

A Laboratory Platform to Control a Digital Model Railroad Over the Web Using Java

Roger W. Webster, Ph.D. Mary A. Klaus Timothy A. Bish
Roger.Webster@millersville.edu makluas@cs.millersville.edu tabish@cs.millersville.edu

Department of Computer Science
Millersville University
Millersville, PA USA 17551

1. ABSTRACT

This paper describes the work-in-progress of a client-server system to control a digital model railroad over the World Wide Web Using Java. The software engineering objective of this real-time system is to maintain control of multiple digital locomotives each running on the same track layout while at the same time allowing users, anywhere in the world, to manually control the operation of the trains using a Java applet running in a web browser. A video camera is connected to the web server showing the users a video stream of the actual physical train system. The Java client allows the user to: stop, reverse, and change the speed of any train (by address). Also, the user can switch any of the computer-connected turnouts on the layout. The control software (Java server) constantly monitors reed contact sensors to keep track of each train's location and direction, and is continuously performing collision avoidance testing. Each digital locomotive and digital turnout switch responds to computer commands that are sent to its address. The computer system, an Intel Pentium running Windows NT®, runs its own web server at <http://javatrains.millersv.edu/>. This laboratory platform requires students to utilize and exercise their knowledge of mathematics, physics, engineering, real-time programming and computer science.

2. INTRODUCTION

In this railroad layout there are 4 digital turnout switches, two digital locomotives, and fifteen reed contact sensors to manage and control (see Figure 1). The fifteen reed contact sensors are placed in appropriate locations around the track (Figure 1). Magnets are attached to each locomotive, which trip reed contact switches, which are implanted in the track. This configuration provides an interesting, experimental platform for the study of controlling a real-time system using a Java client-server architecture, for undergraduates in Computer Science and Computer Engineering. This laboratory platform requires students to utilize and exercise their knowledge of

mathematics, physics, engineering, computer science, and real-time programming. A physical model railroad was used because theoretical modeling and graphics simulations do not always manifest the frustrating and spasmodic problems endemic in actual real-time systems.

3. HARDWARE AND SOFTWARE

The Java server and webserver are run on an Intel Pentium computer running Windows NT® with 32-MB memory and a 1 GB Hard Disk drive. The Marklin® digital railroad system is used to interface the computer to the track as depicted in Figure 1. The Marklin® system is comprised of six interconnected components: a Central Unit, Computer Interface, Keyboard Turnout Control, Track Detection Module (TDM), Control 80f, and a Transformer. All Marklin® modules or components plug together to form a bus architecture between components. The Central Unit is the CPU of the Marklin® system. The Central Unit receives commands from the other modules that control turnouts and locomotives. The locomotives are digitally encoded with a chipset that is addressable, therefore messy block wiring to turn the power on and off is unnecessary. The Central Unit overlays each command on the electric current thereby sending a signal to the track where it is received by the specific decoder for which it is addressed (for example, the C82 decoder chip in each locomotive or the K87 turnout decoder for switch tracks). The S88 Track Detection module (TDM) is an encoder, which translates the incoming signals from the reed contact sensors into a data format that the digital system can then use. The Control 80f module is simply a manual control knob for setting the speed and direction of any digital locomotive. The K87 Digital Turnout control module can digitally switch up to four turnouts. Multiple K87's can be connected in series. The K87 will respond to track switch commands from either the Marklin® Keyboard component or the Computer Interface module.

The Marklin® Computer Interface module is the link between the computer and the Marklin® Digital HO gauge system. Using an RS232 serial interface, all the functions of the Control 80f and the Keyboard Digital Turnout module can be sent as commands from the

computer to the interface module. In addition, a computer command can be sent to the interface to query the TDM information, which specifies which reed contacts have been tripped. In all, up to 80 locomotives, 256 turnout switches, and 496 reed sensors can be controlled or monitored with the computer interface.

4. INTERFACE TO DIGITAL RAILROAD

The Java server sends the low level commands from the computer to the Marklin® Computer Interface hardware via RS232. Methods such as TRAINHALT() were written to initialize and shut down the Marklin® system. The method TRAINSPEED(train-number, speed) issued the addressable speed command, thus each train could be separately controlled. TRAINSTOP(train-number) stopped the train train-number, but not the other trains running. TRAINSWITCH(switchnumber, curved-or-straight) would switch the digital turnout to either its straight or curved position. TRAINREVERSE(train-number) reversed the train. The function call TRAINGET-TDM(tdm1, tdm2) returns the two bytes sent by the Marklin® Track Detection Module. The first byte, tdm1, contains the sensor information for the first 8 sensors on the track. Tdm2 contains sensors 9 through 16. A magnet on the train will trip the reed sensor when it crosses. A bit is on if the sensor has been tripped. The device latches the bit until a computer command read, which resets it to zero. It is interesting to note that a slow train could trip the reed sensor twice. Thus a double hit occurs. This happens when the sensor is tripped, read by a computer read command (inquiry), reset to 0, then read again by the software before the train has completely bypassed the sensor. This is taken care of in the software by masking off the previous reed sensor data.

5. JAVA CLIENT SOFTWARE

The Java client (see figure 1) allows the user to manually control the operation of the trains from anywhere in the world. This Java applet allows the user to: stop, reverse, and change the speed of any train (by address). Also, the user can switch any of the computer-connected turnouts on the layout. The Java client sends commands to the server to determine the viability of the request. Thus, the user is not permitted to make a change that would cause a crash. If so, the request is denied by the server.

6. JAVA SERVER SOFTWARE

The Java server is actually three separate tasks all continuously looping and executing their jobs once for each pass through their loop. The first task is the server to the client. This process simply takes commands from the client and passes them on to the next task, the AI. A timeout is set up to notify the client that something has

gone wrong and ask him to restart if the tasks take too long to respond. The simplicity of this task reduces its chance of failure so that the user can be kept informed if other problems occur.

The second task, the AI receives TDM data from the scan task and uses this to keep track of the positions of each of the trains. If a command sent by the user would result in a collision it modifies or ignores the command and sends this information back to the client so the interface can be updated. The AI contains the code to detect collisions. When one train approaches another train too closely, the AI either issues a slow down command or a stop(train-address) to the train behind, depending on how imminent the collision is. The controller does not want to stop a train unless it is imperative to do so. In imminent situations the AI may issue both a train slow down command to the rear locomotive, and a train speed up to the front locomotive. If the user issues a command the flip a switch, the AI determines if it is safe to switch, and if so, issues the command to the Marklin® system. The scan task is the one that actually talks to the trains. It receives commands from the AI and sends these out to the trains. All commands are sent as one or two bytes. The first byte contains the command code and the second (when appropriate) contains additional information such as train speed. When there are no commands coming in, the scan task continuously asks for TDM data from the trains. This task also makes sure that only one command is sent between successive TDM calls. The scan task gets TDM data by calling the method getTDM() which returns the two bytes sent by the Marklin® Track Detection Module. The first byte contains the sensor information for the first 8 sensors on the track. The second byte contains sensors 9 through 16. The decoding of the sensor data returned by getTDM is accomplished by left shifting the first byte and combining these two bytes into one word. This data is then sent on to the AI.

The AI knows the current direction (forward or backward) of each train, its previous position (which sensor it last tripped) and the state (straight or curved) of each switch. However, the contact does not know which train crossed, just that some train (with a magnet) has crossed. Thus, tripping a contact is not an addressable event. Ambiguity can arise due the fact that tripping a contact is not an addressable event. The AI task figures out which train it probably is given the monitoring information it is maintaining. Using this information, it translates the TDM data into a new position for each train by looking up information about possible next positions for each train in an array. For example, if a train was previously at sensor 11 and all switches were straight, it shouldn't be at sensor 9 the next time. A bit is on if the sensor has been tripped. If the function returns more than two sensors tripped, at least one of the trains has crossed more than one sensor

since the last update or some hardware

All commands are sent as one byte. The upper nibble contains command code and the lower nibble contains additional information, when required, such as in the case of train speed adjustment for example. The server task is responsible for all control of the system. This task accepts all user commands from the Java client and determines if current conditions on the train layout will allow the command to be executed safely (without causing a collision or derailment). If so, the command is executed otherwise the command is blocked from the Marklin® system. The server task keeps track of vital information for each train such as: location, speed, direction, and current zone or sector.

Each time a sensor is tripped, the sensor value is used to index a lookup table, which contains the previous value for each sensor on the track layout. In this manner it is possible to monitor the trains without addressable track detection information. The reed contact will signal the fact that a train (a magnet) has crossed the track. However, the contact does not know which train crossed, just that some train (with a magnet) has crossed. Thus, tripping a contact is not an addressable event. Ambiguity can arise due the fact that tripping a contract is not an addressable event. The Java server control software figures out which train it probably is given the monitoring information it is maintaining. For example, suppose the current sensor read is 8 and the direction is 0. The previous sensor would be 14. This value is compared to the location of each train in the data structure. If a match is found the current sensor value is stored in the location field for that train. If no match is found the system issues a TRAINHALT indicating a lost train, and the server shuts down. In this manner the server always knows where each train is at any time and is never allowed to lag behind.

The Java server contains the code to detect collisions. When one train approaches another train too closely, the server either issues a slow down command to the train behind depending on how imminent the collision is. The controller does not want to stop a train unless it is imperative to do so. In imminent situations the server may issue both a train slow down command to the rear locomotive, and a train speed up to the front locomotive. Upon each train arriving at a switch, the server determines if it is safe to switch, and if so, issues the command to the Marklin® system.

7. CONCLUSION

This paper has described the work-in-progress of a Java client-server controller for a digital model railroad. The control software does accomplish its objective of

maintaining control of multiple digital locomotives each running on the same track layout while at the same time allowing users around the world to manually control the operation of the trains using a Java applet running in a web browser. A video camera is connected to the web server showing the users a video stream of the train system. The Java client allows the user to: stop, reverse, and change the speed of any train (by address). Also, the user can switch any of the computer-connected turnouts on the layout. The control software constantly monitors reed contact sensors to keep track of each train's location and direction, and is continuously performing collision avoidance testing. The project was initiated to provide an interesting, experimental platform for the study of controlling a real-time system over the world wide web with a Java client-server architecture. This laboratory platform requires students to utilize and exercise their knowledge of mathematics, physics, engineering, real-time programming and computer science. Further information and source code can be found on our web site at <http://cs.millersville.edu/javatrains/>.

8. ACKNOWLEDGEMENTS

This project was funded, in part, by the National Science Foundation under grant numbers DUE-9350841 and DUE-9651237, and by the Faculty Grants Committee of Millersville University. Many thanks go to Mrs. Bonnie Work, for administrative assistance. Special thanks go to Robert Sauders setting up the DNS entry Javatrains.millersv.edu.

9. REFERENCES

- [1] Catherall, Thomas, "2-Rail Digital DC", Marklin® Digital SIG Newsletter, Vol. 2, No. 1, New Berlin, Wisconsin, January 1990, pps 1-8.
- [2] Catherall, Thomas, "Sending Data From the Train to the Digital Component", Marklin® Digital SIG Newsletter, Vol. 2 No. 3, New Berlin, Wisconsin, May 1990, pps 1-10.
- [3] Cassidy, Luke, "Industrial Strength Java", New Riders Press, Indianapolis, IN., 1997.
- [4] Flanagan, David, "Java 1.1 in a Nutshell", Second Edition, O'Reilly Press, Cambridge, Massachusetts, 1997.
- [5] Flanagan, David, "Java 1.1 Examples", O'Reilly Press, Cambridge, Massachusetts, 1997.
- [6] Geary, David M., "Graphic Java 1.1 - Mastering the AWT", Second Edition, Sun Microsystems Press, Mountain View, CA, 1997.
- [7] McCormick, John W., "A Model Railroad for Ada and Software Engineering", Communications of the ACM, November 1992, Vol. 35, No. 11, pp. 68-70.

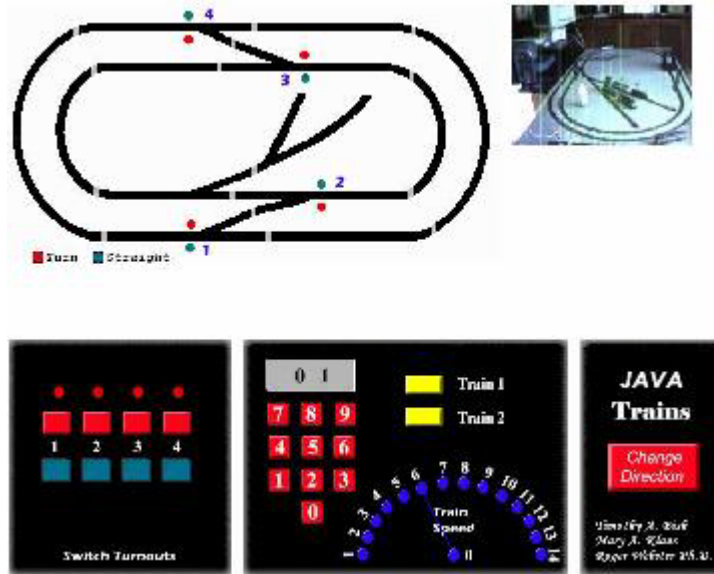


Figure 1. Digital model railroad software client and track layout. Reed contacts are numbered 1 through 15. Digital turnouts are numbered SW1 through SW6.