DESIGN OF A TREATMENT CONTROL SYSTEM FOR A PROTON THERAPY FACILITY*

J. Katuin, J. Collins, C. Hagen, Wm. Manwaring, M. Wedekind & P. Zolnierczuk
Indiana University Cyclotron Facility, Bloomington, IN 47408, USA

Abstract

The IUCF Proton Therapy System (PTS) is designed by Indiana University and operated by the Midwest Proton Radiotherapy Institute (MPRI) to deliver proton radiation treatment to patients with solid tumors or other diseases susceptible to radiation. PTS contains three Treatment Systems, each consisting of four subsystems: Beam Delivery, Dose Delivery, Patient Positioning and Treatment Control. These systems are implemented using different operating systems, control software, and hardware platforms. Therefore, IUCF developed an XML network communication protocol so that subsystems could issue commands to and receive feedback and status from other subsystems over a local area network (LAN). This protocol was also applied to the MPRI clinical database used to access patient treatment plans. The treatment control system was designed so that a single user interface could be used to deliver proton therapy. The use of the XML and the LAN allowed the software of the treatment control system to be designed such that the various systems are treated as objects with properties and methods. This approach not only simplified the overall design of the treatment control system, it also simplified the effort required for software validation, testing, and documentation.

THE MPRI FACILITY

The design and status of the PTS is reported elsewhere [1]. A constant 208.4 MeV proton beam from the IUCF K220 cyclotrons is transported to a Beam Dump via a 57m Trunk Line, as shown in Fig.1, and diverted on demand to one of three Proton Therapy Treatment Rooms via energy selection beam lines, each containing an energy degrader. These lines transmit 65 to 208.4 MeV protons to the three Treatment Rooms.

Since MPRI clinical operation began in TR1 in February, 2003, IUCF has constructed a second treatment room (TR2), the subject of this paper, housing an IBA 360^{o} rotating Gantry system [2]. The Gantry incorporates an IUCF designed beam delivery nozzle containing a compact combined function magnet for beam scanning to deliver lateral beam distributions up to 30cm in diameter and an energy stacking method to achieve range modulations of up to 16 cm. A commercial six axis industrial robot, identical to the one used in TR1, has been adapted to the IBA Gantry system to position the patient for treatment.

CONTROL SYSTEM OUTLINE

From a physical perspective, PTS contains three Treatment Systems (TS1, TS2, TS3), each of which includes a Treatment Room, the beam line that feeds it, an external control area and the equipment contained in those areas. From a system perspective, each TS consists of a Treatment Room Control System (TRCS), a Beam Delivery System (BDS), a Dose Delivery System (DDS)

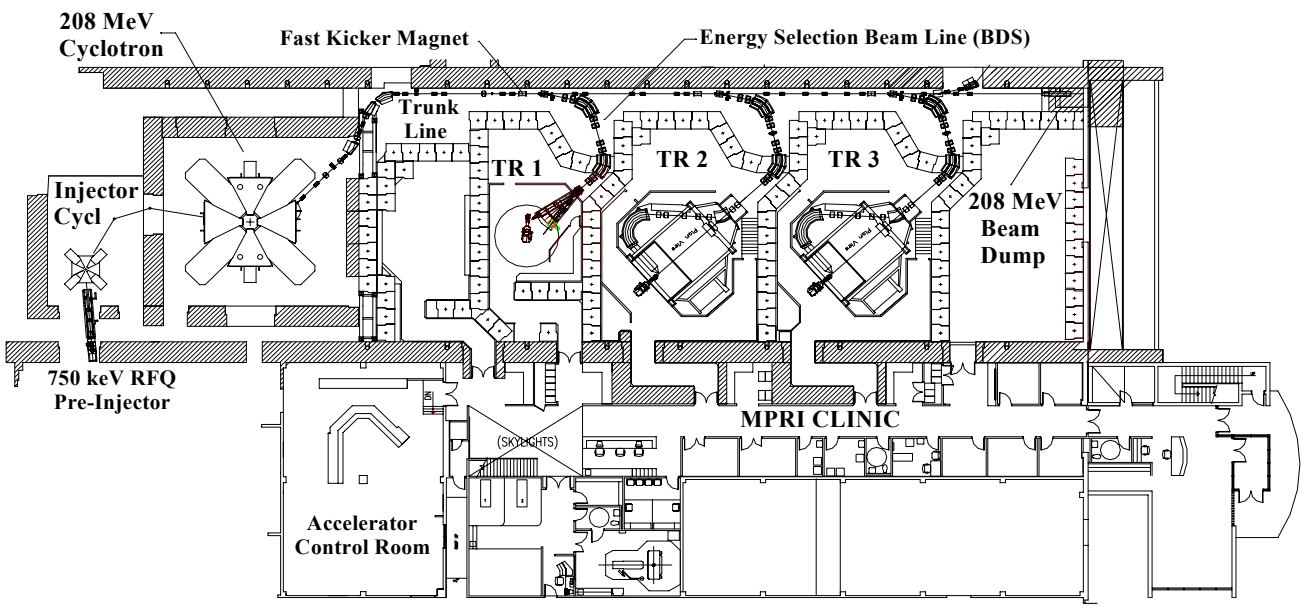

Figure 1. The MPRI facility showing the IUCF Cyclotrons, Trunk and ES lines, Treatment Rooms and Clinic.

* Support for this work is provided by the State of Indiana, Indiana University, Clarion Health, the DOE (Grant No. DE-FG-02000ER62966) and the NIH (Grant No. C06 RR17407-01)

and a Patient Positioning System (PPS). (There are also a few hardware-only subsystems that are not discussed here.) In order to deliver a treatment and record results of those treatments, each TS interfaces to the MPRI information system that manages treatment plans: the Treatment Planning Database (TPDB). TRCS is the subsystem that retrieves a Treatment Plan, presents it to the therapists and downloads parameters to the various subsystems. Each TS is a distributed computing system with connection supplied via TCP/IP over Ethernet. Within each TS there are four different operating systems and three implementation languages, a situation that arose partly from design, partly from history.

COMMUNICATIONS STRATEGY

To ensure reliability and accuracy in the transmission of treatment parameters, the communications both within each TS and between TS and TPDB were designed to use an XML protocol to define the messaging running under TCP/IP over an Ethernet LAN. Each TS subsystem uses XML library routines available to the various operating systems so as to:

1. Minimize programming time. Programmers call library routines to parse messages reliably in order to configure their system for treatment.
2. Minimize the software validation time by using standard parsing libraries.
3. Minimize programmer learning time. A messaging protocol is established so that communications between systems are discussed in terms of function calls. Therefore, if a system such as TRCS needs to "call" a PRIME function in BDS, then the high level design specifies a pseudo-function (see Figure 2 for an example) hiding the implementation of the XML message structure (see Figure 3).

The message protocol uses a client-server model, with TRCS being a client with respect to all other subsystems. Each TS has its own separate network leg, firewalled off from other TSs and from the outside world.

Schema checking is implemented to validate each message for structure and data type rules. Consequently, software alerts a user if incorrect information is received and moves to a "fail safe" condition. Schema checking is done using .NET facilities in PPS and home-brewed code in BDS, DDS and TRCS. All subsystems except PPS use the expat library [3] to parse messages. After parsing, the receiving subsystem tests that each parameter received in a message falls within a range acceptable for that parameter. Otherwise, the message is rejected.

Example - BDSPRIME (GANTRYANGLE)

TRCS sends the Gantry Angle to BDS and requests a final check of beam properties just before beam is delivered (Prime operation). The BDS checks the legality of the Gantry Angle and performs the Prime operation, verifing that measured beam Range, Energy Spread and Intensity meet the requested treatment values. It then returns these measured values to TRCS. Figure 4 shows the documentation the programmer uses. Figure 5 shows the actual XML message created.

Parameter	Format	Definition
GantryAngle	F5.1	Angle of Gantry in degrees (0-359.9 degrees) "???" indicates that the angle is unknown.

Figure 2. Definition of the BDS PRIME Command.

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes" ?>
<PTSMESSAGE DateTime="2006-05-04T19:59:06" Version="1.0">
<BDCS Source="TRCS" TS="2">
<BDSPRIME GantryAngle="90.0"/>
</BDCS></PTSMESSAGE>
```

Figure 3. Implementation of the BDS Prime Command in XML.

With schema checking, parameter range checking and the implicit TCP/IP error detection, it was judged to be a waste of time and bandwidth to implement any handshaking in the message protocol. However, each command message expects a prompt return message containing status and parameter values, perhaps followed by a delayed return indicating that a "slow" process has succeeded or failed. The protocol requires that there be a timeout applied to each prompt return. In the example above, the first Return message from BDS indicates that the message was received, successfully parsed and range-checked. A second Return indicates whether the Prime operation succeeded or failed and returns the measured values to TRCS.

DESIGN OF TRCS

The TRCS is designed to orchestrate the treatment process by sending parameters and commands to the other subsystems at appropriate times in the treatment process. Both BDS and DDS obtain all their instructions and data from TRCS, while PPS supports some user inputs. The TRCS GUI is designed as the primary interface between the medical user and the TS control system. TRCS is programmed in C++ and runs a Linux Operating System (OS) on a 2.8GHz Pentium 4, using NI cards [4] for I/O. TRCS allows a user to do the following:

1. Select treatment plans for a given patient.
2. Download parameters for a selected treatment field to TS subsystems.
3. Verify that all subsystesm are ready for treatment.
4. Be informed of the status of the other subsystems.
5. Check the validity of subsystem responses.
6. Start and stop treatments.
7. Send the results of a treatment to the TPDB.

BDS

BDS controls the beam line that transports beam from the trunk line, through the gantry and into the nozzle. BDS uses the same hardware and software as in the cyclotron control system. While BDS supports a GUI accessible by medical staff, the GUI is for display purposes only. It is programmed in C and runs the OpenVMS OS on an AlphaServer DS10, using VME as its field bus. BDS performs the following:

1. Setup all beam line devices according to the requested beam range (energy), including use of hysteresis loops to set magnetic elements.
2. Monitor all beam line devices for deviations from desired settings.
3. Perform the Prime operation that measures beam Range, Energy Spread and Intensity.
4. Monitor beam intensity, terminating beam delivery if intensity is too high.
5. Use steering loops to maintain constant beam position and angle at the nozzle entrance.

DDS

DDS controls devices on the nozzle (beam transport and detection hardware closest to the patient) and measures dose-related values. DDS implements the energy-stacking scheme used to obtain uniform depth-dose distributions. While DDS supports a GUI accessible by medical staff, the GUI is for display purposes only. DDS is programmed in C and runs the QNX OS. It runs on an 800MHz Pentium 4 and uses VME as its field bus. DDS performs the following:

1. Load, start and stop the wobbling magnet controller.
2. Prepare for energy-stacking at the specified beam range and range spread.
3. Manipulate the
4. Read, display and record data from the beam detectors in the nozzle.
5. Calculate and monitor beam properties (e.g., balance, flatness, symmetry), terminating beam delivery if any property exceeds predefined limits.

PPS

PPS performs patient positioning operations. PPS includes a robot [1] that moves the patient support device into the treatment position. Because patient motion must take the positions of many other movable devices into consideration, PPS includes a motion interlock system. PPS supports a GUI accessible to the medical staff which is actively used while positioning and removing patients. PPS is programmed in C#/.NET and runs the Windows OS on a 2.8GHz Pentium 4 using the same NI cards as TRCS. PPS allows a user to do the following:

1. Select, load and execute robot job files that specify the treatment position and path to follow.
2. Rotate the gantry to the desired angle.
3. Extend and retract Digital Radiography X-Ray detection panels.
4. Extend and retract panels that constitute the service floor within the gantry.
5. Extend and retract the snout within the nozzle (by communicating with DDS).

TESTING

Unit/Beta Testing (Bench Testing)

Both software and hardware were tested at the component level. Software testing involved identifying "units" which were exposed to a battery of test cases, of both "happy path" (inputs within expected limits) and "unhappy path" style. Test frameworks were either custom made (as with DDS and BDS) or used a package framework such as CPPUnit (TRCS) or NUnit (PPS), which provide library calls allowing a unit of software (such as a Class), to be tested for errors and successes.

Beta (integration) testing required the use of network and hardware emulators to mimic missing hardware and the other TS subsystems. These tests also use the happy /unhappy path philosophy, with emphasis on seeing that each subsystem fails in ways that do not endanger the patient or attending staff.

The network emulator developed for testing is called NePTUNE. Written in Perl, this emulator can be configured to act as a client or server and executes scripts that cause NePTUNE to act like a given subsystem, sending and receiving messages. Transmitted messages may be specified to be legal or have various illegal properties, testing the response of the target subsystem. Received messages are displayed and can be examined for proper construction and parameter values.

The final testing stage involves witnessed tests using written Test Procedures, often requiring between two and four shifts (8 hour units) to perform. For each subsystem, a software test is followed by a system test. In both cases, the subsystem is as isolated, with as many inputs emulated and outputs interpreted, as practical. Typically, only beam-related inputs could not be emulated. Also, typically, the software tests take significantly longer to perform than the system tests.

CONCLUSIONS

Using a distributed control system allowed us to leverage our experience in beam line control (BDS), use vendor supplied software packages (PPS) and apply a realtime OS where needed (DDS). Defining the messaging protocol allowed a clean separation of software development tasks.

Were we to redesign this control system, we would retain its distributed nature, but use fewer OSs and languages and redistribute responsibilities for device motion interlocking.

Finally, testing takes three longer than one thinks it will.

REFERENCES

[1] D.L. Friesel *et al,* WEPCH179, these proceedings.
[2] Ion Beam Associates, Chemin du Cyclotron, 3-1348 Louvain-la-Neuve, Belgium; www.iba.be.
[3] The Expat XML Parser, expat.sourceforge.net.
[4] National Instruments 6025 DAQ PCI Card.