmcData;

#endif // RPCREQUESTHANDLER_H
```

Source file: rpcrequesthandler.cpp

```
#include "rpcrequesthandler.h"
#include <iostream>
#include <QString>
using namespace std;

DMCData *dmcData = new DMCData;

RPCRequestHandler::RPCRequestHandler()
{
    //initializeServer();
}

RPCRequestHandler::~RPCRequestHandler()
{
    terminateServer();
}

void RPCRequestHandler::run()
{
    initializeServer();
}


void dispatch_me(struct svc_req *Request, SVCXPRT *Server)
{
    int result = 0;
    double fresult = 0;
    char* args;
    //QString args;
    PRConfig argss;
    argss.beamNum = 0;
    argss.planNum = 0;
    argss.IPSUP = 0;
    argss.IPSPG = 0;
    argss.patientID = 0;
    argss.cubeNum = 0;
    argss.markerCarrierID = 0;
    //int argsss;

    switch (Request->rq_proc)
    {
    case SWITCH:
        cout << "Switch COMMAND recieved: Entering Serialization State..." << end
        args= "OK";
```

```
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            cout << "Reply send successfully: " <<args<< endl;
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case STATUS:
        cout << "Status COMMAND recieved: Sending DMC Server status to client..."
        args = "UNKNOWN";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
     case CONTES:
        cout << "CON TES COMMAND recieved..." <<endl;
        args = "DMCOK";
        dmcData->dose1 = 0;
        dmcData->dose2 = 0;
        dmcData->time = 0;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
     case CONSTOPONA:
        cout << "CON STOPON A COMMAND recieved..." <<endl;
        args = "OK";
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
```

```
        break;
    case CONSTOPONB:
        cout << "CON STOPON B COMMAND recieved..." <<endl;
        args = "OK";
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case LOCK:
        cout << "Lock COMMAND recieved: Locking SPG Server to client..." << endl;
        args = " LOCKED";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case UNLOCK:
        cout << "Unlock COMMAND recieved: Locking SPG Server from client..." << e
        args = " UNLOCKED";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case DMCSENDCONFIG:
        cout << "SendConfig COMMAND recieved: Reading inputs..." << endl;
        args = " PRConfig received";
        svc_getargs(Server, (xdrproc_t) xdr_PRConfig, (char *) &argss);
        cout<<"PRConfig recieved:"<<endl<<"IPsup = "<<argss.IPSUP<<endl;
        cout<<"IPspg = "<<argss.IPSPG<<endl;
        cout<<"PatientID = "<<argss.patientID<<endl;
        cout<<"Beam Number = "<<argss.beamNum<<endl;
        cout<<"Plan Number = "<<argss.planNum<<endl;
        cout<<"Marker Carrier ID = "<<argss.markerCarrierID<<endl;
```

```
        cout<<"Cube Number = "<<argss.cubeNum<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTDOSE1:
        {
        cout << "OUT DOSE1 COMMAND recieved..." << endl;
        QTime midnight(0, 0, 0);
        qsrand(midnight.secsTo(QTime::currentTime()));

        result = qrand() % 10;
        dmcData->dose1 = (double)result;
        cout<<"dmcData->dose1: "<<dmcData->dose1<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &dmcData->dose1)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    }
    case OUTDOSE2:
        {
        cout << "OUT DOSE2 COMMAND recieved..." << endl;
        QTime midnight(0, 0, 0);
        qsrand(midnight.secsTo(QTime::currentTime()));

        result = qrand() % 10;
        dmcData->dose2 = (double)result;
        cout<<"dmcData->dose2: "<<dmcData->dose2<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &dmcData->dose2)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
```

```
    }
    case INCVOLT:
        cout << "IN CVOLT COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_int, (char *) &result);
        dmcData->cvolt = result;
        cout<<"dmcData->cvolt: "<<dmcData->cvolt<<endl;
        args = "CVOLT SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case INCVOLT2:
        cout << "IN CVOLT2 COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_int, (char *) &result);
        dmcData->cvolt2 = result;
        cout<<"dmcData->cvolt2: "<<dmcData->cvolt2<<endl;
        args = "CVOLT2 SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case INSETDOSE:
        cout << "IN SETDOSE COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_double, (char *) &fresult);
        dmcData->dose = fresult;
        cout<<"dmcData->dose: "<<dmcData->dose<<endl;
        args = "DOSE SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case INSETTIME:
```

```
        cout << "IN SETTIME COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_int, (char *) &result);
        dmcData->time = result;
        cout<<"dmcData->time: "<<dmcData->time<<endl;
        args = "TIME SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case INMAXRATE:
        cout << "IN MAXRATE COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_int, (char *) &result);
        dmcData->maxrate = result;
        cout<<"dmcData->maxrate: "<<dmcData->maxrate<<endl;
        args = "MAXRATE SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case INDOSD:
        cout << "IN DOSD COMMAND recieved..." <<endl;
        svc_getargs(Server, (xdrproc_t) xdr_double, (char *) &fresult);
        dmcData->dosd = fresult;
        cout<<"dmcData->dosd: "<<dmcData->dosd<<endl;
        args = "DOSD SET";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTCVOLT:
        cout << "OUT CVOLT COMMAND recieved..." <<endl;
        result = dmcData->cvolt;
```

```
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTCVOLT2:
        cout << "OUT CVOLT2 COMMAND recieved..." <<endl;
        result = dmcData->cvolt2;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTSETDOSE:
        cout << "OUT SETDOSE COMMAND recieved..." <<endl;
        fresult = dmcData->dose;
        cout<<"result: "<<fresult<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_double, (char *) &fresult)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTSETTIME:
        cout << "OUT SETTIME COMMAND recieved..." <<endl;
        result = dmcData->time;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
```

```
        break;
    case OUTMAXRATE:
        cout << "OUT MAXRATE COMMAND recieved..." <<endl;
        result = dmcData->maxrate;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTMAXARATE:
        cout << "OUT MAXARATE COMMAND recieved..." <<endl;
        result = dmcData->maxarate;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTMAXBRATE:
        cout << "OUT MAXBRATE COMMAND recieved..." <<endl;
        result = dmcData->maxbrate;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTDOSD:
        cout << "OUT DOSD COMMAND recieved..." <<endl;
        fresult = dmcData->dosd;
        cout<<"result: "<<fresult<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_double, (char *) &fresult)){
            result = 0;
        }
```

```
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTARATE:
        cout << "OUT A RATE COMMAND recieved..." <<endl;
        result = dmcData->arate;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case OUTBRATE:
        cout << "OUT B RATE COMMAND recieved..." <<endl;
        result = dmcData->brate;
        cout<<"result: "<<result<<endl;
        if(svc_sendreply(Server, (xdrproc_t) xdr_int, (char *) &result)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    case EXIT:
        cout << "Exit COMMAND recieved: Exiting Serialization State..." << endl;
        args = " OK";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    default:
        cout << "Unknown message recieved."<< endl;
        args = "Unknown Command.";
        if(svc_sendreply(Server, (xdrproc_t) xdr_wrapstring, (char *) &args)){
```

```
            result = 0;
        }
        else{
            cout << "Reply NOT send. " << endl;
            result = -1;
        }
        break;
    }
}

void RPCRequestHandler::initializeServer()
{
    cout << "Inside InitializeServer()...." << endl;
    int TCPSocket;
    struct sockaddr_in Address;
    //SVCXPRT *Server;

    dmcData->arate = 0;
    dmcData->brate = 0;
    dmcData->cvolt = 0;
    dmcData->cvolt2 = 0;
    dmcData->dosd = 0;
    dmcData->dose = 0;
    dmcData->dose1 = 0;
    dmcData->dose2 = 0;
    dmcData->fcdose1 = 0;
    dmcData->fcdose2 = 0;
    dmcData->maxarate = 0;
    dmcData->maxbrate = 0;
    dmcData->maxrate = 0;
    dmcData->time = 0;

    TCPSocket = socket(PF_INET, SOCK_STREAM, 0);
    Address.sin_family = AF_INET;
    Address.sin_port = htonl(7775);
    Address.sin_addr.s_addr = INADDR_ANY;
    bind(TCPSocket, (sockaddr*) &Address, sizeof(Address));
    Server = svctcp_create(TCPSocket, 0, 0);
    cout<<"Before service register..."<<endl;
    cout <<svc_register(Server, DMCSERVER_NUM, DMCSERVER_VERSION,
      &dispatch_me, IPPROTO_TCP)<<endl;

    //GlobalVar = -1;
```

```
    svc_run();
}

void RPCRequestHandler::terminateServer()
{
    svc_destroy(Server);
    svc_unregister(DMCSERVER_NUM, DMCSERVER_VERSION);
    delete dmcData;
}

bool_t xdr_SPGConfig (XDR *xdrs, SPGConfig *objp)
{
    if (!xdr_wrapstring (xdrs, &objp->IPSUP)){
        return FALSE;
    }
    if (!xdr_wrapstring (xdrs, &objp->IPPR)){
        return FALSE;
    }
    if (!xdr_wrapstring (xdrs, &objp->patientID)){
        return FALSE;
    }
    if (!xdr_int (xdrs, &objp->beamNum)){
        return FALSE;
    }
    if (!xdr_int (xdrs, &objp->planNum)){
        return FALSE;
    }

    return TRUE;
}

bool_t xdr_PRConfig (XDR *xdrs, PRConfig *objp)
{
    if (!xdr_wrapstring (xdrs, &objp->IPSUP))
        return FALSE;
    if (!xdr_wrapstring (xdrs, &objp->IPSPG))
        return FALSE;
    if (!xdr_wrapstring (xdrs, &objp->patientID))
        return FALSE;
    if (!xdr_int (xdrs, &objp->planNum))
        return FALSE;
    if (!xdr_int (xdrs, &objp->beamNum))
        return FALSE;
    if (!xdr_int (xdrs, &objp->cubeNum))
```

```
        return FALSE;
    if (!xdr_int (xdrs, &objp->markerCarrierID))
        return FALSE;
    return TRUE;
}

bool_t xdr_fileBody (XDR *xdrs, fileBody *objp)
{
    cout<<"Entering xdr_fileBody..."<<endl;
    if (!xdr_int (xdrs, &objp->pos))
        return FALSE;
    if (!xdr_int (xdrs, &objp->byteLength))
        return FALSE;
    if (!xdr_wrapstring (xdrs, &objp->data))
        return FALSE;
    cout<<"Leaving xdr_fileBody."<<endl;
    return TRUE;
}

bool_t xdr_fileHeader (XDR *xdrs, fileHeader *objp)
{
    cout<<"Entering xdr_fileHeader..."<<endl;
    if (!xdr_int (xdrs, &objp->fileExists)){
        cout<<"fileExists not int"<<endl;
        return FALSE;
    }
    if (!xdr_int (xdrs, &objp->fileSize)){
        cout<<"fileSize not int"<<endl;
        return FALSE;
    }
    if (!xdr_wrapstring (xdrs, &objp->fileName)){
        cout<<"fileName not char*"<<endl;
        return FALSE;
    }
    if (!xdr_wrapstring (xdrs, &objp->timestamp)){
        cout<<"timestamp not char*"<<endl;
        return FALSE;
    }
    if (!xdr_wrapstring (xdrs, &objp->checksum)){
        cout<<"checksum not char*"<<endl;
        return FALSE;
    }
    cout<<"Leaving xdr_fileHeader."<<endl;
    return TRUE;
```

```
}
```

Boost.Spirit Parsers

Parsers.h

```
#ifndef PARSERS_H
#define PARSERS_H

#include <boost/config/warning_disable.hpp>
#include <boost/spirit/include/qi.hpp>
#include <boost/spirit/include/phoenix_core.hpp>
#include <boost/spirit/include/phoenix_operator.hpp>
#include <boost/spirit/include/phoenix_object.hpp>
#include <boost/fusion/include/adapt_struct.hpp>
#include <boost/fusion/include/io.hpp>
#include <boost/fusion/include/std_pair.hpp>

#include <map>


namespace parsers
{
    namespace qi = boost::spirit::qi;
    namespace ascii = boost::spirit::ascii;

    typedef std::multimap<std::string,std::string> pairs_type;

    using qi::lit;
    using ascii::char_;
    using qi::eol;

    ////////////////////////////////////////////////////
    //  Our supervisory treatment plan (*.sup) parser
    ////////////////////////////////////////////////////

    template <typename Iterator>
            struct supervisoryTreatmentPlan : qi::grammar<Iterator, pairs_type()>
    {
        supervisoryTreatmentPlan() : supervisoryTreatmentPlan::base_type
         (supervisoryTreatmentPlanStart)
        {
            supervisoryTreatmentPlanStart %=  supervisoryTreatmentPlanPair >>
```

```
            *(+eol >> supervisoryTreatmentPlanPair);
        supervisoryTreatmentPlanPair  %=  supervisoryTreatmentPlanKey >>
            -(-lit("   ") >> supervisoryTreatmentPlanValue);
        supervisoryTreatmentPlanKey   %=  (+qi::space | !eol) >>
            *(char_ - (+lit("   ") | +eol));
        supervisoryTreatmentPlanValue %=  +(char_ - eol);

    }

    qi::rule<Iterator, pairs_type()> supervisoryTreatmentPlanStart;
    qi::rule<Iterator, std::pair<std::string, std::string>()>
       supervisoryTreatmentPlanPair;
    qi::rule<Iterator, std::string()> supervisoryTreatmentPlanKey,
       supervisoryTreatmentPlanValue;

};
//]


////////////////////////////////////////
//  Our treatment plan (*.PLN) parser
////////////////////////////////////////

template <typename Iterator>
        struct treatmentPlan : qi::grammar<Iterator, pairs_type()>
{
    treatmentPlan() : treatmentPlan::base_type(treatmentPlanStart)
    {
        treatmentPlanStart %=  treatmentPlanPair >>
            *(+eol >> treatmentPlanPair);
        treatmentPlanPair  %=  treatmentPlanKey >>
            -(-lit("   ") >> treatmentPlanValue);
        treatmentPlanKey   %=  (+qi::space | !eol) >>
           *(char_ - (+lit("   ") | +eol));
        treatmentPlanValue %=  +(char_ - eol);

    }

    qi::rule<Iterator, pairs_type()> treatmentPlanStart;
    qi::rule<Iterator, std::pair<std::string, std::string>()>
     treatmentPlanPair;
    qi::rule<Iterator, std::string()> treatmentPlanKey,
       treatmentPlanValue;
```

```
    };
    //]

    ////////////////////////////////////////////////////////////
    //  Our Supervisory Configuration file (Config.sup) parser
    ////////////////////////////////////////////////////////////

    template <typename Iterator>
            struct config : qi::grammar<Iterator, pairs_type()>
    {
        config() : config::base_type(configStart)
        {
            configStart %=  configPair >> *(+eol >> configPair);
            configPair  %=  configKey >> -(-lit('=') >> configValue);
            configKey   %=  *(char_ - (+lit('=') | +eol));
            configValue %=  +(char_ - eol);

        }

        qi::rule<Iterator, pairs_type()> configStart;
        qi::rule<Iterator, std::pair<std::string, std::string>()> configPair;
        qi::rule<Iterator, std::string()> configKey, configValue;

    };
    //]

////////////////////////////////////////////////////////////////////////////
//  Patient record file (patientID || planNumber || treatmentUnit.rec) parser
////////////////////////////////////////////////////////////////////////////

    template <typename Iterator>
            struct rec : qi::grammar<Iterator, pairs_type()>
    {
        rec() : rec::base_type(recStart)
        {
            recStart %=  recPair >> *(+eol >> recPair);
            recPair  %=  recKey >> -(-lit(';') >> recValue);
            recKey   %=  *(char_ - (+lit(';') | +eol));
            recValue %=  +(char_ - eol);

        }

        qi::rule<Iterator, pairs_type()> recStart;
        qi::rule<Iterator, std::pair<std::string, std::string>()> recPair;
```

```
        qi::rule<Iterator, std::string()> recKey, recValue;

    };
    //]
}

#endif // PARSERS_H
```

Dose Algorithm

The following sequence of instructions embodies the dose algorithm for a given treatment plan:

1. Before starting the treatment for the first time, initialize the treatment record by setting and recording the following parameters in the treatment record for $\alpha = 1, \ldots, M$ and $j = 1, \ldots, N$:

   a) Set and record the cumulative delivered doses as $d^s_{\alpha,\,j} \leftarrow 0$.

   b) Set and record the delivered doses as $d^r_{\alpha,\,j} \leftarrow 0$.

   c) Set and record the preset doses as $d^p_{\alpha,\,j} \leftarrow d_\alpha$.

   d) Set and record the binary treatment flags $P_{\alpha,\,j} \leftarrow 0$ and $F_{\alpha,\,j} \leftarrow 0$.

2. Determine the next fraction i for which the treatment must be completed. This is done by first loading the flags $F_{\alpha,\,j}$ for $\alpha = 1, \ldots, M$ and $j = 1, \ldots, N$ (if they are not yet loaded), and then searching for the smallest value of i such that $F_{\alpha,\,i} = 0$ for at least one value of $\alpha \in [1, M]$. If $F_{\alpha,\,N} = 1$ for all $\alpha = 1, \ldots, M$, then the entire treatment pertaining to the given treatment plan is completed. No further irradiations for this treatment plan are then allowed.

3. The user is allowed to select the field α that has to be administered for the current fraction. The user is only allowed to select a field α for which $F_{\alpha,\,i} = 0$.

4. For the treatment with the field α in the i-th fraction, use the following instructions:

   a) Assume that $\dot{D}^{ref}_{\alpha,\,i} \approx \dot{D}^{ref}_{nom}$ and set the theoretical dose overrun as $\Delta d^{th}_\alpha \leftarrow O_\alpha \, \dot{D}^{ref}_{nom} \, \Delta t$.

   b) Load the values $d^s_{\alpha,\,i}$, $d^r_{\alpha,\,i}$, $d^p_{\alpha,\,i}$, and $P_{\alpha,\,i}$ if they are not already loaded.

   c) If $F_{\alpha,\,i} = 1$, then return to Step 2 above.

d) If $P_{\alpha,i} = 1$, then prompt the user to provide the approximate value $d^m_{\alpha,i}$ for $d^r_{\alpha,i}$ as obtained from the mechanical counters of the DMC, and then set and record $P_{\alpha,i} \leftarrow 0$ and $d^r_{\alpha,i} \leftarrow d^m_{\alpha,i}$.

e) **C**ase $[d^p_{\alpha,i} = d_\alpha$ and $d^r_{\alpha,i} = 0]$:
Set and record $d^p_{\alpha,i} \leftarrow d_\alpha - \Delta d^{th}_\alpha$.

Case $[d^r_{\alpha,i} \geq d^p_{\alpha,i}]$:
Set and record $d^s_{\alpha,i} \leftarrow d^s_{\alpha,i} + d^r_{\alpha,i}$, $d^p_{\alpha,i+1} \leftarrow d^p_{\alpha,i+1} - (d^s_{\alpha,i} - d_\alpha)$ and $F_{\alpha,i} \leftarrow 1$.

Case $[d^p_{\alpha,i} - d^r_{\alpha,i} > f\, d_\alpha$ for $i < N$ or $d^p_{\alpha,i} - d^r_{\alpha,i} > 0$ for $i = N]$:
Set and record $d^s_{\alpha,i} \leftarrow d^s_{\alpha,i} + d^r_{\alpha,i}$, $d^p_{\alpha,i} \leftarrow d^p_{\alpha,i} - d^r_{\alpha,i}$ and $d^r_{\alpha,i} \leftarrow 0$.

Case [Default]:
Prompt the user to determine if the dose difference $d^p_{\alpha,i} - d^r_{\alpha,i}$ should be distributed over the remaining fractions. If not required, then set and record $d^s_{\alpha,i} \leftarrow d^s_{\alpha,i} + d^r_{\alpha,i}$, $d^p_{\alpha,i} \leftarrow d^p_{\alpha,i} - d^r_{\alpha,i}$ and $d^r_{\alpha,i} \leftarrow 0$, otherwise set and record $F_{\alpha,i} \leftarrow 1$, $d^s_{\alpha,i} \leftarrow d^s_{\alpha,i} + d^r_{\alpha,i}$ and $d^p_{\alpha,j} \leftarrow d^p_{\alpha,j} + (d^p_{\alpha,i} - d^r_{\alpha,i})/(N - i)$ for all $j = i + 1, \ldots, N$.

f) If $F_{\alpha,i} \leftarrow 1$, stop SPG, and return to Step 2

g) Set $P_{\alpha,i} \leftarrow 1$ and then send the preset dose as $d^p_{\alpha,i}$ to the DMC by using the `IN SETDOSE`($d^p_{\alpha,i}$) command. The execution of the command `IN SETDOSE` must be checked with an `OUT SETDOSE` command. If this check fails, then the treatment must be terminated with a fatal error message.

h) The Supervisory System must remain in a suspended state until the user indicates that the irradiation may commence. A `CON TES` command must then be executed.

i) Record the values of $P_{\alpha,i}$ and $d^p_{\alpha,i}$ immediately when the `BEAM ON` signal is obtained.

j) When the beam is terminated, as indicated by the `BEAM OFF` signal, execute the `OUT DOSE1`(d_A) and `OUT DOSE2`(d_B) commands. If either or both these commands fail, then terminate the treatment with a fatal error message (the user should be prompted to record the dose values obtained from the mechanical counters of the DMC).

k) Ideally, the condition $d_\mathrm{A} \geq d_\mathrm{B}$ should be satisfied. If this condition holds true, then set $d^r_{\alpha,i} \leftarrow d_\mathrm{A}$. However, if $d_\mathrm{A} < d_\mathrm{B}$, then the user should be informed that Monitor B has overrun Monitor A. The user must then choose whether d_A or d_B should be taken as

the legitimate value for the delivered dose – set $d^r_{\alpha,i}$ to the selected value.

l) Set $P_{\alpha,i} \leftarrow 0$ and record $P_{\alpha,i}$ and $d^r_{\alpha,i}$ before returning to Step 4(e).

Dose distributions

Assuming that no dose differences were distributed over the remaining fraction for any of the treatment fractions, the dose algorithm formulated in Section `Dose Algorithm` above will result in the following dose distribution for the field α:

$$d^s_{\alpha,i} = \begin{cases} d_\alpha + \sum\limits_{j=1,\ odd}^{i-1} (\Delta d_{\alpha,j} - \Delta d_{\alpha,j+1}) + \Delta d_{\alpha,i} - \Delta d^{th}_\alpha & \text{if } i \text{ is odd} \\ d_\alpha - \sum\limits_{j=1,\ odd}^{i-1} (\Delta d_{\alpha,j} - \Delta d_{\alpha,j+1}) & \text{if } i \text{ is even.} \end{cases}$$

In this expression, the sums only run over odd values of j. Adding the cumulative doses for all the fractions yield

$$\sum_{i=1}^{N} d^s_{\alpha,i} = D_\alpha + \sum_{j=1,\ odd}^{N-1} (\Delta d_{\alpha,j} - \Delta d^{th}_\alpha)\,.$$

If we now assume that $\Delta d_{\alpha,j} = \Delta d^{th}_\alpha + \varepsilon_{\alpha,j}$ for all $j = 1, \ldots, N$, with $\varepsilon_{\alpha,j}$ being small random errors that are normally distributed around zero, then

$$d^s_{\alpha,i} - d_\alpha = \begin{cases} \sum\limits_{j=1,\ odd}^{i-1} (\varepsilon_{\alpha,j} - \varepsilon_{\alpha,j+1}) + \varepsilon_{\alpha,i} & \text{if } i \text{ is odd} \\ \sum\limits_{j=1,\ odd}^{i-1} (\varepsilon_{\alpha,j} - \varepsilon_{\alpha,j+1}) & \text{if } i \text{ is even,} \end{cases}$$

while

$$\sum_{i=1}^{N} d^s_{\alpha,i} - D_\alpha = \sum_{j=1,\ odd}^{N} \varepsilon_{\alpha,j}\,.$$

These results demonstrate that the objectives of the dose algortihm are indeed achieved, especially when N is large.

Dosimetric relationships

This section summarizes some of the dosimetric equations that are of importance to the supervisory system.

The prescribed dose per fraction is given by

$$d_\alpha = D_\alpha / M_u \,. \tag{7.0.1}$$

The preset treatment time required for the preset dose $d^p_{\alpha,\kappa}$ is given by

$$\Delta t^p_{\alpha,\kappa} = \mathrm{Max}((d^p_{\alpha,\kappa}/d_\alpha)\ \Delta t_\alpha,\ \Delta t_{min})\,, \tag{7.0.2}$$

whereas the maximum allowed difference between the readings of monitors A and B are given as

$$\Delta_{AB} = \mathrm{Max}(\%\Delta_{AB}\ d^p_{\alpha,\kappa}/100,\ \Delta^{min}_{AB})\,. \tag{7.0.3}$$

In these expressions, $\mathrm{Max}(x, y)$ returns the largest value of its two arguments x and y. Upon the normal stopping of the beam, the dose delivered can be expressed as

$$d^r_{\alpha,\kappa} = d^p_{\alpha,i} + \Delta d_{\alpha,\kappa}\,, \tag{7.0.4}$$

where $\Delta d_{\alpha,\kappa}$ is the dose overrun. This overrun is theoretically given by the estimate

$$\Delta d^{th}_\alpha \approx \dot{D}^{nom}_\alpha\ \Delta t_{sys} = O_\alpha\ \dot{D}^{nom}_{ref}\ \Delta t_{sys}\,. \tag{7.0.5}$$

The beam-stop response time Δt_{sys} of the beam delivery system can be estimated using the procedure outlined in Section `Measurement of the response time` below. The maximum dose-rate that should be tolerated during the delivery of the dose $d^p_{\alpha,\kappa}$ is given by

$$\dot{D}^{max}_\alpha = O_\alpha\ \dot{D}^{max}_{ref}\,. \tag{7.0.6}$$

Measurement of the response time

Suppose that the field α is the reference field and d_{ref} is the prescribed dose per fraction for this field. Thus, after M normally completed fractions, with $d^p_{\alpha,\kappa} = d^p_{ref,\kappa} = d_{ref}$ for all fraction $\kappa = 1, \ldots, M$, the average response time Δt_{sys} can be approximated from eqs (7.0.4) and (7.0.5) as

$$\Delta t_{sys} \approx \frac{1}{M}\sum_{i=1}^{M}(d^r_{ref,\kappa} - d_{ref})/\dot{D}_{ref,\kappa}\,, \tag{7.0.7}$$

with $\dot{D}_{ref,\kappa}$ denoting the average dose rate for the reference field during the fraction κ. Equation (7.0.7) represents a practical method for measuring Δt_{sys}. In this experiment, large values should be chosen for both M and d_{ref}. The large value for d_{ref} ensures long irradiation times ΔT_κ, so that the average dose rate $\dot{D}_{ref,\kappa}$ can be approximated as

$$\dot{D}_{ref,\kappa} \approx d_{ref}/\Delta T_\kappa\,. \tag{7.0.8}$$

By combining eqs (7.0.7) and (7.0.8), we find that

$$\Delta t_{sys} \approx \frac{1}{M} \sum_{\kappa=1}^{M} \Delta T_{\kappa} \, (d^{r}_{ref,\kappa} - d_{ref})/d_{ref} \, . \tag{7.0.9}$$

When conducting this experiment, the beam current should be kept as constant as possible throughout the irradiation time ΔT_{κ} for each fraction $\kappa = 1, \ldots, M$. Furthermore, the beam current must be set at a value to ensure that $\dot{D}_{ref,\kappa} \approx \dot{D}^{nom}_{ref}$ for all $\kappa = 1, \ldots, M$.

SABUS Technologies

SABUS crates

A South African standard for a communication bus for the Z80 microprocessors was developed during the late 1970's. iTL has simplified this SA-bus (SABUS) for in-house use.

The SABUS is used as a 8-bit communications interface between a master card and slave cards. The cards are mounted in a 19 inch crate, which is referred to as a SABUS crate, so that they are connected by DIN-41612 connectors to the backplane that provides the communication bus between the cards. The backplane also connects the cards to the low-voltage power-supply unit (PSU) of the SABUS crate. The SABUS crate and cards in the crate is referred to as a SABUS system.

The iTL implementation of the SA-bus has 8 data lines, 8 address lines, read and write control lines, and traditionally a reset line. The SABUS master card drives the address, data and control lines.

In all the new TCS equipment, the reset line will be used as a SABUS monitor line to indicate the functional status of all the cards in the SABUS crate – this is referred to as the SABUS status. The monitor line is normally at a logically high state, which signifies that the entire SABUS system is functional. A failure of the master card to address the watchdog-timer devices on any one of the slave cards, or a failure of any one of the SABUS cards, will cause a voltage drop on the SABUS monitor line, thereby changing it to a logically low state to indicate the SABUS failure status.

General features of the SABUS cards

Description

Each SABUS card typically uses a field-programmable gate array (FPGA) device or complex programmable logic device (CPLD) to achieve the required functionality. These devices contain reconfigurable logic components and interconnections. The FPGA devices are configured during the power startup of

the card by the configuration data (or "firmware") stored in a memory device specially provided for this purpose, whilst a CPLD is programmed before use.

Some of the SABUS cards may also implement a microcontroller that executes a program to provide additional features in conjunction to those provided by the re-configurable logic (CPLD or FPGA) device. Furthermore, some cards may provide for specialized input and output to external devices through connectors that are mounted on the front edge of the card. In these cases, the card will typically implement additional components, such as buffers, differential drivers, opto-isolators and/or relays to prove the required interfaces with the external devices. Figure 1 not only illustrates those features that are common to all SABUS cards, but also (in generic form) functional aspects that are specific to certain specialized cards.

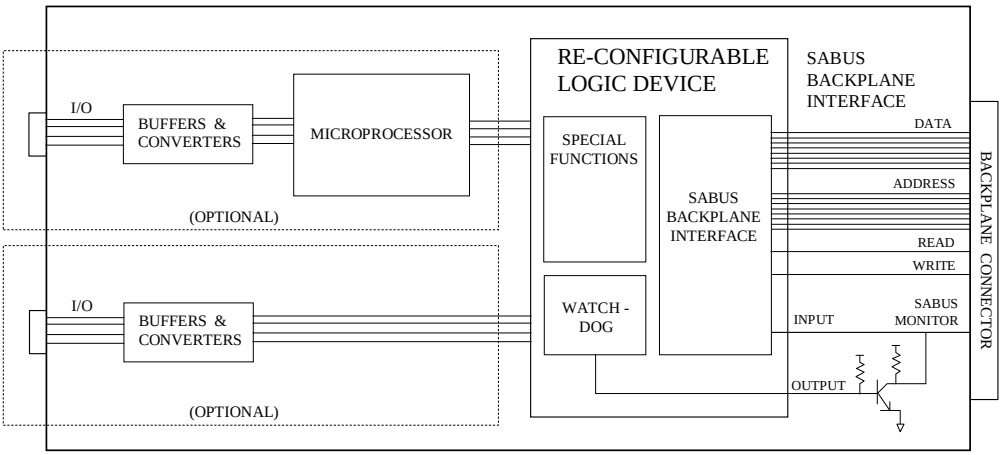

Figure 1: Functional diagram of a SABUS card indicating common as well as specialized features

Common functions and operation

The re-configurable logic device of a SABUS card provides the following set of functions that are common to all SABUS cards:

- An interface with the bus on the backplane of a SABUS crate, which is called the SABUS backplane interface.
- A backplane-status monitor that uses the input from the SABUS monitor line to continually observe the logical state of this line, thereby notifying the card when the SABUS monitor line changes to a logically low state (to indicate the SABUS failure status).

- A watchdog-timer device, which is coupled with a fail-safe, open collector transistor configuration to the SABUS monitor line. This transistor configuration will drop the voltage on the monitor line as soon as a timeout of the watchdog occurs, thereby changing the monitor line to a logically low state.

- A timeout will occur if the SABUS master card fails to regularly update the watchdog device of a slave card by writing a watchdog data value (via the SABUS backplane interface) to the correct address space in the slave card's re-configurable logical device.

SABUS-interface card

Description

The SABUS-interface card is a SABUS master card that allows an external unit, such as a personal computer (PC), to communicate with the cards in the SABUS crate. This interface between the external unit and the SABUS system is referred to as the SABUS communications interface.

The SABUS-inteface card is equipped with a differential communication bus (diff-bus) and an FPGA device. The diff-bus acts as the SABUS communication interface to the external unit. The FPGA device provides the required SABUS backplane interface and watchdog-timer device, as well as an interface between the diff-bus and the SABUS backplane (the diff-bus interface). The card also provides for one output in the form a relay contact-pair and one optically isolated input. The main features of the card are illustrated in Figure 2.

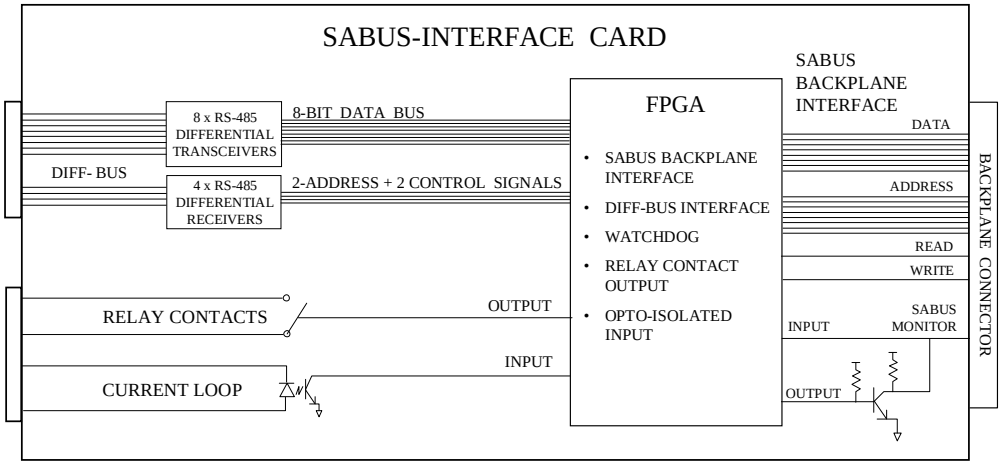

Figure 2: Functional diagram of the SABUS-interface card

The implementation of the diff-bus uses eight RS-485 differential transceivers to provide a 8-bit, bi-directional communication bus for the transferring of data, and four RS-485 receivers to provide two address lines, a strobe signal line and a directional control line to control the data flow (see Figure 2).

Due to the implementation of the differential drivers to establish the diff-bus, the SABUS-interface card is also referred to as the differential-driver card. We will refrain from this nomenclature since other SABUS cards also implement a diff-bus.

Functions and operation

- The SABUS-interface card provides the general functions as described in Section 7.
- The diff-bus, together with the diff-bus interface, acts as the SABUS communication interface that allows an external unit to communicate with the SABUS-interface card, and hence with all the other cards in the same SABUS crate. The diff-bus is a 8-bit, bi-directional cummunication bus with two address lines and two control lines.
- The external unit is responsible for keeping the watchdog-timer device from timing out. This must be done by regularly writing a special watchdog data value to the required address space in the card's FPGA device. The success of this write operation can be confirmed by reading back the value of the last written value. This read-back value is given as the last written value incremented by 1. The status of the watchdog can also be read back to check whether a timeout of this device has occurred.
- The relay contact is opened when the SABUS monitor line is pulled down, or when the watchdog fails. It may therefore be used to transmit the SABUS status as an output signal to an external device.
- The optical-isolator may be connected in series with an external current loop. This will then generate an input signal that indicates whether a current is flowing through the loop. The FPGA may also be configured to pull down the SABUS monitor line when no detectable current is flowing in the loop.

Multipurpose PCI card

Description

The multipurpose PCI card enables an external computer to communicate with a SABUS system that is equipped with a master card that provides a SABUS communication interface via a 8-bit, bi-directional differential communication

bus (diff-bus). The PCI card may be used in any computer that is equipped with a 33 Mhz PCI bus.

The basic design of the multipurpose PCI card is displayed in Figure 3. The card consists of a base board on which a PCI-bus bridge is connected to a FPGA device. The PCI-bus bridge provides the interface between the PCI bus of the computer and the logic circuits of the FPGA (*the PCI interface*). The PCI-bus bridge is specified for a standard 32 bits, 33 MHz PCI-bus with a 3.3 V or 5.0 V signaling voltage. The base board is also equipped with an optically isolated input, a normally open (N/O) contact relay switch and a differential-driver output that are all connected to this FPGA.

The main FPGA on the base board is connected to a daughter board on which a second FPGA is mounted. The daughter board is also equipped with RS-485 differential transceivers to provide a differential communication bus (diff-bus) that allows the PCI card to be interfaced with the SABUS system. The diff-bus is a 8-bit, bi-directional communication bus equipped with two address lines and two control lines (similar to the diff-bus described in Section 7). The secondary FPGA on the daughter board provides the logic circuits that links the diff-bus to the main FPGA (the diff-bus interface).

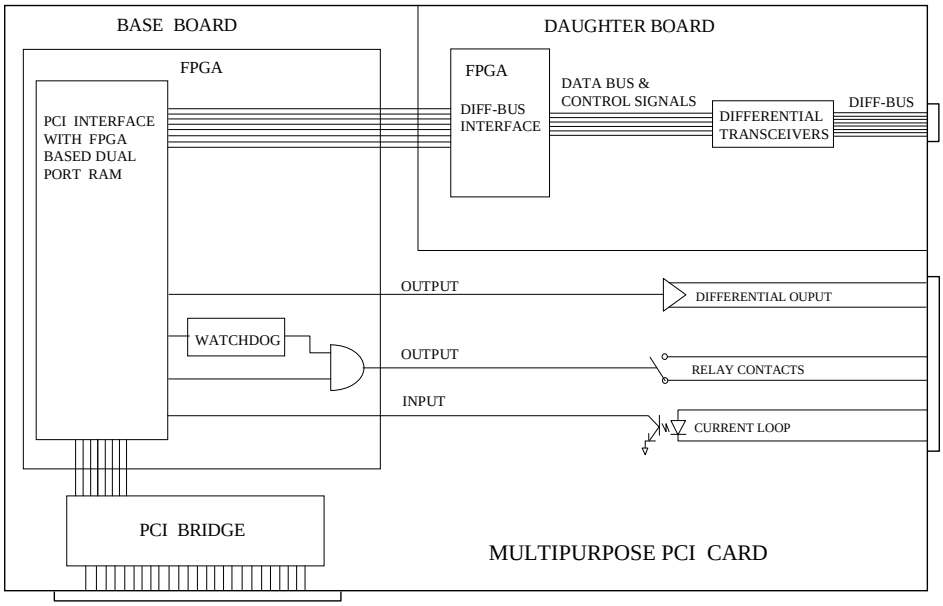

Figure 3: Functional diagram of the multipurpose PCI card

The input and output (I/O) of data between the PCI card and the PCI bus of the computer is illustrated in Figure 4.

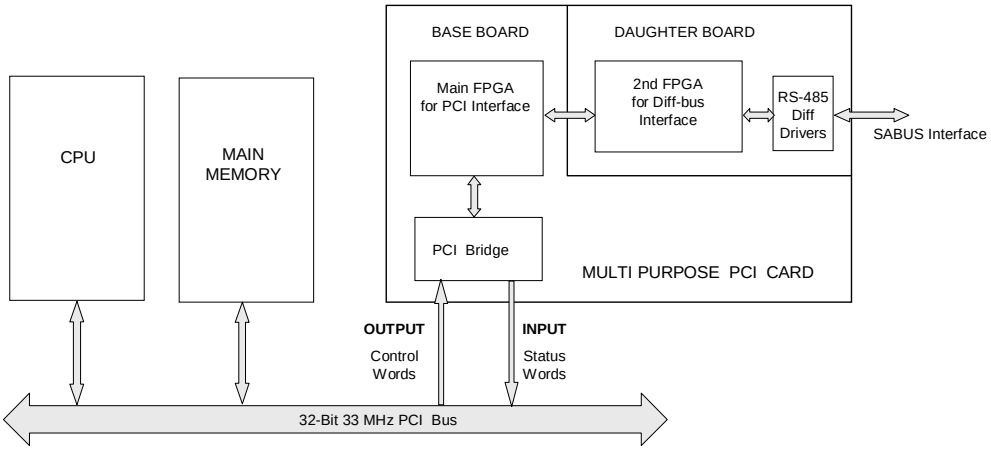

Figure 4: Communication between the PCI card and the computer

Functions and operation: Base board

- Data can be accessed asynchronously by both the PCI interface and the diff-bus interface (and hence the SABUS communications interface) with the use of a dual ported RAM on the main FPGA.
- The contacts of the relay switch can be opened and closed directly by instructions from the computer. A watchdog-timer unit on the main FPGA also controls the status of the relay contact. The relay is typically used as an emergency-stop switch.
- The watchdog-timer unit must be updated regularly at specified intervals by writing a watchdog data value to the correct address space in the main FPGA. Failing this, the watchdog will automatically open the relay contact. The watchdog data value must be the compliment of the previously written value. The success of the write operation can be confirmed by reading back the value of the last written value. This read-back value is given as the last written value incremented by 1. The status of the PCI watchdog can also be read back to check whether a timeout of this device has occurred.
- The timeout period for the watchdog unit is set at approximately 600 ms.
- The optical-isolator may be connected in series with an external current loop. This will then generate an input signal that indicates whether a current is flowing through the loop. The optically isolated input is typically used to obtain the status of the SABUS system without having to use the SABUS communication interface.

- The differential-driver output is controlled by the computer. It is typically used to provide a synchronization signal to an external device.

Daughter board

- The diff-bus interface controls the data flow in the SABUS interface (i.e. the connection between the SABUS system and the PCI system).
- Data are transmitted and received via the diff-bus. It uses eight RS-485 differential transceivers to provide the 8-bit, bi-directional data bus. The data flow is controlled by four RS-485 differential transmitters that drives two address lines, one strobe signal, and one direction control signal.
- All data transfers, when done on a continuous basis, are completed within 500 μs.

ETX computer module

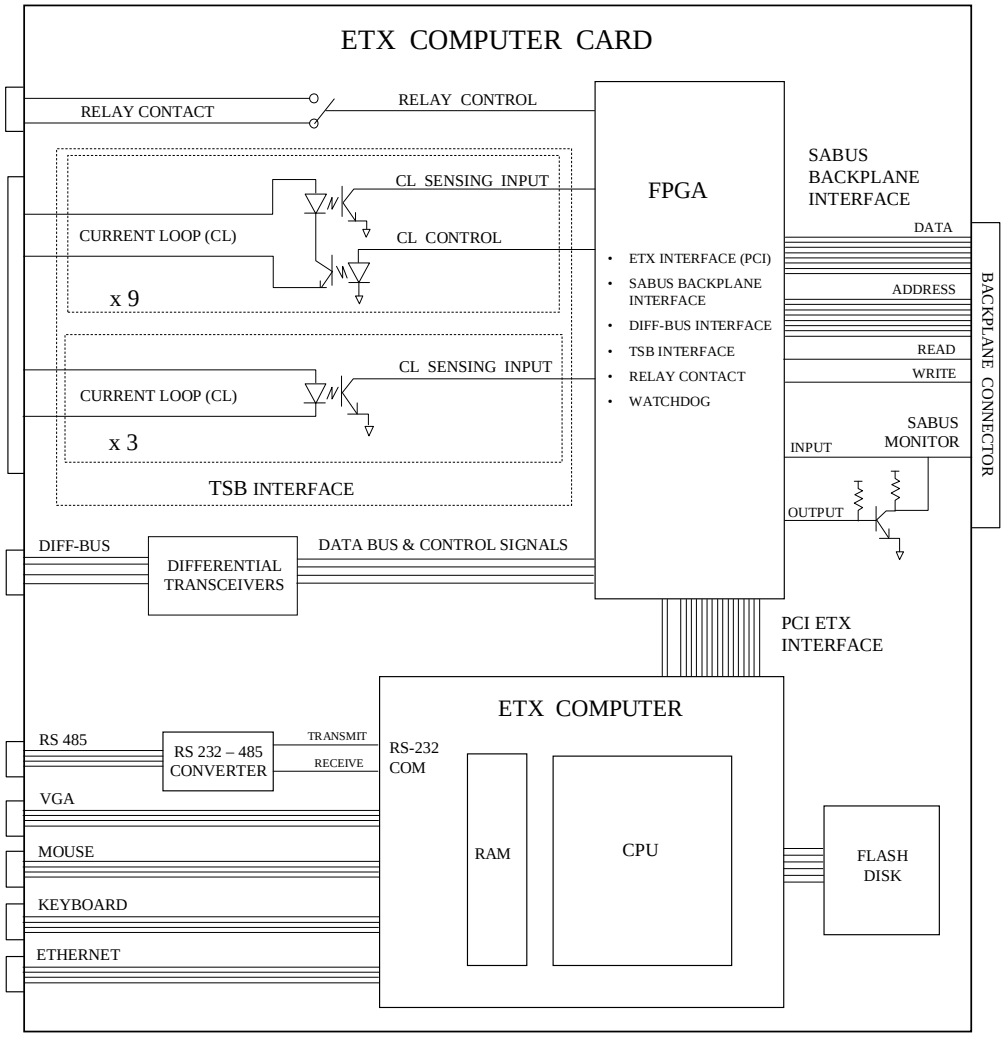

Figure 5: Functional diagram of the ETX computer module