A Web-Based Interface for the
Control of Model Trains

Graduate Project Report
Submitted to the
Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi

By
O. Michael Benavides

Spring 2001

Committee Members
R. Stephen Dannelly, Ph.D.
Chairman

Patrick Michaud, Ph.D.
Member

Holly Patterson-McNeill, Ph.D.
Member
Abstract

This project is the development of a control interface for the model railroad at the Computing and Mathematical Sciences’ Real-Time Computing Laboratory at Texas A&M University-Corpus Christi. The client application developed for this project allows users to control speed, direction, and path of multiple digital locomotives. The software functions via the Internet using a Perl/Common Gateway Interface (CGI) Client/Server scheme. The interface provides students the freedom of location to utilize the model train system in a round-robin fashion.
Table of Contents

Abstract........................................................................................................................................... i

Table of Contents ............................................................................................................................... ii

List of Figures ...................................................................................................................................... iii

Chapter 1: Project Overview
  1.1 Background ............................................................................................................................ 1
  1.2 Related Projects .......................................................................................................................... 3
  1.3 Overview of the System Operation ............................................................................................ 4
  1.4 This Document ........................................................................................................................... 6

Chapter 2: Environment and System Resources
  2.1 Real-Time Computing Laboratory ............................................................................................ 7
  2.2 Railroad Simulator ...................................................................................................................... 8
  2.3 Railroad Hardware ...................................................................................................................... 9

Chapter 3: Construction of The Control Interface
  3.1 Chronology .............................................................................................................................. 12
  3.2 Overview of System Components ............................................................................................ 14
    3.2.1 Access ............................................................................................................................... 14
    3.2.2 Processing User Request ................................................................................................. 15

Chapter 4: Control Interface Operation
  4.1 Login Procedure ....................................................................................................................... 18
  4.2 Locomotive Control .................................................................................................................. 21

Chapter 5: Control Interface Modules
  5.1 Module List .............................................................................................................................. 25
  5.2 Testing ..................................................................................................................................... 30

Chapter 6: Results and Future Work
  6.1 Results ..................................................................................................................................... 31
  6.2 Suggestions for Future Work ................................................................................................. 31

References ......................................................................................................................................... 32

Appendices

Appendix A: Source Code .................................................................................................................. 33
Appendix B: Simulator Documentation ............................................................................................. 34
List of Figures

Figure 1. Real-Time Computing Laboratory at Texas A&M-Corpus Christi .......... 7
Figure 2. CAMS Railroad Layout ...................................................................... 11
Figure 3. Relationship between Software Components ................................. 16
Figure 4. Unofficial Home Page for CAMS Digital Model Railroad .............. 18
Figure 5. Network Password Display .............................................................. 19
Figure 6. Login Queue Page .............................................................................. 19
Figure 7. Control Panel ..................................................................................... 20
Figure 8. Control Panel View Settings ............................................................. 21
Figure 9. Locomotive Speed Portion of the Control Panel ............................. 22
Figure 10. Switch Portion of the Control Panel .............................................. 24
Figure 11. Context Level DFD ......................................................................... 25
Chapter 1 Project Overview

1.1 Background

Texas A&M University-Corpus Christi is a public university with approximately 6800 students of which 250 undergraduates and 80 graduates belong to the Computer Science portion of the Department of Computing and Mathematical Sciences (CAMS). This program follows an applied-oriented curriculum, resulting in students with an ability to solve real-world problems, a quality desired among future employers.

The Department has constructed a Real-Time Computing Laboratory (train lab) of which one focus is to manipulate a digital model railroad. The system currently supplements the Systems Programming class (COSC 4348), a three-hour senior-level course focusing on the design and implementation of system software. Incorporating the model railroad system into this course allows students to extend theory into reality. Systems Programming assignments include writing software libraries for the system. Upon completion of a sufficient library, other courses such as Artificial Intelligence, Operating Systems, and graduate-level Mathematical Modeling will be able to take advantage of and extend the software system for many years to come.

A model railroad component of the laboratory was selected for five reasons. First, discrete and continuous control problems can be modeled. Second, the availability of a digital model railroad allows students to work on a wide variety of projects. Third, equipment needed for the laboratory is readily available. Fourth, the cost effectiveness of such a laboratory is extremely competitive. Finally, the faculty involved believe that the model railroad will stimulate interest within the University and heighten public relations.
Since the laboratory is in its infancy all related projects have focused on building basic functionality. The initial objective of the graduate project described in this paper was to develop a World Wide Web-based control interface for the digital model railroad using library functions written by five students in Dr. Michelle Moore’s System Analysis and Design course in the fall of 2000. The students were unable to complete the libraries in a timely fashion, so Dr. Stephen Dannelly (chair of this project) instructed me to write the interface to connect to Dr. Patrick Michaud’s railroad simulator known as RRsim (Michaud, 2001). Dr. Dannelly stated the interface of the simulator server and the actual railroad control server would be identical. The only change in source code of a client would be the port number of the TCP/IP connection subroutine. Later it was decided that port number 2999 would be used for the simulator and port number 3999 used for the control server.

A control interface is needed for three reasons. First, a Web-based control interface is needed to provide locomotive access from outside the laboratory. Second, a control interface can provide a scheduling scheme to allow sharing of railroad resources. Third, the control interface is needed to provide a user-friendly interface.

Traditionally model railroad engineers use a device known as a throttle to operate locomotives. This throttle is attached to the train track and can only be operated from a few feet away by one engineer. The primary goal of this project was to design an interface that would allow engineers to operate the digital model railroad without having to be physically in the room.

In addition to providing remote access via the World Wide Web, the control interface provides a means of allowing multiple users to share hardware resources. Users
who login to the control interface are allowed 30 minutes to operate the railroad. Users are scheduled in a round-robin queue fashion.

The control interface has been written in Perl and operates as a Common Gateway Interface (CGI) application. Since the railroad simulator and railroad control server use TCP/IP sockets as a means of interaction, the Perl programming language was an ideal choice for its ease of use in socket communication as well as Web development.

1.2 Related Projects

Other graduate projects related to this interface project have been a monitoring project by Maruthi Dantu and a real-time image analysis project by Pamela Castro. The interface and monitoring projects work together in that users can change railroad settings and then watch their changes take effect using various views provided by the monitoring project. Dantu’s project enables users to view a digital representation of the railroad’s status (Dantu 2000). Castro has written library functions that find the location of all five model trains in a real-time image. An image is taken in specific time intervals from a Web camera. When the library function is invoked, it takes that image file and determines where the model trains are by looking for certain characteristics the model trains may have such as color or shape. It then returns to the caller all five trains’ locations and the percentages of certainty that what it found was indeed a model train (Castro, 2000). While Castro’s project is directed at providing a library function to programmers, Dantu’s project utilizes this and other data to provide real-time system monitoring.
1.3 Overview of the System Operation

This section briefly describes system operation. Details of the graphic user interface are discussed in Chapter 4 and details of the system implementation can be found in Chapter 5.

Throughout the remainder of this document the terms “control interface” and “control panel” are used to indicate similar but different things. The term “control interface” refers to the entire system of HTML pages, CGI scripts, data files, etc. The “control panel” is the HTML form subcomponent of the system that enables a user to control the railroad hardware.

The Real-Time Computing Laboratory Model Railroad home page, http://www.sci.tamucc.edu/~trainlab/monitor, contains links to the control interface and monitoring interfaces as well as links to view the digital model railroad and frequently asked questions. There are three major sections to the control interface: a login session, the control panel, and a communication component to the control server/simulator. The system is comprised of three computers, the user’s, the Web server (named penguin), and the control server (named conductor) located in the Real-Time Computing Laboratory.

When a user decides to access the control panel, he/she is presented with an Apache generated login screen. After the user enters a valid password, he/she is presented with a page where they are asked if they want to join the waiting queue. Users other than designated professors must request a username and password from Dr. Dannelly. Since only one person at a time can effectively control the railroad, a time interval of 30 minutes is given to each conductor. At the end of this time, the user must return to the login page to resubmit his/her name. Initially a logout button was proposed
so users could leave before their 30-minute time limit. It was thought that a user could logout and the waiting time of other users would be adjusted. After careful consideration the logout button was removed because some users might think of their conductor time as reserved, similar to a flight reservation. For example, if user A has 27 minutes remaining when user B and then user C enter the waiting queue then user C will be told that his/her control interval will begin in 57 minutes. If user A and B logout after 5 minutes of use, then user C’s 30 minute time interval will have expired well before he/she expected to start controlling the system.

Upon a successful login session, a user is presented with the railroad control panel, his/her place in the queue, and the time until his/her turn. The current operator has access to all five locomotives, but is able to view/change the settings of only one locomotive at a time. Should a user want to change the settings of another locomotive, he/she selects that locomotive using the radio buttons at the top of the page and presses the view button. The settings for that locomotive will appear below in a text box along with a confirmation message. Changes to the operation of the railroad take place only after the submit button has been pressed.

Users who are waiting to gain control of the railroad are also taken to this control panel. When one of these users submits a change, an error message is displayed. However, these users are allowed to press the view buttons and receive the most recent hardware settings.

Sets of Perl subroutines send the control settings to the control server via Transmission Control Protocol/Internet Protocol (TCP/IP). The control server, running on conductor in room CI346, receives these requests and sends the appropriate LocoNet
commands to the train hardware. The communication protocol to the control server and the simulator are identical. A description of the RRsim communication protocol (Michaud, 2001) can be found in Appendix B.

1.4 This Document

Chapter Two explains the hardware configuration on which this system is implemented. Chapter Three overviews the software components and presents the chronology of how the system was designed and implemented. Chapter Four explains the operation of the system’s interface from a user’s point of view. Chapter Five presents detailed design information of each of the software components. Chapter Six presents a brief summary and recommendations for future expansions of this system. The appendices include source code for the control interface and documentation for Dr. Michaud’s railroad simulator.
Chapter 2 Environment and System Resources

2.1 Real-Time Laboratory Computers

The Real-Time Computing Laboratory contains several computers. However for this project, two Pentium III PCs have been used for software development and testing. Each of these Linux-based computers are attached to the campus local-area network and hence to the Internet. This allows students to use the system from a wide variety of locations. A diagram of the Real-Time Computing Laboratory can be seen in Figure 1.

Figure 1. Real-Time Computing Laboratory at Texas A&M-Corpus Christi
One computer in CI346, named *conductor*, is attached to a Digitrax DCS100 Command Station signaling device that controls all components of the track system. A control server application, developed by Benavides and Dannelly, runs on *conductor* and issues appropriate LocoNet commands through the command station. Client systems can communicate with *conductor* over a TCP/IP connection. A direct connection to the control server program is made by opening a connection to *conductor.tamu.cc.edu* port number 3999.

The Web server, known as *penguin*, is a Linux-based computer located in room CI227 at Texas A&M University-Corpus Christi. This computer is running Perl version 5.6.0, Apache server software and Red Hat Linux release 7.0 (Guinness), Kernel 2.2.17-14smp on a 2-processor i686. *Penguin* is the home for Dr. Patrick Michaud’s railroad simulator located at *penguin.tamu.cc.edu* port 2999. There are plans to move the railroad simulator to *conductor.tamu.cc.edu* port 2999. The Perl/CGI scripts that generate the control panel are executed on the *penguin* Web server, but physical changes to the train system are controlled by *conductor* in the Real-Time Computing Laboratory.

### 2.2 Railroad Simulator

A simulator known as RRsimg was written by Dr. Michaud to support train lab activities. It has the same capabilities as the railroad control server program with the added benefit that expensive locomotives can not drive off the track or collide. RRsimg is a program designed to simulate a model railroad using the Simple Railroad Control Protocol (SRCP). RRsimg simulates a model railroad layout consisting of blocks of track, position and speed of locomotives, and turnout states. Client systems can communicate
with RRsim via SRCP over a TCP/IP connection. Multiple clients may communicate with a single instance of RRsim, thus emulating a real-world environment where there may be multiple controlling interfaces. Users can directly connect to the simulator by opening a telnet session to port 2999 on the host that is running RRsim (Michaud, 2001).

2.3 Railroad Hardware

Digitrax, Inc. manufactures the primary electronic components, switch and locomotive decoders used in the laboratory. The system was constructed by Bob Thiele and Dr. Dannelly under the direction of Charles Irby. The railroad has been constructed on half-inch paperboard that is fixed to a 30-inch high, 6’x 8’ plywood table. A photo album at http://www.sci.tamu.edu/trainlab/photoalbum/index.html presents the CAMS digital model railroad development.

Nearly one hundred feet of track was used to construct one main track and one sidetrack. There are 34 turnouts on the railroad, each of which can be set to a thrown or closed position. The thrown position forces a locomotive to change paths while the closed position forces a locomotive to continue on the same path. The control server machine is connected to the railroad via a command station that facilitates railroad control.

The entire railroad track was divided into 6-8 inch sections and each section was assigned an identification number using a scheme developed by Dr. Dannelly. The labeling scheme created for the railroad track is as follows. The innermost track leading to the locomotive turntable begins with 100. All sections of this track begin with a 1XX. All turnouts or switches are a multiple of 10. Therefore the section of track labeled 140 is on the innermost track and can switch to a thrown or closed state. As the
track proceeds outward, the block labels use numbers that begin with a 2XX and continue until they reach the outermost track which begins with 8XX. A diagram of the track numbering scheme, as modified by Dr. Michaud, is shown in Figure 2.
Chapter 3 Construction of The Control Interface

3.1 Chronology

The steps taken to implement this project and an overview of the project design are explained. The following steps were taken before the design and implementation of the Web-based control panel.

1. Consulted with Dr. Stephen Dannelly (Assistant Professor and project director for the Real-Time Laboratory) to determine which functions this control interface will supply.

2. Consulted with Charles Irby (Director of Computer Services at Texas A&M University-Corpus Christi), Michael Anderson, Jeff Jones, Greg Mabrito, Chris Wells, Kase Wright (writers of the control library, also known as Group 4), Maruthi Dantu, and Pamela Castro (railroad monitoring project members) to gain a better understanding of what the railroad looks like, how the locomotives function, and how the computer communicates with the railroad and locomotives.

3. Consulted with Dr. Patrick Michaud (Perl/CGI expert) to discuss the feasibility of constructing this project using Perl/CGI.

4. Consulted with Dr. Stephen Dannelly, to determine who will be able to login to the control panel and how access will be controlled.

5. Consulted with Dr. Patrick Michaud to discuss restrictions of access.

6. Consulted with Dr. Patrick Michaud to discuss Perl sockets and whether it was best to fork the server.
7. Dr. Stephen Dannelly recommended the project connect to the simulator as the C libraries to control the railroad had not been completed. He stated the simulator and control server should share similar attributes and communication protocols with the exception of the location and port settings.

8. Met with Pamela Castro and Maruthi Dantu to gain a better understanding of what their projects were about and how they worked.

9. Researched various digital model railroad Web sites.

10. Began learning Perl and CGI programming by writing a series of test programs.

11. Studied RRsim by creating a series of test programs.

Once the system’s design, over viewed in section 3.2, was complete, implementation of the various components was begun. The following steps were taken to implement the system:

1. Wrote scripts to create a graphic user interface (GUI) and requested feedback from Dr. Patterson-McNeill.

2. Wrote a basic CGI program to communicate with the railroad simulator.

3. Created and tested the major subroutines.

4. Tested wait and conductor queues with simultaneous logins.

5. I used the HCI technique known as Cognitive Walkthroughs, to test the system's usability.
3.2 Overview of System Components

Maruthi Dantu, Pamela Castro, and I have developed the home page for the control and monitoring of the digital model railroad. It has been developed using the hypertext markup language (HTML) and resides on the Web server (*penguin*) at http://www.sci.tamu.edu/trainlab/monitor. The home page includes many links. One link allows users to proceed to a monitoring page that allows them to view the railroad as a digital image and/or streaming video. Other links provide history and answers to frequently asked questions. The link to the control panel takes a user to a login page before proceeding to the control panel.

3.2.1 Access

The login page and each of the other CGI scripts in this project make use of the HTTP authentication protocols available through the Apache Web server (Apache, 2000). The Apache Web server software authenticates users according to a password file maintained by Dr. Dannelly and supplies the authenticated username to CGI scripts via the *REMOTE_USER* environment variable. Each control interface CGI script checks this environment variable and a locally available "queue file" named `~trainlab/CI_curConductor.txt` to determine if a user is currently allowed to access the train system. If access is not allowed because the system is busy, the user is then added to the end of the waiting queue. To make use of the HTTP authentication protocols a `.htaccess` file was created with the following text in the `~trainlab/cgi-bin` directory.
Then a blank .htpasswd file was created in the cgi-bin directory as well. If the .htpasswd file is created in an unprotected directory, a malicious user could view the .htpasswd file and hack his/her way into a protected directory. Upon creation of the .htpasswd file the command htpasswd .htpasswd username can be executed to add users to the .htpasswd file. To combine these steps into one command, the train lab account administrator can go to the directory ~trainlab/cgi-bin/ specified in AuthUserFile and use the htpasswd program with the -c switch to create .htpasswd. The command htpasswd -c .htpasswd username creates the file and adds "username" as the first user. The program prompts for a password, then verifies by asking again. The password will not be visible when typed. To add more users in the future, the administrator can use the same command without the -c switch. To delete users, the administrator opens the .htpasswd file in a text editor and deletes the appropriate lines.

### 3.2.2 Processing User Requests

Figure 3 illustrates the relationships among the major software modules.

Processing users’ requests is a four-step process.

1. The user submits locomotive and/or switch settings via the control panel.

2. The settings are sent from the user’s machine to the submit_Action subroutine in CI_panel.cgi which first checks if that user is the current conductor, then
manipulates the data into a string that can be read by the simulator or control server running on conductor.

3. The request is sent from penguin to the simulator or control server on conductor via a TCP/IP socket connection. A message similar to -200 [131265.52] RRsim/0.5 ready is sent by the control server or simulator back to the control client. Should the control client not receive the appropriate message from the control server, a message to the user will appear that the server is down.

Figure 3. Relationship between Software Components
4. If the message cycle in step 3 is successful then transmission of user requests from the CGI program to the server or simulator begins. These messages, as in step 3, are in Simple Railroad Command Protocol.

As an example of these four processing steps, the following actions are taken when a user changes locomotive one’s speed to 70:

1. The Web browser sends form data to control panel.
2. Control panel verifies user’s request
3. Control panel attempts to connect to the control server. Control server sends a -200 [0.00] RRsim/0.5 ready response.
4. Control panel responds with the message
   setloco 1 speed=70 isrev=0.

The server receives a message, if correct the server responds with a -200 [0.00] OK indicating that the message sent was valid.
4.1 Login Procedure

This chapter discusses in detail the inner workings of the control interface and shows a typical control interface session. The first page a user often sees is the home page for the Real-Time Computing Laboratory at Texas A&M University-Corpus Christi at \url{http://www.sci.tamucc.edu/trainlab/monitor/} (Figure 4). To become a conductor he/she needs to click on the link “Control the Railroad” or “Control the Simulator”. The user is then asked to enter his/her password (Figure 5). Upon successfully submitting a valid user name and password, the future conductor is sent to the login queue page (Figure 6).

![Figure 4. Unofficial Home Page for CAMS Digital Model Railroad](image-url)
Figure 5. Network Password Display

Figure 6. Login Queue Page
The login queue page script first checks the conductor queue and waiting queue to removes users whose time has expired. Should he/she decide to become a conductor by pressing “yes”, the user is redirected to the control panel (Figure 7). At the control panel the user is presented with a variety of input fields. If the user is not the current conductor then he/she can view the most recent settings, but can not submit new settings. A user becomes a conductor when the previous conductor’s time has expired.

![CAMS Control Interface](image)

Figure 7. Control Panel
4.2 Locomotive Control

Each of the five locomotive selection radio buttons on the control panel corresponds to a locomotive. When a locomotive has been selected and its view button is pressed all current settings for that locomotive are made visible to the user. For example, suppose the user wants to check the speed of locomotive 1. After selecting the appropriate radio button and then pressing the view button, the default message “Speed must be an int –100 thru 100” will change to “Train One's current speed is –70. Motion is Reverse”. The speed text box displays the current speed (Figure 8). This is done using a refresh method that retrieves values from ~trainlab/trainX.txt and ~trainlab/turns.txt files containing railroad and locomotive settings.

Figure 8. Control Panel View Settings
The speed of the selected locomotive is controlled using a text box that contains integers ranging from -100 to 100. The integers refer to a percentage of locomotive power. The user is able to type the desired power and submit it for processing (Figure 9). When the submit button is pressed the locomotive number and its speed are sent to a Perl/CGI subroutine called set_Speed where the parameters are verified, manipulated and sent to the control server or simulator. Invalid values result in an error message and no change is made. The Stop_All button also invokes the set_Speed subroutine and sets all speeds to zero.

Figure 9. Locomotive Speed Portion of the Control Panel
Each turnout has two radio buttons (Figure 10). A text message displays label and button information. The button closest to its label sets the turnout to the thrown position. As seen in Figure 10, turns 22, 24, 37, 43, 71, and 73 have been set to a thrown position. To find the proper block multiply the turnout by 10 and this number will correspond to its label on the railroad track. For instance if a user wanted to change the turnout at block 110 he/she would change turnout 11. A labeled picture of the track is made available at the bottom of the control panel so the user does not have to change from the control and monitor pages constantly. The thirty-four turnouts or “switches” represented on the control panel are controlled using the `~trainlab/cgi-bin/CI_panel.cgi set_Turns` subroutine which takes advantage of the simulator’s `setswitch` command.
The button closest to its number sets the switch to an thrown position. For example the switch below is set to an thrown position.

80  O  Rown  O  closed

10  O  11  O  12  O  14  O  20  O  21  O  22  O  24  O

25  O  27  O  28  O  30  O  31  O  33  O  34  O  35  O

36  O  37  O  38  O  41  O  43  O  44  O  45  O  47  O

52  O  53  O  62  O  63  O  71  O  72  O  79  O  81  O

82  O  83  O

Figure 10. Switch Portion of the Control Panel
Chapter 5 Control Interface Modules

5.1 Module List

There are 6 important modules and approximately 800 lines of code that make the control interface possible. A context level data flow diagram can be found in Figure 11.

![Context Level DFD Diagram]

Figure 11. Context Level DFD

For each module a description of the type of language used, what the module should expect as input, what it accomplishes and what is expected as output is listed below.
Control Panel

Name: CI_panel.cgi

Type: CGI application using CGI.pm, Perl, and HTML

Input: Locomotive selection, speed settings, and/or switch settings.

Function: This processes the HTML form sent from the user’s Web browser.

Output: Form data is sent back to the user’s Web browser, data is posted to CGI subroutines and SRCP commands are sent to the control server/simulator.

Pseudo code:

code
get time
open data files
if (conductor time < 30)
   keep conductor
else
   get new conductor
if (conductor)
   display minutes remaining
else
   display user minutes until session
if (submit turns and valid conductor)
   set_Turns
   get_turns
else
   get_turns
if (submit stopAll and valid conductor)
   set locos to zero
   submit_Action
validate_Speed
if (submit speed and valid speed and valid conductor)
   set_Speed
   create messages
   connect_to_Server
else
   view_Settings
print forms
**Subroutine: view_Settings**

Type: Perl CGI Subroutine

Input: This subroutine reads the assignment number $trainX$ of the current radio button from the control panel HTML form data.

Function: Get current settings from settings file $~\text{trainlab/trainX.txt}$ for the selected train.

Output: Displays to the browser the control panel with current information such as speed, direction, turnout positions, and other optional settings.

Pseudo code:

get locomotive number from form
open train.txt or die
get exclusive lock or die
while (read file){
    using split
    assign to var train
    assign to var speed
}
close(train.txt)

**Subroutine: set_Speed**

Type: Perl CGI Subroutine

Input: This subroutine retrieves two integers from the Control Panel. One from speed text box and the other from a locomotive select radio button value.

Function: This subroutine places its parameters in a message $\text{setloco LOCO [speed=PCT] [isrev=X][fwd][rev]}$ which is sent to $\text{connect_to_Server}$ for processing.
Output: The designated locomotive moves about the railroad according to a set speed after its decoder receives input. The Perl/CGI application sends to the browser the control panel with current information such as speed, direction, turnout positions, and other optional settings.

Pseudo code:

```perl
get locomotive from form
open train.txt or die
get exclusive lock or die
print number and speed to locomotive file
close(train.txt)
```

**Subroutine: set_Turns**

**Type:** Perl CGI subroutine

**Input:** This subroutine retrieves two integers which were extracted from the Control Panel form data. One being the turnout number and the other an integer indicating thrown or closed.

**Function:** This subroutine creates a message to be passed to `connect_to_Server`. It takes advantage of the simulator’s `setswitch` function. The CGI script relays this request to the control server or simulator for processing.

**Output:** The designated switch will direct locomotives to turn right or straight at the designated turnout. The Perl/CGI application `~trainlab/cgi-bin/CI_panel.cgi` returns to the browser the control panel with current information such as speed, direction, turnout positions, and other optional settings.
Pseudo code:

get turn settings from form
open turns.txt or die
get exclusive lock or die
print settings to turns.txt

create set switch message for each turn

connect_to_Server
close connection

Subroutine: connect_to_Server

Type: Perl CGI subroutine

Input: Request passed from Perl subroutine.

Function: Relays data from several Perl subroutines to the control server.

Output: Requests sent via TCP/IP to control server or simulator.

Pseudo code:

make socket connection or die
receive message from server
send message from subroutines to server
close connection

Control Server/Simulator

(Control server developed by Dr. Dannelly and Benavides, simulator by Dr. Michaud)

Type: C Program

Input: SRCP Requests via TCP/IP from Control Client.

Function: Calls appropriate library function to operate the command station and sends a confirmation message to control client.

Output: Confirmation message to control client.
5.2 Testing

Several students and I tested the system over a two-month period. It was tested for its load capability, invalid settings, switch testing, invalid conductors, transfer of control, and ease of use. Load testing of these functions was done by a group of 5 users who synchronized simultaneous logins to determine how the queuing system was affected. Since the login page can be accessed by more than one user at a time, testing proper functioning is a must. The outcome resulted in the system placing one user in the conductor queue and all others in the waiting queue.

Other testing included entering invalid speed settings. Integers less than –100, greater than 100, and non-numerical values were entered as speed settings. The system responded with an error message to the conductor stating an integer must be entered. The switches were also frequently and quickly changed to determine how the system would react. The switches properly responded to the tests.

The transition from being a waiting user to being the user designated as the conductor was also tested. To test for transition effectiveness a single user was asked to log in with 0 users waiting in the user queue and no conductor. This process was repeated for 0 – 4 users waiting in the user queue and 1 conductor. The testing process accounted for expired conductor and user times as well as valid times. Four students were asked to use the system in order to rate the ease of use. The students were asked to perform a task without instructions. All four students felt comfortable performing tasks on the system within 15 minutes.
Chapter 6 Results and Future Work

6.1 Results

The desired result of this project was to have a graphical user interface that would control the digital model railroad located in the Real-Time Computing Laboratory at Texas A&M University-Corpus Christi. An interface that handles multiple users was developed. The system operates efficiently and incorporates appropriate security measures. It has been designed to be easily modified and extended by future projects.

6.2 Suggestions for Future Work

Future projects may improve upon this software by adding the capability of multi-user simultaneous control -- perhaps one locomotive per user. Future projects may add the crane and turntable to the control panel once they become operational. Future projects may also improve upon the current scheduling scheme in which professors and students can request a time and duration of a logon session. There should be some sort of locomotive collision control to protect the University’s investment. Also, a function to cause the locomotives to move to and stop at a designated area of the railroad is essential for replicating experiments.
References


Castro, Pamela C. *Re: Project.* E-mail to O. Michael Benavides. 4 Nov. 2000.


Dantu, Maruthi. *Re: Project.* E-mail to O. Michael Benavides. 4 Nov. 2000.


Appendix A: Source Code

On compact disc.
Appendix B: CAMS RR Simulator (RRsim) Related Documentation

Note: This is the entire documentation for the RRsim. Below is the documentation from Dr. Patrick Michaud’s RRsim. All commands of the RRsim are presented below. To read the original document in its entirety please go to http://www.sci.tamucc.edu/~pmichaud/camsrr/rrsim.txt.

1. Overview and example session

RRsim is a program designed to simulate a model railroad with an associated Simple Railroad Control Protocol (SRCP) server. RRsim simulates a model railroad layout consisting of blocks of track, position and speed of locomotives, and turnout states. Client systems can communicate with RRsim via SRCP over a TCP/IP connection. Multiple clients may communicate with a single instance of RRsim, thus emulating a real-world environment where there may be multiple controlling interfaces.

To start the simulator, simply enter the command "rrsim". By default, RRsim will initialize a default model railroad layout and begin listening for TCP/IP connections on port 2999. Users can directly connect by opening a telnet session to port 2999 on the host that is running RRsim. A different port can be chosen upon startup by adding "port=NNNN" on the command line when starting RRsim (e.g., "rrsim port=3015").

The remainder of this section describes a sample RRsim session through the command-line interface. When a client connects to RRsim (e.g., "telnet penguin.tamucc.edu 2999"),
RRsim sends SRCP responses to indicate the current state of the simulation. SRCP responses look similar to the following:

+200 [0.00] RRsim/0.2 ready
+200 [0.00] addloco OK
-200 [0.00] OK: layout=camsrr.txt (51 blks, 1 locos) map=camsrr.map

The first response indicates the simulator is running version 0.2 of RRsim, and the last response indicates the simulator has been initialized with a track layout of 51 blocks and 1 locomotive. Section 2.1 below describes the SRCP messages supported by RRsim.

The "start" command causes RRsim to enter real-time mode, where the locomotives in the simulation are moved according to wall-clock time:

start
-200 [0.00] start OK, simulation speed 1

The "showloco" and "showblock busy" commands will let you see the state of any locomotives in the simulation and which blocks have a locomotive on them:

showloco
-221 [24.00] loco 1 speed=0.00 isrev=0

showblock busy
-223 [34.00] block 101 isbusy=1

RRsim's response to the showloco command above indicates that as of simulation time (simtime) 24.00, locomotive 1 had a throttle setting of 0.00--i.e., it is stopped. The response to the
showblock command shows that as of simtime 34.00, block 101 is busy (i.e., it has a locomotive on it). From this we would probably infer that locomotive 1 is stationary in block 101.

Note: You may be wondering why the locomotive's current block is not directly returned as part of its status. The reason for this is that in DCC model railroading a locomotive doesn't have any way of determining its location on the track--it only knows its speed and direction. A locomotive's position has to be determined by other means such as inference from occupied blocks or some sort of external detection system. Future versions of RRsim will model these detectors as they are developed.

Having a stationary locomotive is not very interesting, so let's start locomotive 1 moving using the "setloco" command:

```
setloco 1 speed=100 fwd
-200 [42.00] setloco OK
```

To see the locomotive moving, we can use the "showblock busy" command to watch the locomotive move from block to block:

```
showblock busy
-223 [49.00] block 102 isbusy=1
showblock busy
-223 [55.00] block 103 isbusy=1
```

Thus at simtime 49.00 the locomotive was in block 102, and at simtime 55.00 the locomotive was in block 103.
Of course, the point of having a simulator is so that we can see what is going on in the simulation as we're completing it. RRsim allows the current simulation status to be displayed in real-time through its status interface. To see this, open another terminal window, connect to the simulator ("telnet penguin 2999"), and enter the command

```
status
```

This command requests RRsim to display detailed status information on the terminal output. The status information typically includes the current simtime, an ASCII map of the track layout with digits indicating the current positions of any locomotives, and the full status of each locomotive. The status display will be updated after each RRsim command (to see this, hit enter repeatedly on the command line).

Note: Client programs should not rely on the output of the status command. The output produced by the status command is implementation (and possibly layout) dependent. Furthermore, many of the details provided by RRsim status display will only be available from a simulation, and not a real model railroad layout.

You can have the status display automatically redisplay itself by adding a refresh attribute to the status command, as in:

```
status refresh=2
```

This tells RRsim to automatically send a status display at least once every two seconds. Note that the display may actually refresh more often than this, depending on other commands being sent to RRsim.
The remaining simulation commands are described in section 2.2 below. To end your connection to RRsim, simply enter the command "exit". To completely shut down the RRsim process, send it a KILL or HUP signal (you must be the owner or superuser to do this).

2. RRsim and SRCP communications protocol

Commands in RRsim consist of SRCP commands, as well as a number of simulation-specific commands. Users or user programs typically interact with RRsim by sending line-oriented commands ("SRCP requests") to RRsim and observing the responses. Requests are sent as lines of ASCII text described in section 2.2 below. After acting on a request, RRsim sends back one or more responses indicating the results of the request and/or the current state of the simulation.

2.1 Server responses

All SRCP responses have the same format:

    +nnn [sss.ss] text

The first character of each response is either a "+" or a "-". "+" indicates that more responses follow, while "-" indicates the end of responses from the server until another command is sent. The three-digit "nnn" value provides a response code indicating the type of response. "[sss.ss]" gives the current simulation time in seconds, and "text" contains either response-specific information or a human-readable message.

Similar to many popular network protocols, the first digit of the response code indicates the type of response:

1xx  Information not directly related to the request

2xx   Request handled successfully
3xx   Follow up request required
4xx   Error in request
5xx   Error in simulator

At this stage of SRCP, the last two digits do not have any specific meanings other than to identify specific responses.

In the current version of SRCP there is no maximum number of responses that may be returned by the server, nor is there a maximum length for individual responses. The following sections describe SRCP responses which follow a specific format.

2.1.1  221 locomotive status

SYNOPSIS

-221 [nn.nn] loco LOCO speed=PCT isrev=X

DESCRIPTION

The 221 response message is used to return information about a locomotive. LOCO is the locomotive identifier, speed gives the throttle percentage as a floating point number from 0 to 100, and isrev is "0" or "1" to indicate forward or reverse motion.

2.1.2  222 switch status

SYNOPSIS

-222 [nn.nn] switch NUM isthrown=X

DESCRIPTION
The 222 response message is used to return information about a turnout block (switch). NUM is the switch identifier, isthrown is "0" for a closed switch and "1" for a thrown switch.

2.1.2 223 block status

SYNOPSIS

-223 [nn.nn] block NUM isbusy=X

DESCRIPTION

The 223 response message is used to return information about a block. NUM is the block identifier, isbusy is "0" if the block is clear and "1" if there is at least one locomotive on the block.

2.2 SRCP requests and RRsim commands

2.2.1 null command (NOP)

SYNOPSIS

# [comment] or [blank line]

DESCRIPTION

A command beginning with a '#' or consisting entirely of whitespace is a null operation and does not make any changes to the simulation. RRsim returns a 200 response.

RESPONSES

200 OK

2.2.2 reset

SYNOPSIS
reset

DESCRIPTION

Resets the simulation to the default layout and a simulation time of 0.0.

RESPONSES

200  Layout loaded successfully
400  Cannot open requested LAYOUT
500  Insufficient memory to load layout.

2.2.3  setloco

SYNOPSIS

setloco LOCO [speed=PCT] [isrev=X] [fwd] [rev]

DESCRIPTION

Sets the attributes of the locomotive given by LOCO. The speed parameter controls the throttle and is a value from 0 to 100. The isrev parameter is set to non-zero to put the locomotive in reverse; "fwd" and "rev" are synonyms for "isrev=0" and "isrev=1" respectively.

RESPONSES

221  OK, updated locomotive status
400  No such locomotive

2.2.4  setswitch

SYNOPSIS

setswitch NUM [isthrown=X] [throw] [close]

DESCRIPTION
Throws or closes the switch given by NUM. Setting the isthrown parameter to non-zero throws the turnout; "throw" and "close" are synonyms for "isthrown=1" and "isthrown=0".

RESPONSES

222  OK, updated switch status
400  No such switch

2.2.5  start

SYNOPSIS

start [rate=RATE]

DESCRIPTION

Places RRsim in "real-time" simulation mode. The locomotives are moved according to their parameters while RRsim is waiting for commands. The rate parameter can be used to speed up or slow down the simulation relative to the wall clock time; a rate value of 2.0 causes the simulation to progress twice as fast as the wall clock. The default value for rate is 1.0. Setting the rate to zero stops the real-time simulation, as does executing the "step" command below.

RESPONSES

200  OK

2.2.6  step

SYNOPSIS

step [delta=DELTA]

DESCRIPTION

Single-steps the simulation forward DELTA seconds.
If real-time simulation mode was enabled (see the "start" command above), then it is halted prior to performing the step. If the delta parameter is omitted, Rrsim steps forward the same amount given by the previous step command.

RESPONSES

200 OK

2.2.7 showloco

SYNOPSIS

showloco [LOCO ...] [all]

DESCRIPTION

Returns the status of the locomotives specified as arguments. If no locomotives are specified as part of the command, returns the status of all locomotives. Each locomotive's status is returned via a "221 locomotive status" response (section 2.1.1).

RESPONSES

200 OK (if no locomotives available)

221 locomotive status

400 No such locomotive

2.2.8 showswitch

SYNOPSIS

showswitch [NUM ...] [all] [thrown] [closed]

DESCRIPTION
Returns the status of the turnouts (switches) specified as arguments. The "all" argument causes all turnouts to be returned; the "thrown" argument returns only those turnouts that are thrown; the "closed" argument returns only closed turnouts. If no arguments are specified, "all" is assumed.

RESPONSES

200  OK (if no switches available)
222  switch status
400  No such switch

2.2.9  showblock

SYNOPSIS

showblock [NUM ...] [all] [busy]

DESCRIPTION

Returns the status of blocks in the layout. The "all" argument returns all blocks; the "busy" argument returns only those blocks that are occupied by a locomotive. If no arguments are specified, "all" is assumed.

RESPONSES

200  OK (if no blocks listed)
223  block status
400  No such block

2.2.10 status

SYNOPSIS
status [refresh=SEC]

DESCRIPTION

The status command provides a mechanism for watching the details of the simulation as it takes place. The output of this command is intended primarily for debugging and displaying RRsim, and is not expected to be normally available when working with a "real-world" train system. If a refresh attribute is specified, then a new status display is generated at least once every SEC seconds (however, it may be much more frequent than this depending on other interactions with the system). Omitting the refresh attribute causes a status display to be whenever a command is sent to RRsim from any client.

RESPONSES

200 OK

2.2.11 waitshow

SYNOPSIS

waitshow [loco=LOCO] [block=BLOCK] [switch=SWITCH]

DESCRIPTION

The waitshow command causes the server to wait and send a response only when a change in locomotive speed, block status, or turnout state is detected. When a change is detected, the server returns status responses according to the conditions set by the "loco", "block", and "switch" arguments. The arguments and responses for the waitshow request are similar to the "showloco", "showswitch", and "showblock" requests given above (2.2.7-2.2.9). LOCO may be a locomotive number, "all", or "none" to send 221 status responses for a particular locomotive, all locomotives, or no locomotives. Similarly, BLOCK may be a block number, "busy", "all", or "none"; and
SWITCH may be a switch number, "thrown", "closed", "all", or "none". A server is allowed to respond to the waitshow request even if items requested are not the cause of the change in state. For example, a server may send a response to a "waitshow block=101" request even if block 101 itself did not change. The default arguments for the waitshow request are "loco=all", block=busy", and "switch=thrown".

RESPONSES

200   OK
221   locomotive status
222   switch status
223   block status

3. Format of the layout file

A layout file is an ASCII text file used to specify the track layout and the initial positions of the locomotives. Each line in the file may be blank, a comment line (beginning with '#'), an "addblock" command, or an "addloco" command. Other lines will result in error messages when loaded.

3.1 Track (block) layout specifications

The "addblock" command is used to describe the track (block) layout of the simulation. The simulation software knows about two types of blocks: "linear blocks" that have only two ends, and "turnout blocks" that have a turnout mechanism to allow a train to be switched from one block to another.
Linear blocks: Linear blocks are the easiest to conceptualize. An example linear block is given as follows:

```
addblock 113 sblk=112 cblk=101 cdist=60
```

This command says to create block 113 that starts at block 112 (sblk) and connects to block 101 (cblk) with a distance of 60 units (cdist). Thus, any locomotive that enters block 113 from block 112 will arrive at block 101 after traveling 60 units. Similarly, any locomotive entering block 113 from block 101 will arrive at block 112 after traveling 60 units. Distances in the simulation can be whatever units you desire--using scientific units such as centimeters (cm) is recommended.

Turnout blocks: A turnout block contains a switch track that can be either "closed" or "thrown". Thus, a locomotive entering a turnout block may be sent to one of two destination blocks depending on the state of the turnout. A sample turnout block is specified as:

```
addblock 102 sblk=103 cblk=101 cdist=23 tblk=201 tdist=20 swi=12
```

This command says to create block 102 that starts with block 103 (sblk) and connects to block 101 (cblk) with a distance of 23 (cdist) whenever the turnout is closed. If the turnout is thrown, then block 102 connects block 103 to block 201 (tblk) with a distance of 20 (tdist). The switch number (swi) associated with this block is 12.

Note that the starting block must always specify the "point" of the switch, and a locomotive's destination is determined at the moment it enters the turnout block. In other words, if a turnout is changed after the locomotive has entered the turnout block, the locomotive will not respond to the change.
If a locomotive enters a turnout block from either the cblk or the tblk, it will travel to the starting block regardless of the closed/thrown setting.

Turnout blocks have an optional "isthrown=X" setting, where "X" is non-zero to indicate a thrown switch. The attributes "throw" and "close" are synonyms for "isthrown=1" and "isthrown=0" respectively.

Dead ends: Block 0 is a special block used to model bumpers or ends of track. Thus the addblock command

```
addblock 404 sblk=403 cblk=0 cdist=48
```
says that a train entering block 404 from block 403 will reach a dead end (cblk=0) after traveling 48 units. Turnout blocks may also have dead ends.

3.2 Locomotive specifications

Locomotives are specified using the "addloco" command. For example, the command

```
addloco 4 blk=203 maxspeed=5
```
says to add locomotive number 4 starting at block 203 and its speed at 100% throttle is 5 units/sec. By default, a locomotive will be placed at the start point of a block and moving in the forward direction. If the maxspeed attribute is not specified a default of 5 units/sec is assumed.

The attribute "isrev=1" can be added to specify that the locomotive should initially start in reverse. The attributes "rev" and "fwd" are shortcuts for "isrev=1" and "isrev=0" respectively.
4. Simulator limitations

Things which RRsim does not model (yet):

- Locomotives do not have any length, thus they are considered to be "point source" objects.
- Locomotives do not have any mass or inertia, thus they respond instantaneously to speed and direction changes.
- Collisions between locomotives are not detected.
- Track grades are not modeled.
- In the simulation if a locomotive reaches a dead end, it just stops. In the real world the result is often more disastrous.
- Throwing a turnout block while a locomotive is in the block does not affect the locomotive.
- Invalid parameters (such as negative speeds and times) are likely to completely confuse RRsim.