Multi-User Computer Controlled Railroad

Documentation & Manuals

Samuel Joski

CSCI-460 Senior Capstone Experience

22 Jan. – 10 May 2018

Table of Contents:

Introduction

Definition:

Communicating with multiple robots that share common resources is rather common. Your Amazon order might trigger controls on a robot in the warehouse. It is instructed to ride rails to locate items on shelves and bring them to the shipping department. More complex is dealing with multiple robots filling orders for several customers while trying to share rails and avoid collisions.

Design a multiple user operating system for the Computer Controlled Railroad that allows multiple trains to operate in real time on a shared track layout controlled remotely by several users.

General Requirements:

1. Each user becomes a CAB that controls one train at a time.
2. Decide on a CAB platform.
3. Each CAB shows the state of the layout system.
4. Develop an API that communicates with the Digital Command Control (DCC) system.
5. Use operating system concepts to deal with racing conditions, resource allocation, scheduling, etc.
6. Deal with starvation and deadlock issues.
7. Identify error conditions and attempt to solve them.
8. See Dr. Pankratz for CCR inputs and outputs.

Overview:

This project utilizes the DCC-style model railroad of the SNC Computer Science department. There are five major components of the project:

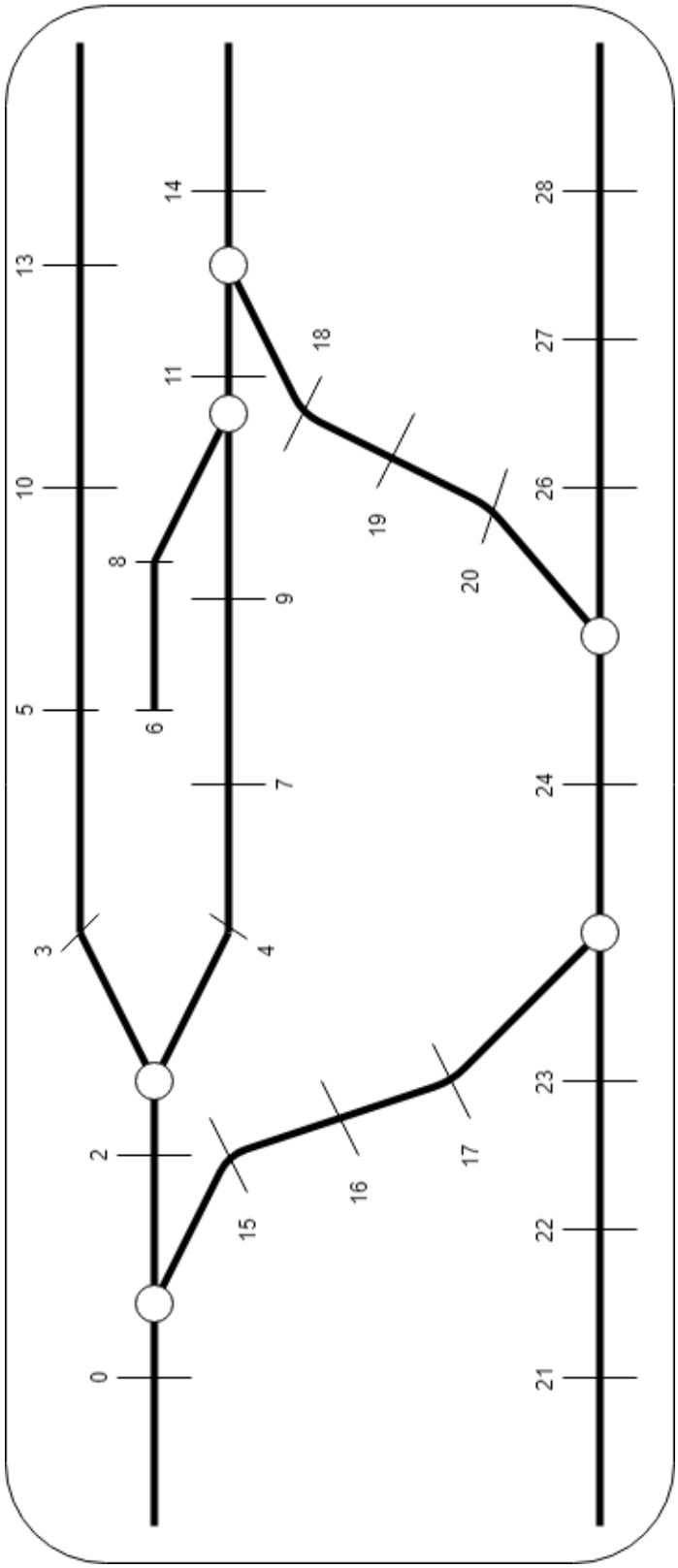
- The Railroad: the railroad itself is a piece of hardware with motors and sensors with which a programmer of the project must interact. The railroad is interfaced via the Digital Command Control boxes. One controls sensor reading, and the other relays commands from the PC into which the COM port is connected.
- Resource files: the resource files hold the information about the track layout, sensor locations, and turnout and train states. They act as the only input to the *Server Application.*
- Server Application: the server application is responsible for reading input from resource files, interpreting them, and if necessary send instructions via the DCC boxes to the railroad. The server application is broken into components:
    - COM Select: the COM Select reads the PC's USB ports to identify the one into which the DCC COM is connected. It sends the selection to the *CCR Server* component.
    - CCR Server: the CCR Server is the "View" in a Model-View-Controller concept of the server application. It does some work by reading files and calling other controls, but it does not (in most cases) directly contact the Railroad via the COM Port. During file reads, *railroad resources'* methods are called to execute the logical components of the server application.

- <u>Railroad Resources:</u> the railroad resources are the classes that contain the data and methods to manage the railroad. It determines if trains can move, if turnouts can switch, etc. The railroad resource methods are the ones that call the *Command DCC* functions to physically make changes to the railroad.
- <u>Command DCC:</u> the Command DCC component is in charge of directly sending information to the DCC box via the COM port. It also has methods to read information from the track sensors, which are set on a timer in the *CCR Server* component

- <u>Master CAB:</u> the Master CAB sits alongside the *Server Application* in a directory. It therefore has visibility of the *resource files* that act as the *Server Application*'s input. The Master CAB is responsible for refreshing its data to match the *resource files*, as any differences between the Master CAB's data and the files signals an action that was unable to be requested.
- <u>User CAB:</u> the User CAB brings in the "multi-user" aspect of the project. The design that was implemented was a WAMP server that other PCs could connect to. HTML & PHP display a controller to the user on a selected train, and send information to the directory on the PC in which the *Server Application* is running. It would act via the WAMP connection to the *Server Application* just as the *Master CAB* does: via the *resource files.* However, this section is **<u>INCOMPLETE</u>***,* and does not function. It needs to be further developed to communicate via WAMP with the *resource files*.

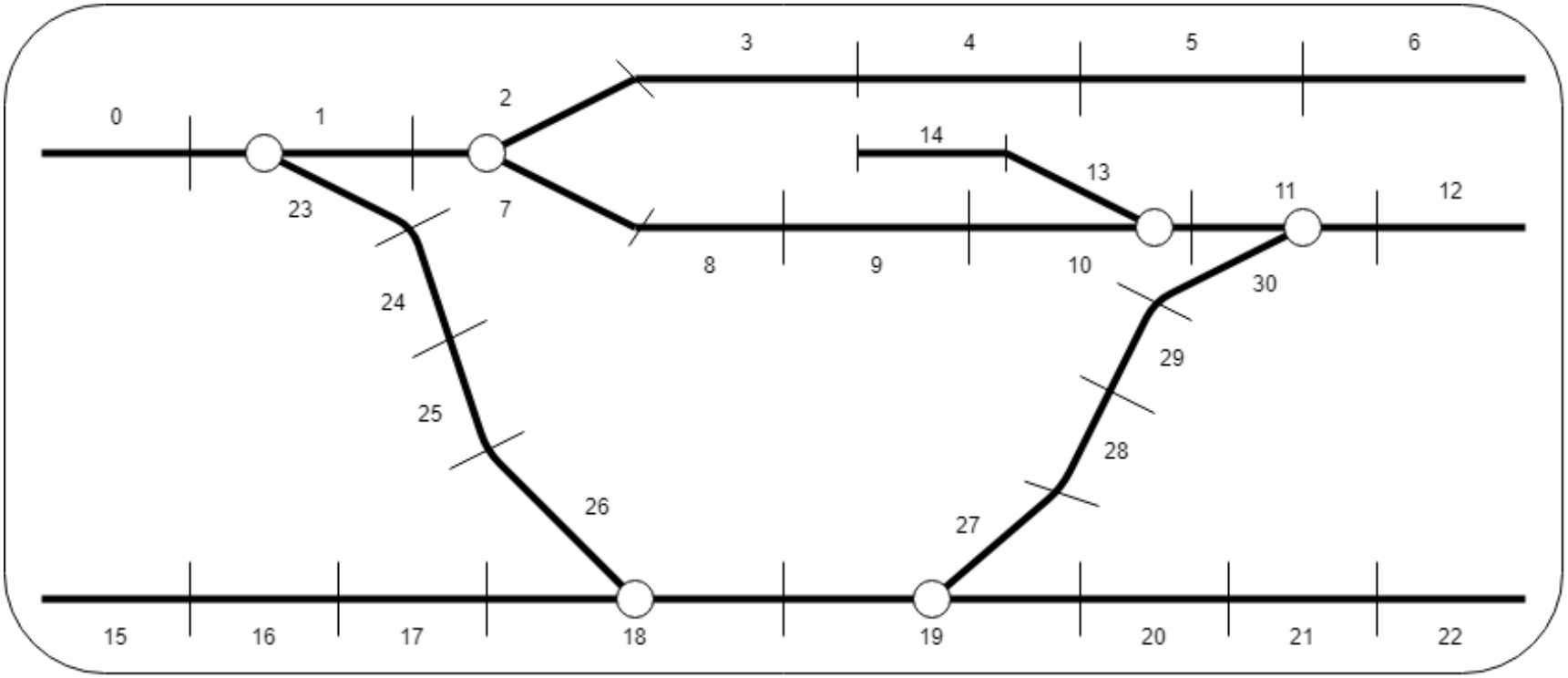
Diagrams

Railroad Layout: Sensors

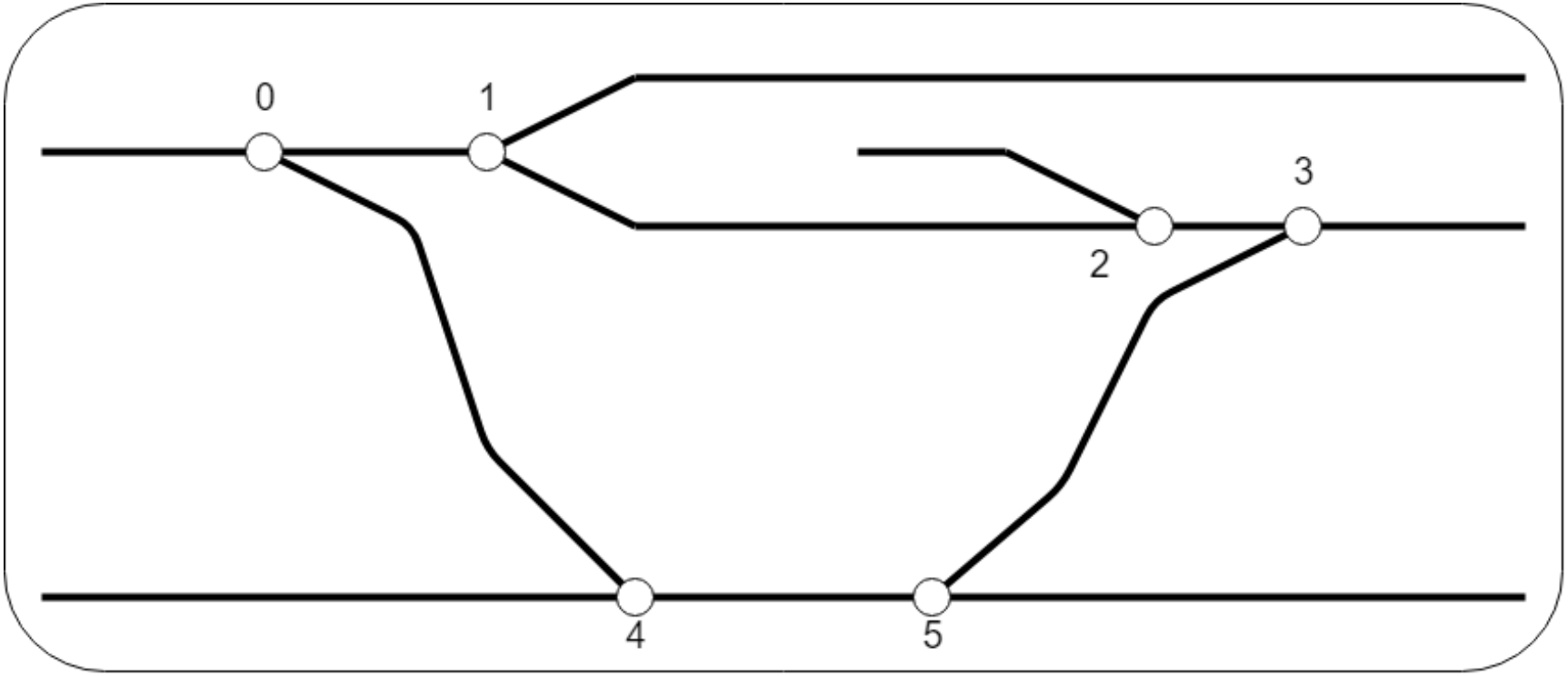


(Notice the lack of sensors 1, 12, and 25. These sensors are on the railroad, but unused here.)

Railroad Layout: Sections



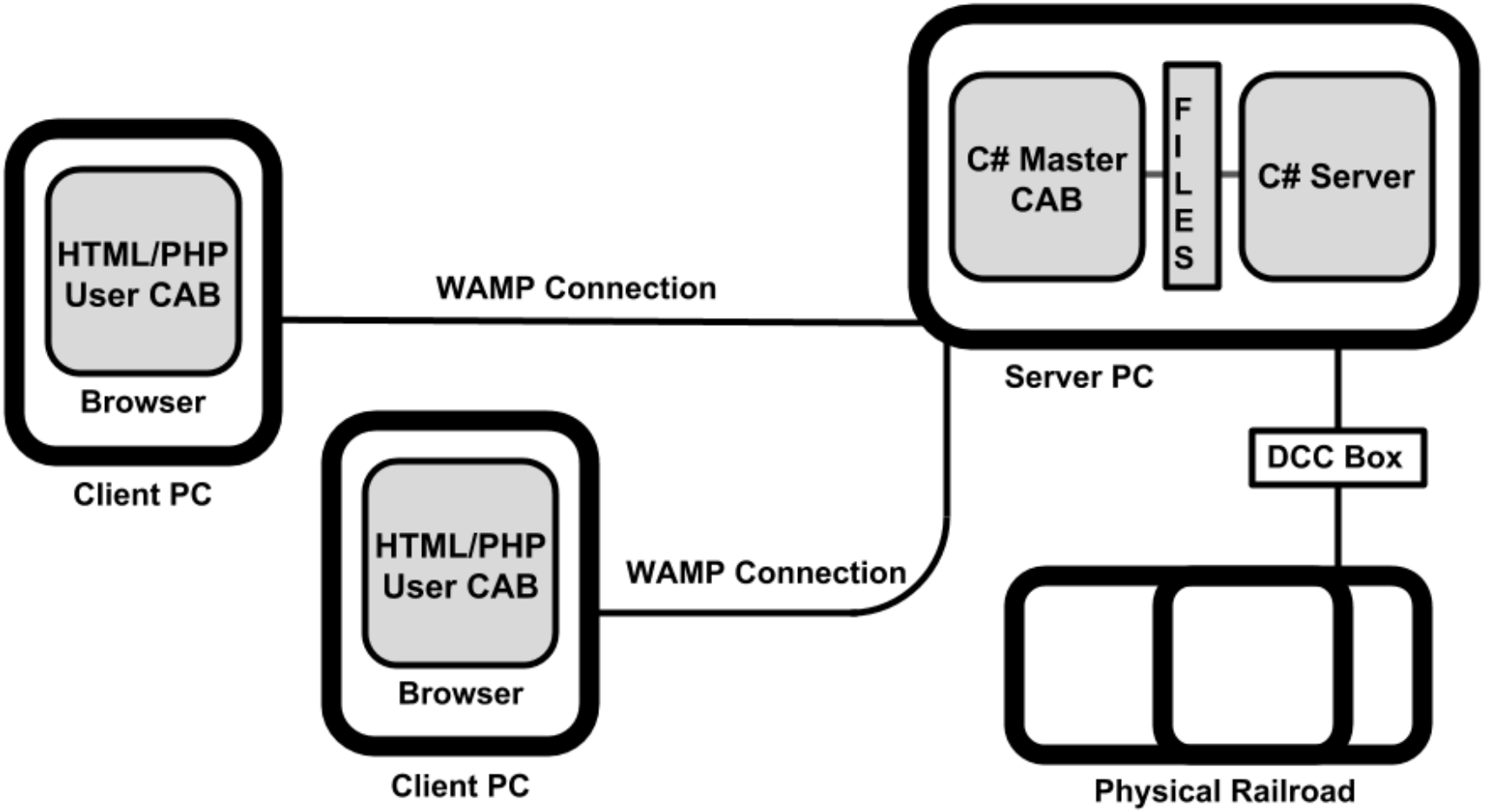
Railroad Layout: Turnouts



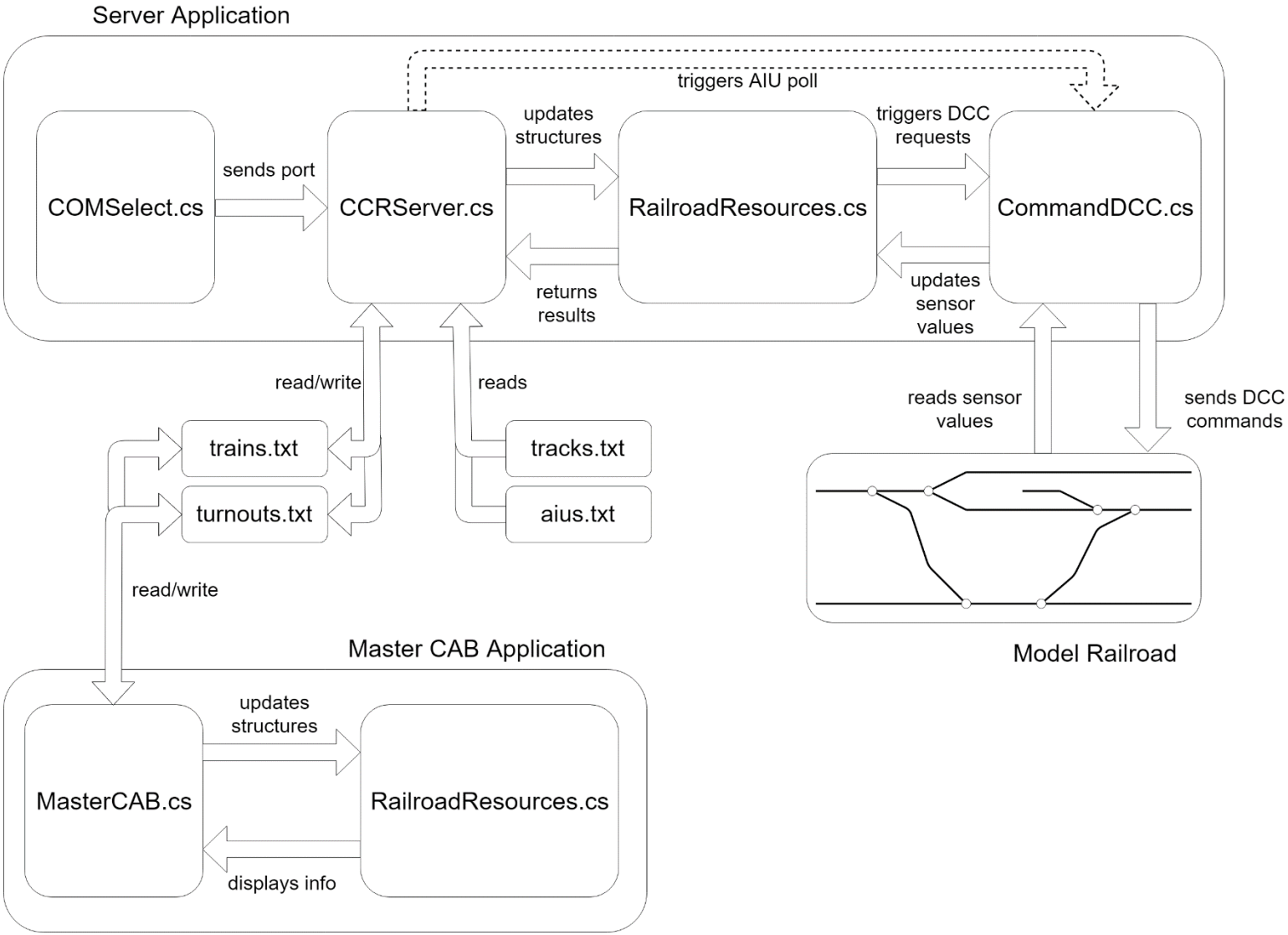
The addresses of the turnouts are:

0	→	24	3	→	8
1	→	36	4	→	14
2	→	16	5	→	4

Overview Diagram



Data Flow


Server Application
triggers AIU poll
COMSelect.cs
sends port
CCRServer.cs
updates structures
RailroadResources.cs
triggers DCC requests
CommandDCC.cs
returns results
updates sensor values
read/write
reads
reads sensor values
sends DCC commands
trains.txt
turnouts.txt
tracks.txt
aius.txt
read/write
Model Railroad
Master CAB Application
updates structures
MasterCAB.cs
RailroadResources.cs
displays info

File Structures

All file structures are set up as comma delimited. Each line represents an element of the structure (i.e. one train, one track, one turnout, one AIU).

Trains (trains.txt)

ID, Address, Speed, Direction, Orientation, Status, Headlight, Horn, Occupied

Tracks (tracks.txt)

ID, Left top, Left bottom, Right top, Right bottom, Speed limit, Turnout ID, Left sensor, Right sensor

Turnouts (turnouts.txt)

ID, Address, Direction, Occupied

AIUs (aius.txt)

ID, Address, [Sensor ID]

The Sensor IDs of an AIU may be 0, 1, or many – each separated by a comma.

Instructions

Setup

Begin by ensuring that the railroad is assembled correctly. Once the railroad is assembled, plug in the DCC boxes to the correct jacks. The thicker cable plugs into the DCC command box, while the grey CAT-3 cable plugs into the AIU box. Make sure that the boxes are correctly connected and that the COM cable is attached. Plug in the power cable to a power-outlet, and plug in the COM cable to the PC that will run the *Server Application*. Turn on both DCC boxes.

Start-up

Once the COM cable is attached and the DCC boxes are powered on, download the zipped folders “CCRServer” and “MasterCAB”. Extract the folders. Once both are extracted, copy the executable “MasterCAB.exe” into the “CCRServer” folder.

Once the CAB is copied, the files in the directory should include:

- CCRServer Visual Studio Project Folder
- CCRServer.exe
- MasterCAB.exe
- Resource files "train.txt", "turnouts.txt", "aius.txt", and "tracks.txt"
- Text file “about.txt”

Run “CCRServer.exe” to bring up the *COM Select* window. A COM port should be listed in the selection. Click the port, and “Select Port”. The window should close and the *CCR Server* window should appear.

How-to's

Add a train

To add a train, go to *Edit* → *Add Train*. A window should appear prompting for information. Enter the train's address (found on top of the locomotive), its orientation (facing left or right, bottom being the table side into which the cables plug), and its location (see *Diagrams – Railroad Layouts – Sections* on page 6). Once all information has been entered, click "Confirm".

Remove a train

To remove a train, go to *Edit* → *Remove Train.* A window should appear prompting for information. Enter the train's address (found on top of the locomotive). If the train is not occupied, it will be removed from the *Server Application* and associated files.

Control a train

To control a train, run "MasterCAB.exe". Once the window appears, any trains that were added to the *Server Application* should be displayed. To control one, click the displayed train address, and then click "Select". If the train was not already occupied, its information will be displayed.

Once the train is occupied, simply click the "Up" button to increase its speed forward, or the "Down" button to increase its speed backward. Click the "Stop" button to bring the train to a halt.

Toggle turnouts

With the *Master CAB* application running, click on a turnout button. If a train is not occupying an associated section of the turnout, the turnout will be flipped on the railroad, and the button will display its current state.

Power down

To power down the CCR, in the *Server Application*, go to *File* → *Exit*. Any moving trains will be brought to a halt, all trains will be deallocated from the server, and all associated files updated to signal to any CABs that no trains are available.

File → *Disconnect* can be used in a similar fashion to instead disconnect from the railroad (stopping all trains, deallocating them and updating files) and bring the window back to the *COM Select* window.

Exceptions

- Occasionally, powering down will throw an exception. This occurs because threads still running attempt to access the DCC port after it has been closed.
- If the *Master CAB* application is closed, any trains that were occupied remain occupied, and the train will not be halted. The train will however stop if and when the Server determines it is no longer safe for the train to continue, as the train will continue to exist in the *Server Application* list of trains. The train cannot be removed unless the *Server Application* is restarted, as the train will be occupied and therefore unable to be removed.
- It is not uncommon for a sensor to miss the magnet attached to the locomotive. This will cause the *Server Application* to lose sight of the train. When the train hits its next sensor, it will not update the train location, as the *Server Application* is still expecting it to trip the one that it missed.
- Derailments are also not uncommon. This is due to the physical construction of the railroad.
- The *User CAB* web controller is **NOT** functional. Scripts need to be written to communicate to the web server (WAMP) to read and write from files.

Source Code

The full source code for the project is zipped and can be found at:

compsci02.snc.edu/cs460/2018/josksa/documents.html

CCR Server:

Points of interests:

- *private void readFilesEvent(object source, ElapsedEventArgs e)*

  This function is the ElapsedEventHandler of a timer called *fileReader*. This event occurs in the background, and reads the states of the turnouts and trains from their *Resource Files*. Any changes detected are send to *RailroadResources.cs* to update data structures and make calls to the railroad via *CommandDCC.cs* if necessary.

- *private void pollAIUEvent(object source, ElapsedEventArgs e)*

  This function is the ElapsedEventHandler of a timer called *pollAIUEvent*. This event occurs in the background, and calls the method: *public Train[] PollAIU(AIU aiu, Train[] trains)* in the class *CommandDCC.cs*. This will detect if any trains have reached their next destination, kicking off the process of determining a train's next location. Any changes that result area written back to the *Resource Files* via *CCRServer.cs* method *private bool writeTrain(Train train)*.

- *public bool setSpeed(int speed)*

  This method of the *Train* class in *RailroadResource.cs* is called from the *readTrains* function of *CCRServer.cs*. It determines if the direction of the train needs to switch, and calls the DCC railroad via *CommandDCC* to physically change the train speed.

- *public bool Move()*

  This method of the *Train* class in *RailroadResources.cs* is called from *EvaluateDCCPoll* method of the *AIU* class. It is a lengthy method which determines if a train can move, where its next section may be, etc. It also calls functions of various classes to request and release resources that are appropriate to the train moving.

- *public Track getNextSection(int direction)*

  This method of the *Track* class in the *RailroadResources.cs* is called from the *Move* method of the *Train* class. It determines through the connection array of a Track section and based on the sent *direction* parameter what track is connected to the current track. It will evaluate any turnout that may be associated with the track, and will catch any dead-ends by spotting *null* connections.

- *public Train[] EvaluateDCCPoll(int[] sensorValues, Train[] trains)*

  This method of the *AIU* class in the *RailroadResources.cs* is called from the *PollAIU* method of *CommandDCC.cs.* It checks all sensors associated with the AIU for any values that may have changed during the reading from the railroad done by the COM port.

Master CAB:

Points of interests:

- *private void readFilesEvent(object source, ElapsedEventArgs e)*

  This function is the ElapsedEventHandler of a timer called *fileRead*. This event occurs in the background, and reads the states of the turnouts and trains from their *Resource Files*. Any changes detected are send to *RailroadResources.cs* to update data structures. This indicated that an action requested was denied by the server, and previous data was written back to the *Resource File* by the *Server Application*.

- *RailroadResources.cs*

  This version of the *RailRoadResources* classes is drastically different from the one on the *Server Application.* It acts here as a collection of data, with simple methods to manipulate and retrieve the data.