IML lab Real-Time Digital Model Railroad Project

Proceedings of the IEEE Conference on Real-Time Applications
May 13-14, 1993
New York, N.Y.
Sponsored by: IEEE Computer Society, Office of Naval Research, Honeywell Incorporated and Naval Surface Warfare Center.

A Real-Time Software Controller for a Digital Model Railroad System*

by
Roger W. Webster, Ph.D.
David Hess
Intelligent Machines Laboratory
Department of Computer Science
Millersville University
Millersville, PA USA 17551
(717) 872-3539
email: webster@cs.millersv.edu

Abstract

This paper describes a real-time software controller for a digital model railroad.
primitives of fork, pipe, and signal are used to perform interprocess communicatic
executing tasks, (1) a Scanning Task, (2) a Scheduler and Collision Avoidance task,
Interface (GUI) task. The software engineering objective of this real-time system i
digital locomotives each running on the same track layout while at the same time al
scheduling system to "run" the trains. The control software continuously monitors r
of each train's location and direction, and is constantly performing collision avoi
digitally encoded with a chipset that is addressable, therefore messy block wiring
unnecessary. Each digital locomotive and digital turnout switch responds to compute
address.

I. Introduction.

In this railroad layout there are six digital turnout switches, two digital locomot
to manage and control (Figure 1). The objective is to move the trains around the tr
to the scheduling algorithm without collision. The fifteen reed contact sensors are
around the track (Figure 2). Magnets are attached to each locomotive which trip ree
implanted in the track. This configuration provides an interesting, experimental pl

real-time systems for undergraduates in Computer Science and Computer Engineering. while many undergraduate courses in Real-Time Systems acquaint students with the fu

---

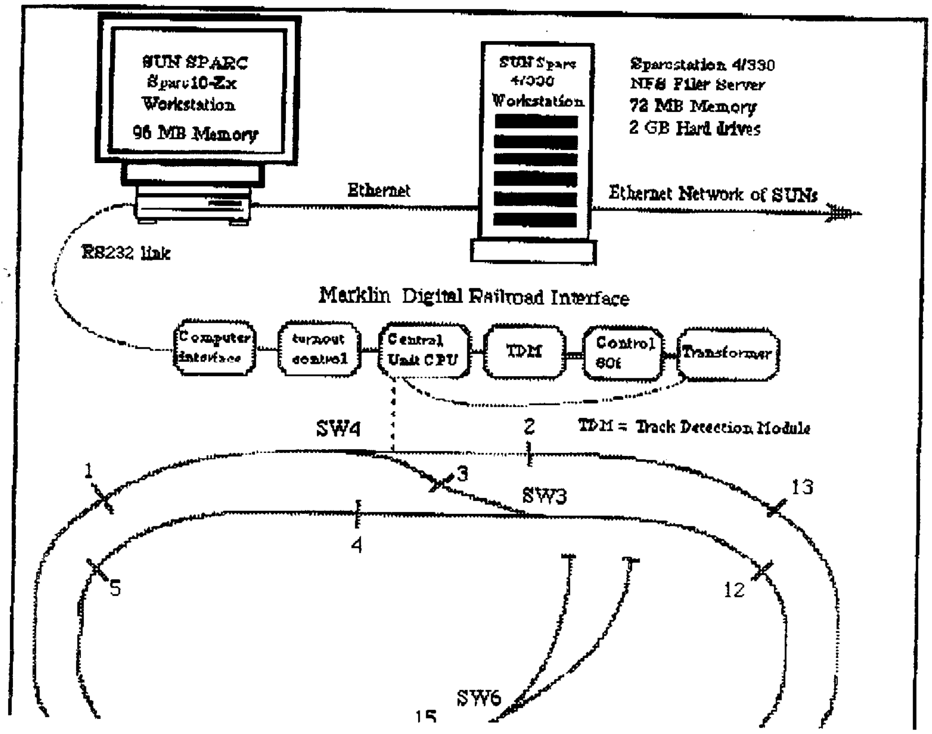
* This work partially funded by the National Science Foundation (NSF) under Grant N Millersville University and by the Faculty Grants Committee of Millersville Univers

in real-time computing, many do not provide adequate laboratory platforms to exerci to build physical real-time systems[2]. Theoretical modelling and graphics simulati frustrating and spasmodic problems endemic in actual real-time systems. This labora to utilize and exercise their knowledge of mathematics, physics, engineering, compu programming.

II. Equipment, Hardware and Software.

The computer controller is a SUN SPARC workstation connected to a SUN 4/330 file se SUN IPC workstation has 16 MB memory and a 207 MB Hard Disk drive. The Marklin digi used to interface the SUN computer to the track as depicted in Figure 2. The Markli interconnected components: a Central Unit, Computer Interface, Keyboard Turnout Con Module (TDM), Control 80f, and a Transformer. All Marklin modules or components plu architecture between components.
The Central Unit is the CPU of the Marklin system. The Central Unit receives comman that control turnouts and locomotives[3]. The Central Unit overlays each command on sending a signal to the track where it is received by the specific decoder for whic C82 decoder chip in each locomotive or the K87 turnout decoder for switch tracks). module (TDM) is an encoder which translates the incoming signals from the reed cont that the digital system can then use. The Control 80f module is simply a manual cc and direction of any digital locomotive. The K87 Digital Turnout control module can turnouts. Multiple K87's can be connected in series. The K87 will respond to track Marklin Keyboard component or the Computer Interface module.

Figure 1. Photo of Digital Railroad System.



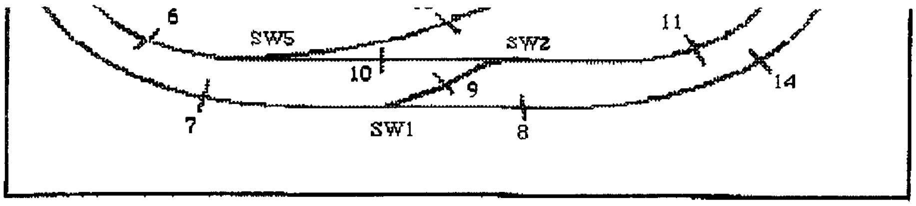


Figure 2. Hardware and Track Layout.
Reed contacts are numbered 1 through 15.
Digital turnouts are numbered SW1 through SW6.

The Computer Interface module is the link between the SUN IPC workstation and the M
system. Using an RS232 9600 baud serial interface, all the functions of the Control
Turnout module can be sent as commands from the computer to the interface module. I
command can be sent to the interface to query the TDM information which specifies w
tripped. In all, up to 80 locomotives, 256 turnout switches, and 496 reed sensors c
the computer interface.
The software is written entirely in 'C'. The 'C' language was chosen as the real-
reasons which are outlined in [4] and [5]. The SUN Developer's Guide was used to ge
Graphic Users Interface (GUI).

III. Device Driver Interface to Marklin System

A device driver was written in 'C' (TRAIN.C) containing the low level commands frc
Marklin Computer Interface hardware via RS232. Functions such TRAIN$START() and TRA
written to initialize and shut down the Marklin system. The function TRAIN$SPEED(tr
the addressable speed command, thus each train could be separately controlled. TRAI
stopped the train train-number, but not the other trains running. TRAIN$SWITCH(swit
straight) would switch the digital turnout to either it's straight or curved positi
reversed the train.
The function call TRAIN$GET-TDM(&tdm1, &tdm2) returns the two bytes sent by the Mar
Module. The first byte, tdm1, contains the sensor information for the first 8 sensc
sensors 9 through 16. A magnet on the train will trip the reed sensor when it cross
been tripped. The device latches the bit until a computer command read, which reset
It is interesting to note that a slow train could trip the reed sensor twice. Thus
when the sensor is tripped, read by a computer read command (inquiry), reset to 0,
before the train has completely bypassed the sensor. This is taken care of in the s
reed sensor data.

\Buuna

IV. Software Controller / Concurrent Tasking.

The real-time software controller consists of three separately executing concurrent
(2) a Scheduler and Collision Avoidance task, and (3) a Graphical User Interface (G
prefers to call tasks - processes, in this paper the terms will be used interchange
Graphical User Interface (GUI) task which allows the user to manually control the c
SUN workstation (Figure 4). This task allows the user to: stop, reverse, and chan
address). Also, the user can switch any of the computer connected turnouts on the l
the control mode and unrestricted mode. In control mode the user's requests are sen
the Scheduler Task to determine the viability of the request. Thus, the user is not
would cause a crash. If so, the request is blocked. In unrestricted mode, the user'
Marklin digital system without collision avoidance checks and therefore could cause
The parent process spawns two child processes, the Scan task and the Scheduler task
the two children via a pipe called Control-Pipe. Both children have the ability to
separate conditions. The Scan task will read the control pipe if the GUI task is i
its user commands directly to the Scan task. The Scheduler task will read the con
control mode, in this way user commands initiated from the GUI task will be sent tc
viability then, if viable, sent via the Command-Pipe to the Scan task.
The Scan task has two jobs and continuously loops executing both jobs once for each
job is to collect and decode the current reed

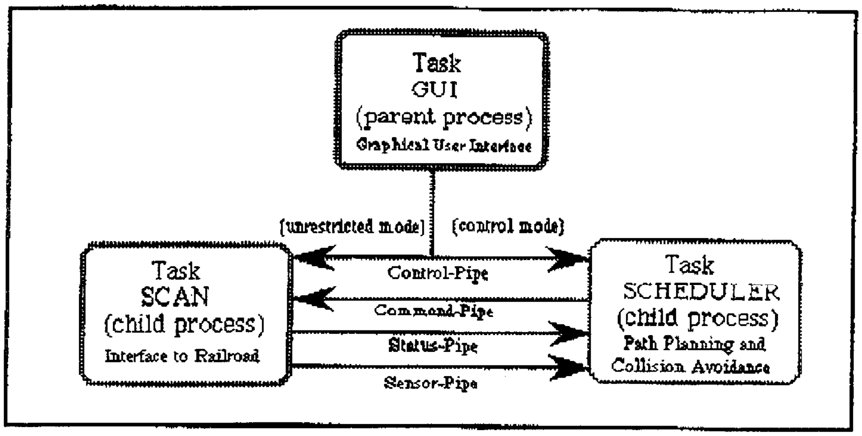


Figure 3. Task Map and Interprocess Communication Piping.

Figure 4. Graphic User Interface (GUI) for Manual Control of the System.

contact sensor information through the Track Detection
Module (TDM). The decoded information is then sent to the Scheduler task for proces
pipe. The Scan task performs this by calling the device driver function TRAIN$GET-T
returns the two bytes sent by the Marklin Track Detection Module. The first byte, t
information for the first 8 sensors on the track. Tdm2 contains sensors 9 through 1
returned in tdm1 and tdm2 is accomplished by combining these two bytes into one wor
comparing to 0x8000 in a for loop. A bit is on if the sensor has been tripped. If t
sensors tripped, at least one of the trains has crossed more than two sensors since
the process is executing too slowly to monitor the trains properly or that some har
exception condition arises, the task immediately issues a TRAIN$HALT() command. Thi
system.

The second job of the Scan task is to relay commands to the Marklin Digital Interfa
only task which accesses the RS232 port connecting the SUN SparcStation to the Mark
Scan task will either accept commands from the Graphical Users Interface task (par
Scheduler task. If the GUI task is in control mode, then the Scan task will receive
which gets its commands from the GUI task (user initiated). If the GUI task is in u
will send the user initiated commands directly to the Scan task without any collisi
commands are sent through the pipe interprocess communication facility in Unix as s
contains command code and the lower nibble contains additional information, when re
train speed adjustment for example.

The Scheduler task is responsible for all control of the system. This task intercep
and determines if current conditions on the train layout will allow the command to
causing a collision or derailment). If so, the command is relayed to the Scan task
Pipe, otherwise the command is blocked from the Marklin system. The Scheduler task
information for each train such as: location, speed, direction, and current zone or
tripped, the sensor value is used to index a lookup table which contains the previc
track layout. In this manner it is possible to monitor the trains without addressak
reed contact will signal the fact that a train (a magnet) has crossed the track. Hc
which train crossed, just that some train (with a magnet) has crossed. Thus, trippi
event. Ambiguity can arise due the fact that tripping a contract is not an address
figures out which train it probabalistically is given the monitoring information it
For example, suppose the current sensor read is 8 and the direction is 0. The previ
is compared to the location of each train in the data structure. If a match is foun
the location field for that train. If no match is found the system issues a TRAIN$H
the system shuts down. In this manner the Scheduler always knows where each train i
allowed to lag behind.

The Scheduler task contains the code to detect collisions. When one train approache
Scheduler either issues a slow down command or a TRAIN$STOP(train-address) to the t
how imminent the collision is. The controller does not want to stop a train unless
imminent situations the Scheduler may issue both a train slow down command to the r

speed up to the front locomotive. Upon each train arriving at a switch, the Schedu
switch, and if so, issues the command to the Marklin system.

V. Conclusion.

This paper has described the work-in-progress of a real-time software controller fc
control software does accomplish its objective of moving the digital locomotives a
according to the scheduling algorithm without collision. Using the Unix real-time p
to perform interprocess communication among three concurrently executing tasks, thi
control of multiple digital locomotives each running on the same track layout while
computerized scheduling system to "run" the trains. The control software continuous
to keep track of each train's location and direction, and is constantly performing
The project was initiated to provide an interesting, experimental platform for the
systems for undergraduates in Computer Science and Computer Engineering. While many
Real-Time Systems acquaint students with the fundamental topics in real-time comput
adequate laboratory platforms to exercise the software skills necessary to build ac
modelling and graphics simulations simply do not manifest the frustrating and spasm
physical real-time systems. This laboratory platform requires students to utilize a
mathematics, physics, engineering, computer science, and real-time programming. A v
system running is available from the authors.

Acknowledgement

This work was partially funded by the National Science Foundation under Grant No. U
Millersville University and the Faculty Grants program of Millersville University.
Dr. Joseph Meier for his expertise on model railroading and for helping us get star
undergraduate research students: Bruce Walters, Chris Coble, Jason Eaby, and David
trade mark of SUN Microsystems, Inc., Unix is a trade mark of AT&T Bell Laboratc

References

[1] Webster, Roger and Paul Ross, "A Workstation Laboratory to Improve Undergraduat
National Science Foundation Instrumentation and Laboratory Improvement Program, Gra
University, Millersville, Pennsylvania, USA.

[2] John W. McCormick, "A Model Railroad for Ada and Software Engineering", Commu
1992, Vol. 35, No. 11, pp. 68-70.

[3] Catherall, Thomas, "2-Rail Digital DC", Marklin Digital SIG Newsletter, Vol.
1990, pps 1-8.

[4] Ripps, David L. An Implementation Guide to Real-Time Programming, Englewood C
Yourdan Press, 1982, pps. 23-44.

[5] LaPlante Phillip, A.. The Design and Analysis of Real-Time Systems, IEEE Compu
pps. 63-85.

[6] Catherall, Thomas, "Sending Data From the Train to the Digital Component", Ma
New Berlin, Wisconsin, May 1990, pps 1-10.

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

Roger W. Webster, Ph.D.	**Mary A. Klaus**
webster@cs.millersv.edu	maklaus@cs.millersv.edu
Timothy A. Bish	**Mike Fiorill**
tabish@cs.millersv.edu	fiorill@alpha.millersv.edu

Department of Computer Science
Millersville University
Millersville, PA USA 17551

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.

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 2). 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.

---

*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.

Hardware and Equipment

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 2. 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.

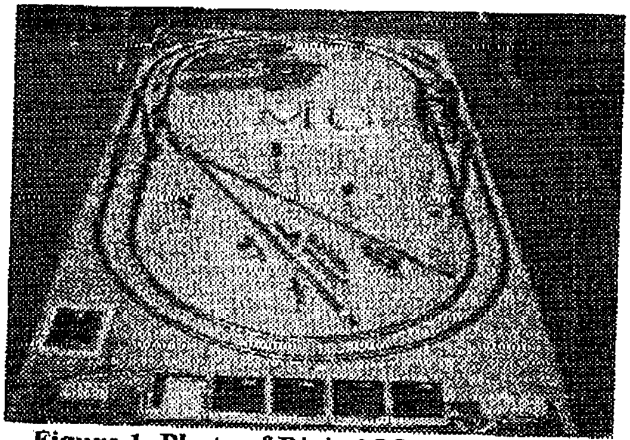


Figure 1. Photo of Digital Model Railroad

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.

Interface to Marklin® Digital Railroad System

The java server sends the low level commands from the computer to the Marklin® Computer Interface hardware via RS232. Methods such and 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 it's 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.

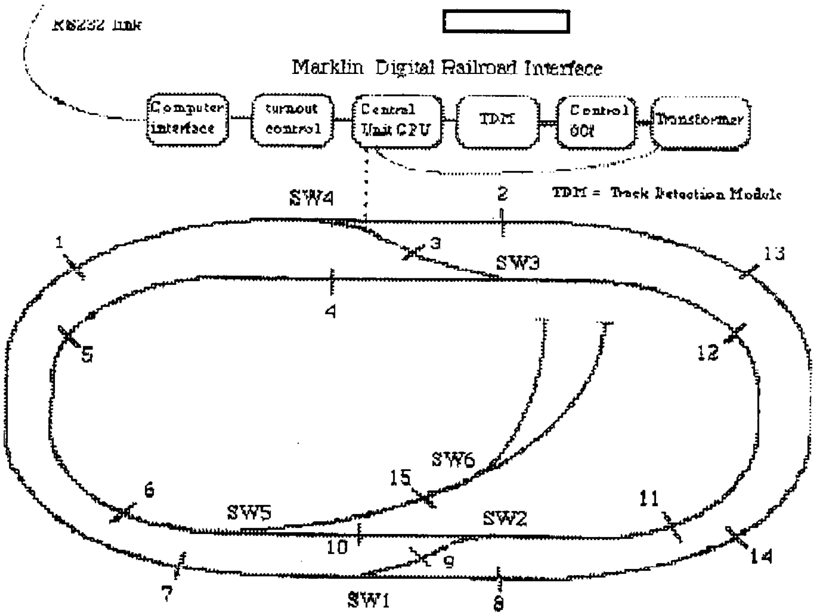


Figure 2. Track Layout. Reed contacts are numbered 1 through 15. Digital turnouts are numbered SW1 through SW6.

Java Client - The User Interface

The java client (see figure 3) 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.

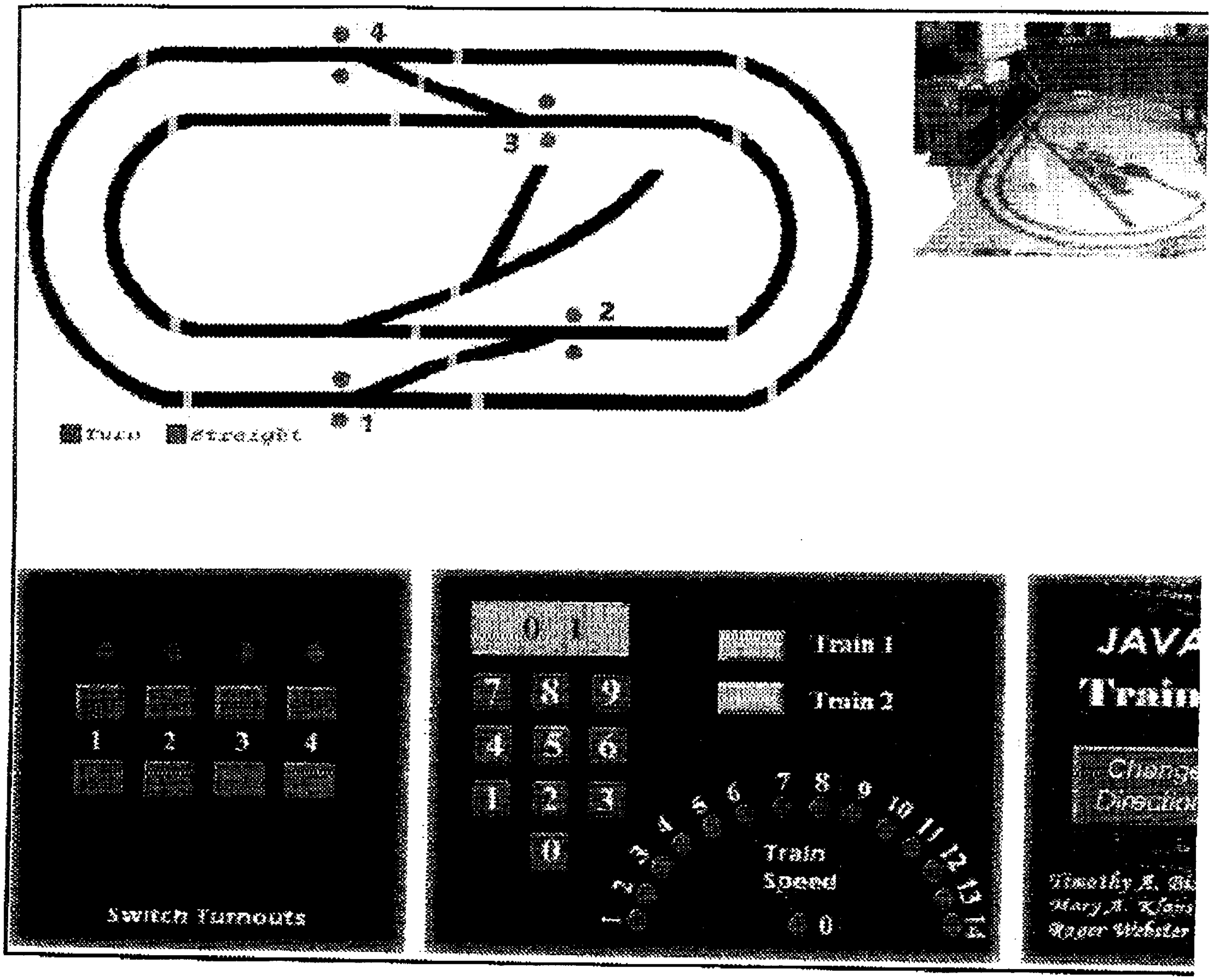
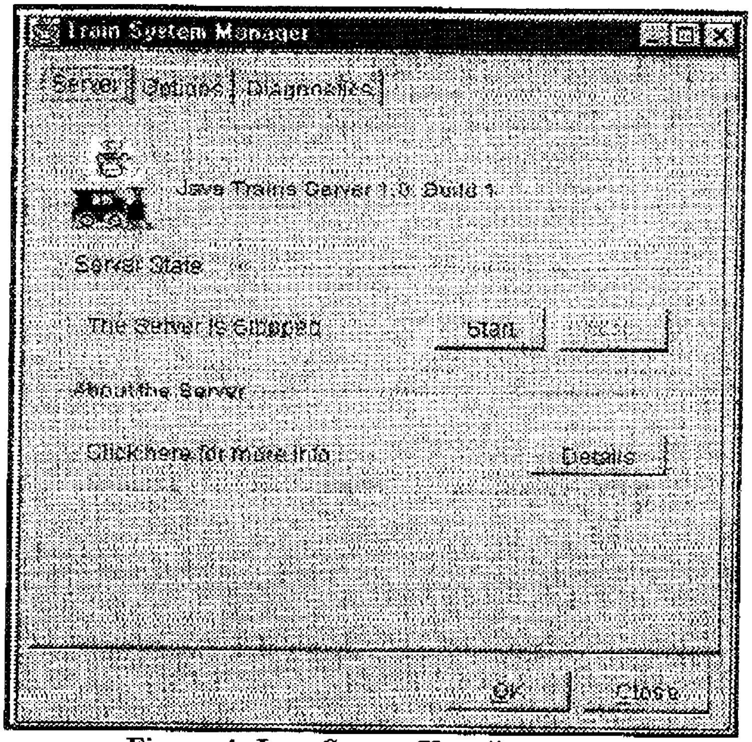


Figure 3. Java Client - The User Interface.

Java Server - The Software Controller

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 Server is Stopped

Click here for more info

Figure 4. Java Server User Interface.

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) or 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 or a TRAINSTOP(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 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.

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.millersv.edu/javatrains/.

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.

References

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

Department of Computer Science
Millersville University

Java Trains Web Site