Real Time Simulation of Rail Dispatcher Operations

by

Santanu Basu

B.Eng., Mechanical Engineering (1997)
McGill University

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

at the

Massachusetts Institute of Technology

June 1999

Signature of Author .....
Department of Mechanical Engineering
May 24, 1999

Certified by .....
Thomas B. Sheridan
Ford Professor of Engineering and Applied Psychology
Thesis Supervisor

Accepted by .....
Ain A. Sonin
Chairman, Department Committee on Graduate Students

Real Time Simulation of Rail Dispatcher Operations

by

Santanu Basu

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Abstract

Cognitive task analyses (CTAs) have been conducted to better understand the thoughts and actions of modern railroad dispatchers. These have provided information necessary to any investigation of dispatching operations, especially the study of the impact of new communications technology on the dispatching environment.

This project was conceived to provide an empirical test-bed for such technologies, in the form of a real time human-in-the-loop simulator. The report elaborates on the CTA background information, the design motivation, capabilities of the simulator, and it's software and hardware architecture. It was written with two goals in mind. First and foremost, it was written to provide instruction for both the participants and the conductors of any experiment based on the simulator. Second, it will help in implementing further functionality through code extension, by providing details of the underlying software.

Thesis Supervisor: Thomas B. Sheridan
Title: Ford Professor of Engineering and Applied Psychology

Acknowledgements

I would like to thank the following people for their help and guidance over the course of this project:

Professor Thomas Sheridan, for providing valuable guidance and help whenever it was asked for. Dr. Jordan Multer for showing me the ins and outs of working at the Volpe Center, and being a great project leader to work with. J.K. Pollard for his invaluable help with gadgets, and answering questions I didn't even think to ask. Steve Jones, for being so helpful in providing us with participants for our experiments. All the dispatchers who very sportingly sat through hours of dispatching runs and then provided invaluable comments and suggestions. Nicolas Malsch, for executing his half of the project so meticulously. Nicolas Oriol, for going above and beyond the call of duty by doing late night test sessions on the simulator with us, not to mention helping us out with our actual experiments. Kari Kulaszewicz for being helpful in general, and in particular for lending us her pickup truck to transport vital equipment to Volpe. Sarah Meischer, for frequently offering to help us out on our project even though she was in no way assigned to it. Jay Einhorn and Ed Lanzilotta, train simulator gurus, for getting me up to speed with prior work. Drew Kendra and Frank Sheelan, for providing advice and suggestions about the simulator.

Table of Contents

1 Introduction

New technologies are being examined by major North American railroads, in the hope that they will increase efficiency, safety, and facilitate higher volume, higher speed travel. The impetus for seeking these improvements was provided by the deregulation of the railroad industry in the early eighties [9]. The resulting market forces, both within the industry, and relative to other transport industries, made the existing operational patterns unprofitable.

While the railroads understand that there is a need to adopt new information technologies (often referred to as data-link technologies), there is an understandable reluctance to deploy them without extensive research and development and prototype testing. Many such pilot projects have been carried out, and in addition to producing valuable scientific results, these have repeatedly shown that deploying a prototype is an expensive proposition [10] [8].

My thesis seeks to demonstrate that simulation of data-link technologies can be used to study them in a more cost-effective manner. It does not propose to replace prototyping with simulation, only to supplement it. The central task in this project is the construction of a simulator that models the basic functionality of computer aided dispatching, based on Amtrak's CETC control center [1]. The simulator is then studied to determine if it can yield information that will help evaluate a sample data-link system, on which was developed specifically for this project. While the data-link system is described and investigated, the development and validation of the simulator is the primary focus of the work. A complementary project by one of my colleagues focuses on the data-link technology itself.

2 Motivation

2.1 Recent Trends in the Railroad Industry

In 1980, the Staggers Rail Act initiated the deregulation of the railroad industry [9]. This came amid a widespread trend toward deregulation of all sectors of the transport industry, rail, truck, and air. Within a few years, several restrictions on railroad operation were relaxed. For instance, greater freedom was allowed in "piggybacking" traffic, encouraging a migration of traffic from higher priced (generally lower traffic) routes to lower priced ones (generally higher traffic). In the high volume northeast, the Northeast Rail Service Act was passed, which made it much easier to abandon lightly used track, and shift the traffic to more heavily traveled routes. Without an entity to oversee the economic framework in which the various railroads operated, market forces began to reshape the face of rail transportation.

Another form of deregulation addressed pricing schemes. Prior to the early eighties, the government had set, or at least placed limits on transport rates, for both freight and passengers. Beginning in 1980, these restrictions were greatly relaxed, essentially allowing free market forces to dominate the behavior of the railroad companies. In contrast to the previous system, each railroad now had to compete against the rest. Geographic separation did not segment the market because in many cases redundant routes belonging to different railroads connected identical regions. And since traffic management was now deregulated, the only way to retain previous traffic levels was to provide competitive pricing.

The economics of operation under the new rules was fairly simple in a sense. If a railroad was handling ten units of traffic, it would be much more profitable to route that traffic over one hundred track-miles than over two hundred. Years of regulated operation had caused railroads to build track networks that stressed service and geographic coverage at the expense of efficiency. Now, it became apparent that abandoning lightly

used routes, and consolidating the traffic was an economic necessity. Thus began the shift toward concentrating more traffic on fewer lines.

Increasing traffic, while decreasing total track-miles caused the number of trains per hour, on average, to increase. There are theoretical limits on how much traffic a given network can handle. This is termed the line capacity. It is related to the speed at which trains travel the line, and inversely to the separation between the trains. Railroads never operate at their line capacity; theoretically such operation would make it impossible to recover from any mistake because there would be no room for maneuvering. Operating at above 80% of that capacity is considered undesirable [5].

Therefore, after deregulation, the railroads were only able to shift so much traffic onto high traffic lines before operations suffered. To increase their line capacity, they began investigating higher speed travel, and reduced train separations. These two factors obviously made it more difficult to ensure safety. The train engineers were required to think more quickly, and their reaction times were reduced. Dispatchers were also required to think more quickly. In addition, they handled more traffic than before. To an extent, this was remedied by increasing the number of dispatchers, but this was not a solution that could be extended ad-infinitum.

A secondary effect of increased traffic was the increased maintenance burden. The problem lay not in the need to hire more maintenance personnel, as this was simply a human resources and financial decision. Rather, the problem arose because the percentage of track undergoing work at any given time increased with increased traffic. This raised safety issues because potentially hazardous conditions became more frequent.

Another secondary effect was the increased difficulty of routing trains through territory in which there was a high maintenance presence. A current example of this type of territory is Amtrak's Shoreline route in the northeast. It is undergoing electrification, so that high speed trainsets may run on existing track. The motivation for this work is the need to become more competitive in the Boston-New York corridor with other forms of travel.

This illustrates clearly how post-deregulation economic forces have had trickle down effects on operational characteristics of the railroad.

In an attempt to maintain or improve the levels of efficiency and safety, railroads are turning to novel technologies. In the United States, they are mostly still in experimental or prototype phases, but they show great potential to positively impact the way railroads operate. For the customer, they seek to provide tangible benefits in safety and convenience. For the railroads, the hope is that they will provide economic gains by improving efficiency while maintaining acceptable safety levels.

2.2 Proposed Technologies

The majority of the new technologies being examined involve some type of digital technology to transmit and display information. Sometimes, the term positive train control (PTC) is a term used to describe efforts being undertaken. Others are termed data-link technologies, and have already found a place in the airline industry, while railroads are slightly behind in adopting them. Several railroads in the United States and Canada have experimented with prototype systems. Some of these are presented here to illustrate the concept [10].

Canadian Pacific Railroad (CP Rail) deployed a prototype PTC system on its Calgary-Alberta line from 1993 to 1995. The system concentrated on using digital communications to convey clearances and releases of track segments. The benefit here was the ability to carry out this common operation, which normally consumes radio bandwidth, via digital means. During the test deployment, CP dictated that the migration was to be carried out in steps that were quickly reversible, so that revenue service was not interrupted by potential failures of the experimental equipment. This methodology illustrates the difficulty in testing new technology in a real world scenario. Ultimately, the cost of maintaining the prototype became excessive and the project was terminated. This illustrates another drawback of real world deployment that will be used to make a case for simulators, namely that prototyping any new technology is an expensive task.

Another well known foray in data-link technologies was the Burlington Northern (BN) Advanced Railroad Electronics System (ARES). It was developed beginning in 1984 in conjunction with Rockwell International, and was quite advanced in the level of information it provided beyond what was (and mostly still is) considered standard. It gave dispatchers a means of viewing the actual positions of trains, not just the discretized block positions. Onboard sensors relayed information to the dispatcher about safety violations, and dispatcher-side programs checked for common mistakes in track clearance and release before relaying the information to the trains. Locomotive crews benefited from global positioning information, and information about conditions on upcoming routes. In the event that the crew was unable to control the locomotive, braking could be applied remotely. Maintenance of way crews were also given global positioning information, and supplied with devices that allowed them to communicate with the dispatcher digitally, thereby freeing radio bandwidth. ARES was deployed from 1986 to 1990 and proved the viability of many of the technologies. However, the project was terminated due to financial reasons in 1992.

2.3 Obstacles

The historical cases described in the previous section illustrate one of the main difficulties in introducing a new technology to an established industry. In the short term the technology always incurs an economic loss for the company, and may also cause operational inconvenience if it is widely deployed. Although safety is of great importance to railroads, the reality is that the cost-benefit analysis is the deciding issue. A prototype project will be terminated, or not even initiated, if the economic planners perceive that the short-term cost is high, even if moderate long term gains are likely. Thus, to better the chances that a new technology will be adopted, the short term economic loss must be minimized. But what makes prototype deployment so expensive?

To begin with, extensive impact assessments are necessary to predict how the test deployment will affect the existing revenue service. These take at least months,

sometimes years to complete, and they extend the period during which the costs are positive and the benefits are zero. In some cases, the impact assessment will determine that it is not possible to deploy a prototype because it will interfere with existing operation; the project may be terminated at this point, without having provided any tangible benefit.

If a prototype is deployed, the cost of construction and integration is substantial. It is often increased by the stipulation that any changes must be quickly reversible. Once built, the new equipment requires maintenance, and the maintenance personnel require training. The proposed users of the new technology also need training, and all this costs money. Finally, in the event that the prototype is not adopted, there is the additional cost of removing it from the testbed, which incurs additional cost.

Apart from the cost-benefit analysis, there is the issue of time. In the BN example the time from project initiation to termination was eight years. And this was a case where the technology was ultimately not adopted. Undoubtedly, if it had been deployed across BN's territory, additional studies would have had to be conducted relating to scalability, maintenance and robustness.

2.4 The Case for Simulation

To alleviate some of the problems of prototype deployment, this project proposes using real time simulation of dispatcher operations. Even for very simple data-link technology, simulation is cheaper and less resource intensive than real world deployment. Different types of data-link systems can be tested using essentially the same hardware, which consists of little more than desktop workstations. Obviously a simulator cannot replace prototyping after a certain point in the R&D cycle, but it can reduce development time and cost in the early phases.

There are some potential pitfalls in using simulators. It is important to make the simulator sufficiently realistic to study the technology. In human-in-the-loop simulators,

the participants have to react, to some degree, in the same way they would react to a fully deployed system. Additionally, the simulator must be carefully designed so that it can provide a good deal of the same information that a prototype system would provide. In the course of this project a rail dispatch operations simulator will be constructed and these issues will be addressed.

3 Objectives

Having described the need for a rail dispatch operations simulator, the objectives of this project can now be laid out. The first objective is to design a simulator to study a novel communications system, in the context of rail dispatch operations. The issues that will be investigated are: how should the simulator collect data, and what kind of data should it collect? Does the data provide useful information that could not be otherwise obtained? How realistic is the simulator? How flexible is the simulator architecture?

The second objective is to study the use of two simple data-link systems by experienced dispatchers, in order to validate the usefulness of this simulator. While the data-link systems will be studied both qualitatively and quantitatively, the primary focus is not to draw conclusions about them. Rather, it is to demonstrate that the simulation model adopted in this project is a suitable one for carrying out such a study. This is by no means clear, as it is very difficult to study such things as intent, cause and effect, and human reaction based only on recorded actions. The complementary report to this one, written by my collaborator Nicolas Malsch, focuses on the data-link systems themselves, and how they compare to standard radio communications.

4 Design

There are two primary design issues addressed in this project – design of the simulator, and design of the data-link system we wish to test. The simulator design mainly addresses the issues that are specific to a simulated, as opposed to real world, rail dispatching system. These are of interest to the people conducting the experiment, or extending the simulator. The data-link design address usability, interface, information flow, and other issues that would appear in a real world implementation of the system. These issues have a direct impact on the user of the system, or in this case, the participant in the simulation.

4.1 Simulator Design

4.1.1 Real World Model

The simulator interface, not including the data-link system, was based on Amtrak's dispatch operations center (called CETC) in Boston South Station [1]. The heart of CETC is a control room where approximately 12 dispatchers are together controlling a large region of track extending from Boston to New Haven. Each dispatcher operates a workstation consisting of multiple computers connected to a central network that maintains state information about various elements of the railroad. Each one is responsible for a portion of the track, called a territory. Each can also refer to a map of the entire track network that is displayed across the wall in the front of the control room.

The dispatcher workstation is the operational unit that forms the basis for the simulator. In CETC, it consists of one or two computers that display the track comprising the territory of the given dispatcher. The monitors used there are touch sensitive, so many interactive sequences are done by touching appropriate regions of the screen. A third screen may display other information, such as train consist information, travel restrictions etc. Each workstation is also equipped with a foot pedal operated "walkie-talkie" which

can be used to contact other railroad employees, such as engineers of trains, or maintenance of way workers. Additional desk space is used for so-called "cheat sheets", which contain condensed information about train schedules, assigned work, and other information that is not quickly accessible through the computer system. These paper backup systems sometimes become necessary in the event that the computer control system breaks down; they allow continued, uninterrupted operation.

4.1.2 Interface Design Rationale

While the conventional (non-data-link) elements of the simulator are based on CETC, the intent was never to duplicate that system in every detail. As with most systems that have large feature sets, the majority of the operator's time is spent using only a handful of them. This was observed during the course of several visits to the CETC control room. An attempt was made to identify standard operations that were involved in the bulk of the dispatcher's workload.

The actions required to execute these operations in the simulator were designed to be similar to the corresponding actions at CETC. The goal was to minimize the differences between the simulated operations and the real ones, apart from the variable communication medium. In this way, variable dispatcher performance could be correlated to differences between data-link and radio, rather than to differences in the standard operative procedures. The operations chosen for inclusion in the simulator were setting and resetting routes, and creating and removing protected zones.

Setting a route refers to the act of marking a portion of track as "reserved" for travel. The next train that approaches this track will be allowed to pass through, and then the track will revert to its default state. This is the primary means of traffic control available to CETC dispatchers. Setting and resetting a route may at first seem to be a quick task that does not consume much of the dispatcher's time. Indeed, the act of setting the route may take only ten seconds. However, the more important factor to consider is the amount of time the dispatcher thinks about the operation before he actually executes it. This

timespan was observed to be substantial in some cases. Moreover, setting a route is one of the most frequently performed tasks. It forms a kind of background activity, upon which the communications load (be it radio or data-link) is superimposed. Finally, the way in which routes are set impacts almost every other aspect of dispatcher operations, from traffic congestion, to potential hazards, to work assignments. Route setting was therefore deemed to be an essential activity, and was selected as one of the operations implemented in the simulator. Route resetting (i.e. removing a route) was included for completeness, even though it is not a very frequent operation.

Protecting a portion of track is a means by which the dispatcher can provide a safe working zone to maintenance of way personnel. Trains are routed around protected zones. Workers ask for protection and are granted or refused it based on traffic patterns, the length of time protection is requested, and other criteria. In a similar manner to setting routes, creating protections does not take very long to accomplish, but the cognitive process prior to the action is rather complex. Several sources of information are consulted, namely the dispatcher's memory, train schedules, state of traffic at the moment, and expected state of traffic in the future.

4.1.3 Human-in-the-Loop, Experimenter-in-the-Loop Design

The suffix "in-the-loop", when applied to a simulator, means that the interaction from and feedback to the experimental participant is real time. Of course, this is a human-in-the-loop simulator. In this experiment the participants were real dispatchers from CETC who interacted with the simulator for two shifts of one hour each. This setup is further detailed in the experimental methodology and procedures section.

Less obviously, this simulator was also designed as an experimenter-in-the-loop system. This means that the experimenter has the ability to dynamically influence the evolution of the simulation in real time. The necessity of this simulation model becomes apparent when one considers the variety and complexity of the stimuli that a dispatcher handles on the job. He directly interacts with multiple humans, and groups of humans. He indirectly

interacts with machinery via his workstation and requests issued to human operators of machinery. The other human participants in the railroad interact with each other, and their actions impact the dispatcher.

In more common simulators (such as driving, or flight) it is possible for the experimenter to act only as an observer, because the participant interacts with a limited number of non-human elements. This is not possible in a rail dispatch operations simulator. The state of the art in artificial intelligence still cannot simulate dynamic, intelligent human responses to interactive sequences (it is debatable whether this will ever be possible). The only way to simulate this is for one or more people, in this case the experimenter(s), to handle the portions of the simulation that model person-to-person interaction. This necessity motivates an experimenter-in-the-loop architecture.

There are restrictions placed on the ability of the experimenter to influence the simulation. These restrictions are imposed to make it impossible for the participant to determine which stimuli are the result of computer simulation and which are the result of experimenter actions. Specifically, the experimenter cannot create any information that could not be potentially created by an element in the simulation (a train engineer, track worker, etc.). The experimenter cannot influence any object within the simulation in a manner that is clearly unnatural. Any communications initiated by the experimenter must be attributed to an existing element of the simulation. These restrictions strive to prevent situations where the experimenter subconsciously biases the outcome of the simulation to support preconceived expectations. At the same time, they allow the flexibility to handle unexpected events, and to design scenarios that exhibit more sophisticated communications sequences than would be possible with pre-programmed behavior alone.

4.1.4 Timing Characteristics

The simulator was designed to operate in "psuedo-real-time." This term is chosen because, while a second in the simulation is a second in the real world, a simulation run lasting two hours in the real world may only encompass an hour and a half of simulated

time. This is possible because the simulator can be stopped for discrete blocks of time, at which point all simulated elements save their state. When the simulation is again resumed, the state continues evolving as if no time had passed. It was felt that this ability was necessary to allow the experimenter to question the participant during the simulation, about key issues, in a qualitative, discussion oriented manner. It is impossible to do this without distracting the dispatcher from the tasks at hand if the simulator can not be halted and resumed on command.

4.1.5 Data Gathering Design

The simulator provides it's own detailed data outputs. It was assumed that video and audio recording schemes would be insufficient to capture the necessary information from the data-link system, which depends heavily on text (see the data-link design section). The design of the simulator's data recording system features real time recording and minimal post processing. Data is not held until the end of a simulation run, but recorded to disk as it becomes available. The amount of post processing is also reduced to a manageable level. The size of the data traces make post processing time-consuming. Automated post processing is also difficult due to the sometimes qualitative, and highly variable nature of the data being recorded.

4.1.6 Designing for Flexibility

From the outset, the one of the design goals was to create a simulator that used data files extensively, providing flexibility without requiring programming. Wherever possible, data has been removed from the software, and shifted to a datafile that is modifiable according to fixed rules. Most of the characteristics of a scenario can be altered by making the appropriate change in those files. Scenarios are discussed in section 6.2.

The datafiles are one means of insuring flexibility. They allow modification of the objects within the simulation. The other type of flexibility is called extensibility, essentially the ability to change the way the objects are viewed and manipulated. This

requires programming new software code, but every attempt has been made to structure the simulator such that the amount of programming required is reduced.

4.2 Data-link Design

A basic data-link system was created for this project. This report focuses on studying the simulator's ability to study the data-link system as it is compared to standard radio communication. For completeness, a discussion of the data-link system design is included here. A more in depth coverage of this topic is presented in the complementary report by Malsch.

Before settling upon one design, a task analysis of CETC dispatchers was carried out to determine what kinds of activities were most common, most time consuming, easiest, most difficult, and so on. Then, a set of features was settled upon and designed into the data-link system of the simulator.

4.2.1 Dispatcher Task Analysis

Through observation of dispatchers on the job, through interviews, and through simple discussion, several key activities were identified. They can be roughly categorized as train management, maintenance work management, and information relay management [7].

Train management includes operations that impact the normal motion of trains through the territory of a dispatcher. This consists primarily of routing the train in the proper way through the territory, managing meets and passes, and prioritizing traffic in high traffic areas. In some cases it also includes short-term speed restrictions, travel bulletins, and other information relayed to train engineers. A significant amount of time is spent thinking about these actions because improperly routed trains will potentially encounter delays. Dispatchers were observed carrying out these actions at CETC. In most cases they study the current position and target position of a train, and try to map out a route to

that target, subject to the restrictions of work in progress, current traffic, predicted future traffic, and speed limit considerations. They tend to route a train as far as they can because they feel it reduces the likelihood they will mistakenly leave a train waiting at a signal. It appears that, in general, they will delay one train ten minutes, rather that two trains five minutes each. In all cases, they try very hard to avoid delays. In conversations with dispatchers, small delays, or none at all are one of the most important indicators of a successful shift. The importance attached to proper routing is a key consideration in designing a data-link system because it is possible that the routing may take precedence in the mind of the dispatcher. Therefore the time spent using data-link must be minimized. The total time spent on train management was observed to be much greater than the time spent simply on executing routing actions. The majority of the time was spent thinking about what action to take; little time was actually spent on executing that action.

Maintenance work management involves scheduling and protecting workers who are located on or near operational track. The scheduling task begins when a maintenance of way (MOW) crew contacts the dispatcher and asks for permission to work. The dispatcher has to make a decision based on the current traffic flow, length of work time requested, and potential future traffic flow. If the permission is granted, the dispatcher sets up a protected zone on the track where the work will take place. Trains can no longer be routed through this zone, which raises the possibility of delays. The work crew is contacted and notified of the dispatcher's decision. It was observed that the dispatchers do not like to give work permissions for timespans that are too far in the future. Instead they tend to ask the work crew to call back later. They also tend to assign higher priority to train movement than to work requests. When granting permission to work, there is a substantial back and forth communication between the MOW crew and the dispatcher. The crew states the locations and time of work. The dispatcher may misunderstand or fail to hear part of this information and ask for it to be repeated. When he is satisfied he has heard it properly he will make the decision to grant or refuse. If granting, he will repeat the locations and times to the MOW crew, which must confirm them. The dispatcher is also required to write down the information in standardized

forms. The whole process can take several minutes in total, during which other activities may be performed. It was hypothesized that this was an area in which a point to point data-link system would be advantageous.

Information relay management refers to the role that the dispatcher plays as a central point of information exchange. For example, consider a train travelling through a dispatcher's territory, where it encounters an exceptional condition. Perhaps there is potentially hazardous condition requiring reduced speed. The train engineer knows that this information will be of interest to other engineers, but he does not know which engineers will be passing through the same territory area in the future. By relaying this information to the dispatcher, he ensures that that it will be provided to other train as they pass through. The dispatcher may also take some action to remedy the situation, such as contacting a MOW crew. Because the dispatcher is the only member of the railroad operation that has a fixed location, he tends to handle situations that have a fixed location. Little time is consumed in relaying information. The difficulty lies in remembering who to relay the information to, when to relay it, and what to relay. A data-link system might be helpful in supporting this kind of information relay management.

4.2.2 System Design

Now that a group of key tasks has been identified, the design of the data-link system can be described. It's characteristics are: text-based, persistent, point to point or multicast, and non-conversational. These characteristics were chosen based on the tasks mentioned above. The meaning of each is described, and the reason for choosing it is explained.

Text based means that all information is created and transmitted in the form of text messages. These messages can refer to images on the display screens, but these visual objects are not themselves embedded in the messages. Therefore there is no facility for transmitting live video, or pictures. As an initial trial system, this seems reasonable because text has more or less the same descriptive power as the spoken word, which is the current communications standard. Creation of text messages is not accomplished by

typing in general, but rather by filling in blanks in pre-formatted messages. This method was clearly favorable to free-form text entry because several dispatchers said that they would not use a system that required them to type even a modest amount of text. This interface is described further in section 5.

Persistent describes the lifetime of a message. In radio communications, this is the amount of time it takes to speak the message. Since it is not recorded, it does not exist after that; the dispatcher cannot go back and review the message, apart from the information that is stored in his memory. This is apparently insufficient, because dispatchers were observed writing down information based on radio communications. Therefore, it was hypothesized that a persistent communications system would be of use to the dispatcher. The data-link was designed such that all incoming and outgoing messages are recorded and remain available on the screen for future reference.

Point-to-point and *multicast* are terms that refer to the way a message is delivered, and who receives it. As a baseline, radio communications are considered *broadcast*. This means everyone receives every message. This is a source of two problems. Dispatchers (and everyone else) receive messages that are not addressed to them, and they have to ignore them. From observation, it seems that they have become accustomed to this and tolerate it reasonable well. More importantly, when two parties are communicating on a broadcast system, no other communication is possible. In practice, this is not always true due to transmitter range, and it is true that independent communication can occur on different channels. Nevertheless, the effective bandwidth is low, and dispatchers routinely complain about this. At the opposite end of the spectrum is point-to-point communications, where the sender of a message chooses a single target, and no one else hears the conversation. Standard telephone conversations are point to point communication. The advantage of this type of system is that multiple communications between many parties can be continuing simultaneously. The disadvantage is that if ten different people need to know of a situation, the dispatcher has to create ten different messages. In between the two extremes is multicast communications. In this case, a primary recipient is chosen, and other secondary recipients can be specified, similar to

the "cc" field in email messages. Multicast can collapse back to point-to-point at the discretion of the sender, simply by not specifying secondary recipients. The simulator provides both point to point and multicast communications.

The last characteristic of the data-link system is a result of the first three features. *Non-conversational* means that if party A sends a message to party B, B is not required to respond immediately. In fact, if A send two messages to B, B may chose to reply to the second message first, and the first second. This contrasts with a conversational approach, where each communications sequence generally finishes before the next begins. Interrupting one sequence to start another is not the norm, and when this is done, some kind of notification is supplied to the party being interrupted. E-mail systems are non-conversational. Radio communications are conversational systems. It turns out the benefit of conversational systems, namely instant feedback, is also its drawback. This becomes clear in situations where there is not sufficient information to complete a communications sequence. Usually, this is handled by terminating the conversation, and restarting it later. This creates a situation where contextual information needs to be repeated when the conversation is resumed. It is not clear how readily dispatchers will adapt to a non-conversational system, but it will be interesting to note their reactions. In any case, a persistent, text based system is inherently non-conversational. One of the goals of the study will be to observe how successfully the simulator can compare a non-conversational system to a conversational one.

5 Implementation

Now that the design rationale of the simulator has been laid out, the software implementation can be outlined. The goal is to show how the features described in the previous section are represented in the simulator, to describe the interface, and to explain how to use them. On the highest level, the simulator is broken into two parts: the dispatcher station, which provides the human-in-the-loop functionality, and the experimenter station, which provides the experimenter-in-the-loop functionality. Each of these stations is further divided into one or two display terminals, and one messaging terminal, each of which corresponds physically to a computer. Each component is described in detail in the following sections.

5.1 Dispatcher Station: Display Terminal(s)

Each display terminal will display roughly half the territory for which the dispatcher is responsible. Figure 5.1 shows a screen snapshot of a terminal displaying the track for one of the scenarios used in our thesis project.

The majority of the screen space is occupied by the dark background on which is drawn the track and other information that can be displayed visually. Any object required to complete a task is "selectable" which means that it can be accessed by clicking on it on the screen. The rest of the lower portion of the screen is used to display buttons that allow the dispatcher to access functionality that is not handled by the messaging screen.

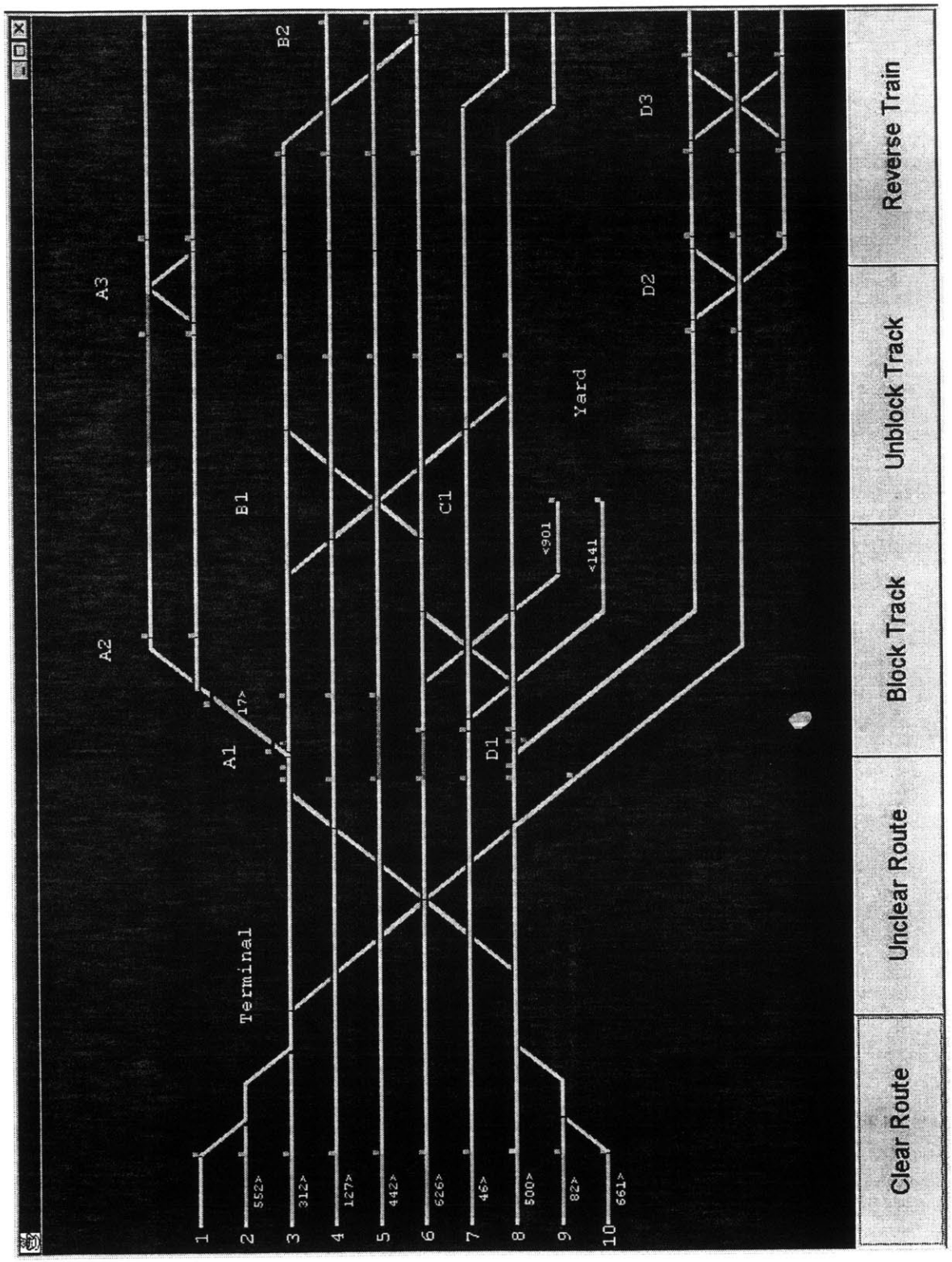


Fig. 5.1 Dispatcher Display Terminal

5.1.1 Track

Track is normally in white, as is most of the track displayed in the figure. This default state indicates that the track is not occupied, no work is being done, and it has not been included in a cleared route. A blue color indicates that the track has been blocked for work. A light green color indicates that a route has been cleared through this track and (presumably) a train will be passing through in the future. A yellow color indicates that the track is being highlighted (see the discussion of the message window for a description of highlighting). A red color indicates a train, or a portion of one occupies the track. When more than one state is in effect (such as a routed track that has become occupied), the ordering is white, green, blue, red, yellow. Thus in the event red and green conflict, red will be shown, not green.

Tracks are divided into blocks. The boundaries between these blocks are not always clear; in fact, it is relatively easy to do this, but it is not done to maintain similarity with the South Station dispatcher display. The blocks are arranged on an invisible grid, such that every block must begin and end on a grid vertex. A block may span more than one grid unit. However, blocks can only be oriented horizontally, or at 45-degree angles, again keeping in line with the South Station displays. As far as the screen display goes, blocks are atomic elements. This means the block cannot be half red and half green. Clicking on any portion of the block is equivalent, and will select the whole block.

Blocks meet at grid vertices. Up to six blocks can meet at one such point. While the blocks themselves only change color, these joints can change shape as well. The shape is altered depending on which two blocks are "connected." The joints are simply the visual representation of switches. See below for a discussion on how to affect the states of these switches.

5.1.2 Interlockings

Interlockings are not really objects in their own right, but special collections of track. Essentially, an interlocking is a group of branching tracks flanked by non-branching

track. Each interlocking is given a name, which is displayed in yellow above the tracks that comprise that interlocking.

5.1.3 Stations

Like interlockings, stations are just special collections of track. They are labeled in yellow as well. An unenforced convention that is used currently is to name interlockings with a letter-number pair, like A5. Stations are given names like Station B, or Terminal. Trains may or may not stop at stations, depending on their schedules. This is discussed further in the sub-section on trains.

5.1.4 Poke Points

Frequently, usually near grid vertices, there are small red squares displayed. These are called poke points. They do not correspond directly to a physical object like a block does, but they are used to clear and unclear routes. To clear a route (that is, to allow travel through it) two poke points are selected, and the track between them is cleared; it turns green to indicate this. The poke points themselves also turn green (from their default color of red). The user may encounter a situation where he has selected two poke points, but after a second or two no route has been cleared. This means that the poke point combination was not a valid pair. Poke points must be on opposite ends of an interlocking, or on opposing ends of two different interlockings, such that the route joining them is "between" the poke points, in the topology of the track. In the event that a route spans more than one interlocking, the poke points along the route will turn green. To unclear a route that has previously been cleared, the dispatcher must simply click one of the poke points that was used to clear the route. In multi interlocking routes, even though poke points long the route will be green, only the end poke points can be used to unclear the route.

Any track not in a cleared route is considered uncleared. Such track permits travel up to the next poke point facing the direction of travel. A poke point's direction can be determined by taking the direction of travel required to arrive at the closest interlocking

from the poke point in question. This implies that poke points on opposite ends of an interlocking are opposite in direction. As a general rule, trains cannot travel through a poke point in the direction the poke point faces unless that poke point is green. This is the primary method of controlling traffic.

5.1.5 Trains

Trains are displayed as numbers, accompanied by a direction arrow, and potentially a + or a – symbol to indicate delays, or ahead of schedule operation. This text is displayed in yellow. The number is the unique identification number of the train. The train will also cause the track on which it sits to become red. Since a train may occupy more than one block, several blocks may be red. However, the train's identification number (ID) and direction are displayed only on the block where the front of the train is located. This text may appear above or below the track. Essentially, if one orients ones view along the track in the direction of motion of the train, the descriptive text will appear on the right of the track. To select a train, click on the image of the ID number, not of the track where the train is located.

Trains will traverse the track according to a few simple rules, some of which have already been mentioned. Trains will not pass through an uncleared poke point in the direction of their travel. They will not pass through another train, unless they are unable to stop in time. It is interesting to note that this is even possible; why should trains not be able to stop in time? The reason lies in the fact that a two aspect system of speed limiting is used in this simulation. Situations can arise where very short blocks result in inadequate stopping distances, and the aspect system does not provIDe enough advance warning. Of course, both the aspect system, and the track lengths can be customized to prevent these situations from arising (as described in the architecture section). In the rare event that two trains cannot stop in time to avoid a collision, the simulator is programmed to cause one train to "die." This means the train will stop moving but will continue to occupy the blocks occupied at the time of collision, rendering them non-traversable.

Trains will stop at stations if the station is listed on the train's schedule, as specified in the data files with .trn extensions. If a train arrives at a station ahead of time, it will wait there until it's scheduled departure time. If it arrives late, it will wait for a minimum stop time, and then continue moving.

Trains normally move in one direction, but may reverse their direction in stations. In exceptional circumstances, the dispatcher may manually tell a train to reverse direction, in which case it will slow to a halt, reverse direction, and continue moving. This functionality is accessed through the reverse button, described below.

5.1.6 Operations

Below the section of the screen used to display track, trains, etc. is a smaller section containing various buttons. Each of these buttons will launch an *operation*. Currently there are five tasks that can be accomplished via these buttons: Clear a route, Unclear a route, Block track, Unblock track, and Reverse train. Interaction with the simulation using these buttons follows a click-select-execute pattern. For example, to clear a route, the dispatcher will click on the "clear route" button, select two valid poke points, and the simulator will then execute the clear route operation, resulting in the route turning green. If the dispatcher then wants to clear another route, he will again click the "clear route' button and go through the same procedure. Clicking a button does not, therefore, permanently select that operation until another button is clicked. This is somewhat counterintuitive; however, it mimics the interaction pattern of the dispatch center in South Station, upon which this simulator is based. The rationale behind this type of interaction is that the dispatcher has no chance of inadvertently executing an operation because he was unaware it was "in effect". Each time he wants to do something, he must explicitly make that task possible by clicking the button. This provides some degree of safety.

The usage of the clear route operation has already been given as an example; the other operations are used as follows. To unclear a route, the "unclear route" button is pressed, a poke point at one end of a cleared route is selected, and the route is cleared. To block a

track, the "block track" button is clicked, a single track block is selected, and the simulator blocks the track, turning it blue. Unblocking works the same way, but the track turns white. To reverse a train, the "reverse train" button is clicked, the train's ID number is selected, and the train will be issued a reverse command. Now that the means of working with these operations is understood, it is necessary to describe what they are used for.

The route clearing operation is the primary traffic management tool available to the dispatcher. If a train approaches an interlocking and the poke points at the point of entrance to that interlocking are red, the train will not be able to pass. In fact, it will start slowing down somewhat ahead of the interlocking. By clearing a route, the dispatcher is indicating to the train that it may travel through the interlocking along the green path and on to the next interlocking. It is important to note a minor difference between the simulator and the South Station system. There, a cleared route would show green all the way to the next interlocking, while in this simulator it is only the track within the interlocking that is green. However, since the portion of track between interlockings is non-branching, the train has implicit permission to travel all the way to the next interlocking.

Route clearing is exclusive. This means that a given block of track, or a given switch cannot be included in more than one route. Thus, routes cannot cross, or be collinear at any point along their length. Attempting to clear a route that would violate this condition will result in the operation being cancelled. Moreover, a route cannot be cleared through track that has been blocked for work (see below). If there is a potential for this kind of route, the simulator will attempt to find the shortest path around the blocked track. If there is no such path, the operation will be ignored. In any event, where there are multiple possible routes between two poke points, the simulator will find the shortest possible path.

Cleared routes are non-directional. A train can enter either end of the cleared route and travel to the other end. Therefore, it is necessary to make sure that two trains are not approaching opposite ends of a route that is to be cleared.

Unclearing a cleared route will simply cause the interlocking to revert to it's default state, which does not allow trains to pass. Unclearing requires only that one of the two poke points used to clear the route is selected to unclear it.

Blocking is used to protect MOW crews at specific locations on the track. Typically, a work crew will contact the dispatcher and ask for work protection before attempting to work at a given location. This communication is accomplished via the messaging console. However, if the dispatcher decides to grant work protection, he must first block the track upon which the work will proceed. He does this using the block track operation. This is essentially a safety measure designed to prevent trains from accidentally travelling on the same stretch of track and endangering the workers. Although the workers will complete their work and notify the dispatcher of this, the blocked track will not automatically unblock itself. This is the responsibility of the dispatcher, and he accomplishes it using the unblock track operation.

The final operation is train reversal. This is done only in exceptional cases, and is not used as a standard traffic management tool. One such situation arises in the current project is a train that is too long to fit on a station platform. This situation requires the train be moved to another platform, and there is no way to do this without reversing the train. Train reversal is not instantaneous. Instead, the train will first brake to a halt. Then, the direction indicator of the train will visually reverse direction, and the train will begin accelerating. Currently, the simulation does not model the time required to change the train's direction (for instance, time required for the engineer or conductor to move to the opposite end of the train). However, the experimenters can insert such a delay by using an operation available only to them called "halt train", which is described in the Experimenter Station section.

5.2 Dispatcher Station: Message Console

In addition to the two monitors displaying the track, the dispatcher has a third monitor on which a message console is displayed. This console is the primary experimental communications system that is being studied in this project. It implements a simple digital messaging system that is used by the dispatcher to quickly communicate standard information to other people or devices in the simulation.

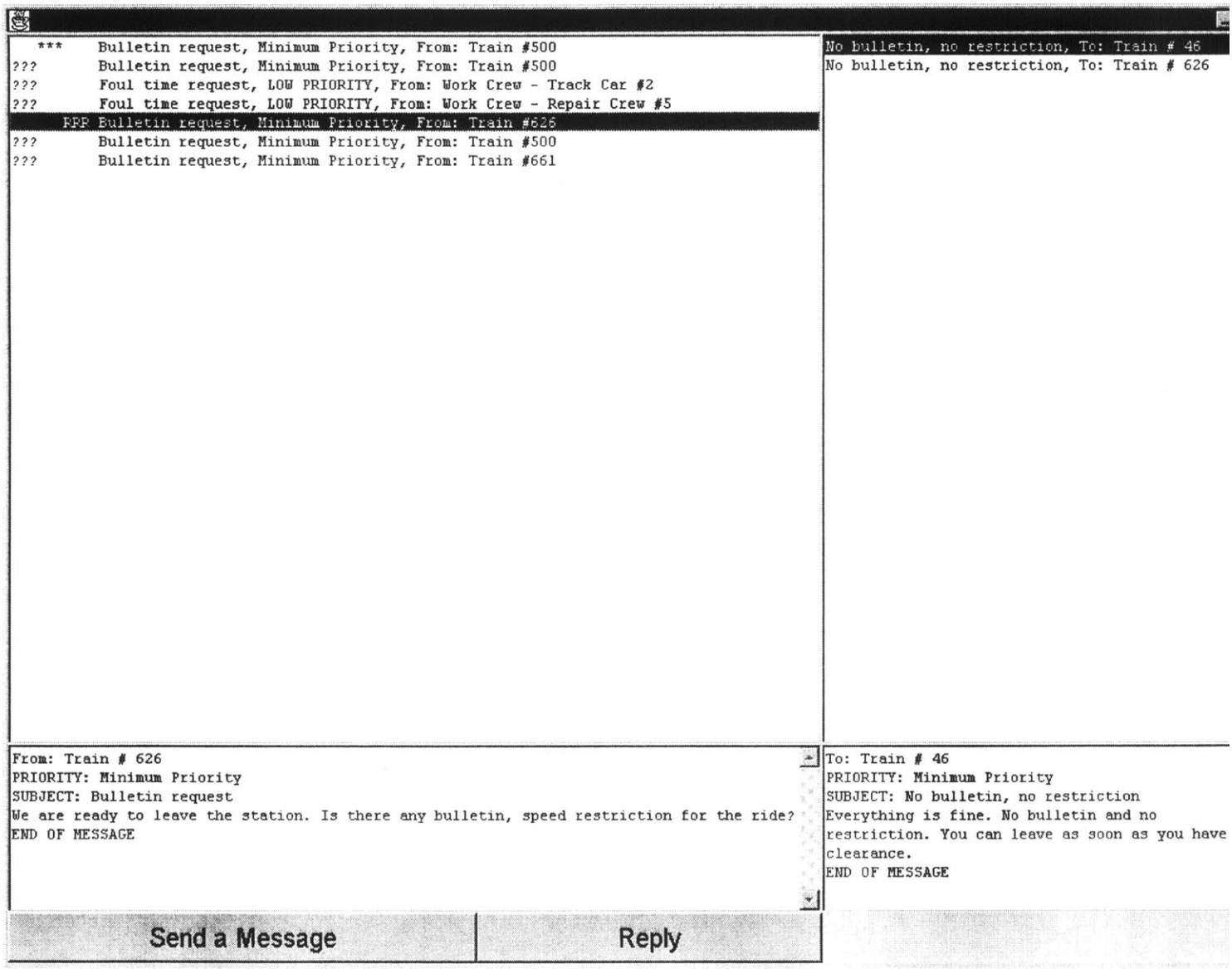


Fig. 5.2 Dispatcher Message Console

5.2.1 Visual Layout

The message console is shown in figure 5.2, after some messages have been received and sent. The majority of the screen space is reserved to displaying text. This area is subdivided into four sections. The two sections on the left are used to display incoming messages, while the two on the right are for outgoing messages. The top sections are used for message summaries (incoming and outgoing) and the bottom sections for detailed message text (incoming and outgoing). Below these four sections is a button bar containing two options. The first is used to send a message, and the second to reply to the currently viewed message.

5.2.2 Incoming Areas

The incoming message summaries section is perhaps the most important of the four, and it occupies the top left quadrant. When a message is addressed to the dispatcher (say a work crew sends a work request message), the first thing that the dispatcher sees is a one line summary of this message in the incoming summary window.

This summary displays critical information about the message as well as markers indicating additional status information. Beginning on the left side of a typical summary line, the dispatcher will see the status marker. This can take on one of three states: not viewed, viewed but not replied to, and replied to. The markers are indented to different depths as well as being constructed from different characters. They are a quick way for the dispatcher to see what needs attending to and what can wait until later. After the status marker, the subject of the message is displayed which provides a rough idea of what kind of information the message will convey. Next is the priority, indicating how urgent the message is relative to others. Finally, the source of the message is displayed (e.g. Train 401).

These one-line summaries will appear in the summary area in the order in which they were received. When there are too many messages to display the summaries in the

summary area all at once, a scrollbar will appear on the right side of this area allowing past message summaries to be viewed.

To see an entire message, the dispatcher simply clicks on the summary line. Upon doing so, the status marker will change from "not viewed" to "viewed but not replied", and the full text of the message will be displayed in the message text area below. The full message text will include all the information found in the summary line, as well as the actual message that is being conveyed.

The message being viewed will often refer to objects that appear in the dispatcher's display windows. For example, a message asking for work permission will refer to track locations on which the work needs to be done. In these cases, the object being referred to will be highlighted using a bright yellow color that is easily distinguishable from the surrounding display. In reality, the only objects that currently possess the highlighting capability are the tracks. Highlighting for other objects is not implemented because it is relatively easy to locate a train, for example, if the train ID appears in the message text. Tracks are highlighted because it is inherently difficult to speak of a specific portion of the track without risking confusion. The yellow highlight will remain for as long as the message is viewed.

5.2.3 Outgoing Areas

The outgoing areas are not used until the dispatcher uses one of the buttons at the bottom of the screen, send and reply. Both of these will result in a message being sent from the dispatcher to some elements in the simulation. Once this is done, summaries of these messages will appear in the top right area exactly as the incoming summaries appear in the top left. However, there will be no status markers preceding the summaries, and instead of the message source, the message destination will be displayed. When one of these summaries is selected, the full message will appear in the bottom right area of the console.

5.2.4 Console Operations

The send and reply buttons at the bottom of the console operate almost identically, except that the latter is used to reply to the currently viewed message, while the former is used to initiate a new message. They both launch a process that allows the dispatcher to select a message template in a methodical manner, and then fill in the blanks of the template to create a complete message. These templates are analogous to paper forms that address specific issues relevant to dispatching operations.

The first part of this process consists of navigating through a hierarchical tree that guides the dispatcher to the desired message. A sample branch of this tree appears in the left side of figure 5.3. For example, to select the train bulletin message, the dispatcher first selects "Low Priority" which launches a sub-tree. Within this, he chooses "Engineer Communications". In the next level he chooses "Bulletin/Speed Restrictions", and in the final level he selects one of the various types of bulletins.

Fig. 5.3 Message Type Selection Tree

At this point, a message form is launched in a separate temporary window that floats above the message console, as seen in figure 5.4. This form is of a standard format regardless of which message is being constructed. This eliminates the need for the dispatcher to figure out how to read and fill in each form, which may lead to more efficient operation (this is one hypothesis that is being tested in this project). Regardless of the message, certain standard fields are always present. These are the recipient, and the sender, although the sender field can be left blank if desired, in which case it will automatically be filled in with the dispatcher's name. Depending on the message, there may be more fields to fill in, as well as the text in the body of the message.

Fig. 5.4 Message Creation Form

There are two ways to fill in the blanks in a message. In all cases, the dispatcher may elect to manually type in the information required in the blank. What exactly this information is becomes clear in the context of the message body. Alternatively, the dispatcher may elect to fill in the blanks by clicking on the appropriate object on the display screens to his left. Of course, there are situations when one method is appropriate and the other is not. For example, it makes sense to fill in a blank describing a location by clicking on a piece of track. Conversely, it is impractical to fill in a blank requiring a time by clicking on a clock. These kinds of fields are filled in via the keyboard. As per standard computing conventions, the dispatcher can move from one blank to the next by pressing the "tab" key. Or, he can click in the field with the mouse cursor. At any point during the form completion, the dispatcher may chose to cancel by pressing the button of the same name in the lower right hand corner of the message form.

Once the required information is completed, one of two actions may be taken. The dispatcher may simply click the send button and the message will be sent to the recipient specified in the form. The form will then disappear, and the main console will again come into focus. Alternatively, the message may be broadcast. In this case, the message will still be sent to the specified recipient. However, an additional window will pop up allowing the dispatcher to select secondary groups of recipients to broadcast the message to (figure 5.5). Examples of such groups are "All trains on a specific branch" or "All workers." This functionality recreates the primary benefit of radio communications that is lacking in digital point-to-point communication, namely, the ability to overhear conversations and extract secondary meaning from them.

Fig. 5.5 Broadcast Selection Window

It is important to be very clear here about exactly what the broadcast button does. First, remember that there is only one human experimental subject in this simulation system at present, and he plays the part of the dispatcher. Thus, broadcasting a message does not mean there is anyone there to hear it. Secondly, broadcasting a message is not really the same thing as overhearing messages in radio communications. In digital broadcasting, the intent lies with the sender. When listening to other people's radio communications, the intent lies with the listener. It is not possible to simulate or measure the intent to listen to a broadcasted message without a secondary human participant, but it is possible to simulate and measure the intent to broadcast a message. Therefore, when the dispatcher presses the broadcast button and selects a broadcast group, the message is not actually sent to members of that group. Only the intent to broadcast to that group is noted, and recorded in a file. Future extensions might alter this, if more than one human subject is involved in a single simulation.

5.3 Experimenter Station: Display Terminal(s)

The purpose of the experimenter station as a whole is primarily to monitor what is happening in the simulation, including the messages the dispatcher is sending and receiving. For the purpose of monitoring the activities of the trains, switches, etc. the experimenter display terminals can view an arbitrary portion of the track network. Most likely, this region will be the same region controlled by the dispatcher, as seen in figure 5.6. With this setup, the experimenter will see exactly what the dispatcher sees. The primary difference is that the experimenter does not have available to him the same set of operations that the dispatcher does. In general he can see but not influence the simulation in the way the dispatcher can. He does possess some ability to affect the course of events, but in the case of such intervention, every attempt is made to portray the results as a natural occurrence as far as the dispatcher is concerned.

Although the experimenter can select objects on his display screen in the same way that the dispatcher can, in practice he only uses that ability to select trains. This is because the only operation currently available to the experimenter is the ability to halt and restart

trains. This ability is provided to create unexpected deviations from normal train behavior, such as engine failure. Unlike pre-programmed events that are based on time, or triggered events based on position, a train failure will be interesting only as the result of the situation it breaks down in. For example, a train that breaks down and blocks several tracks may prevent other trains from moving, but it is not known ahead of time when exactly this situation will arise. This operation, used in conjunction with the appropriate messages, will simulate to the dispatcher a realistic train failure. Other such situational operations could be devised and added to the button bar available to the experimenter.

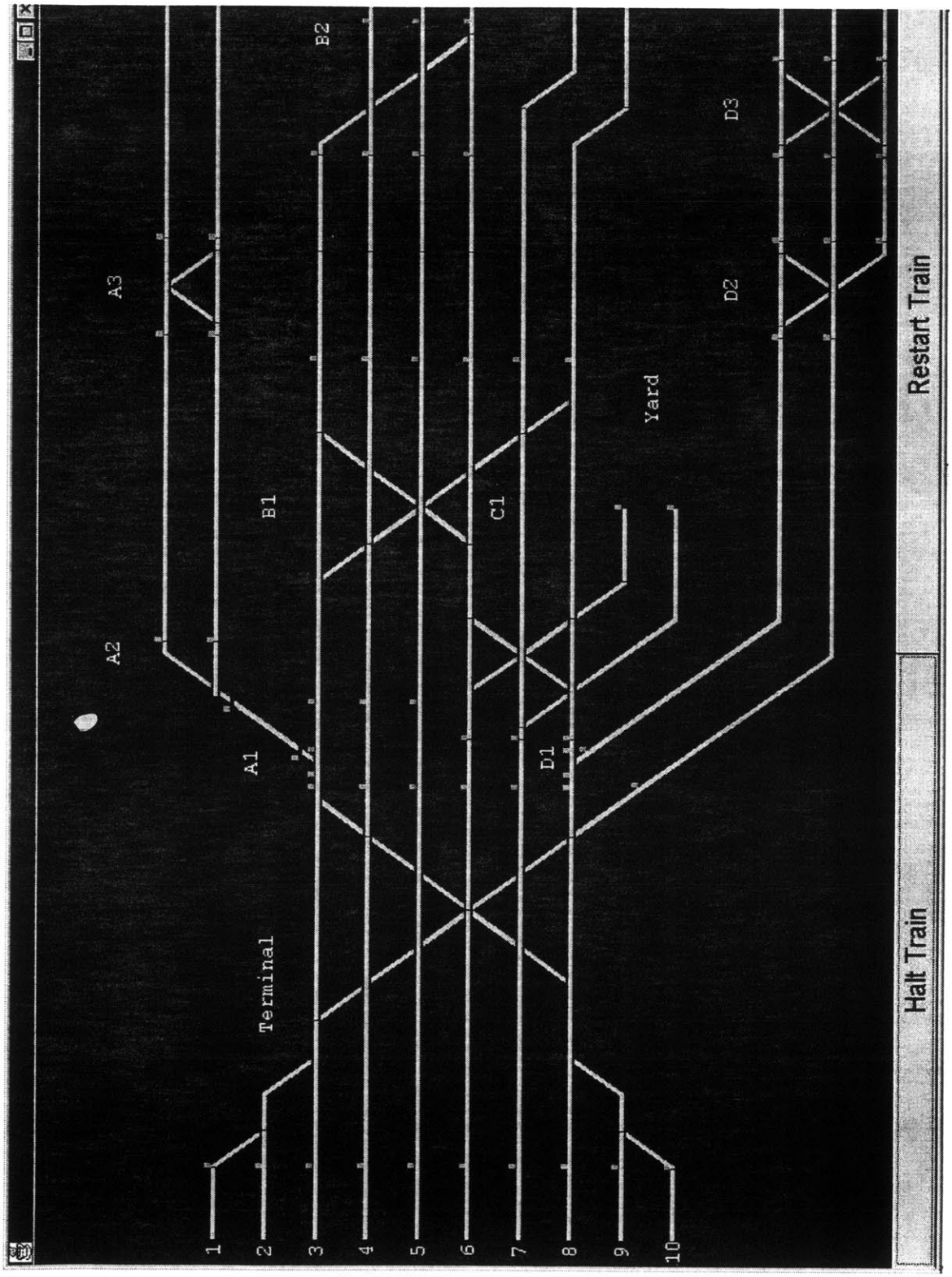


Fig. 5.6 Experimenter Display Terminal

5.4 Experimenter Station: Message Console

The experimenter's message console, shown in figure 5.7, retains the send and reply functionality of the dispatcher console, but these are not really its primary uses. The experimenter's console is used primarily to monitor the message flow throughout the simulator, in a manner that is transparent to the dispatcher. This supports the focus of the project, which is to test the use of a new digital messaging system.

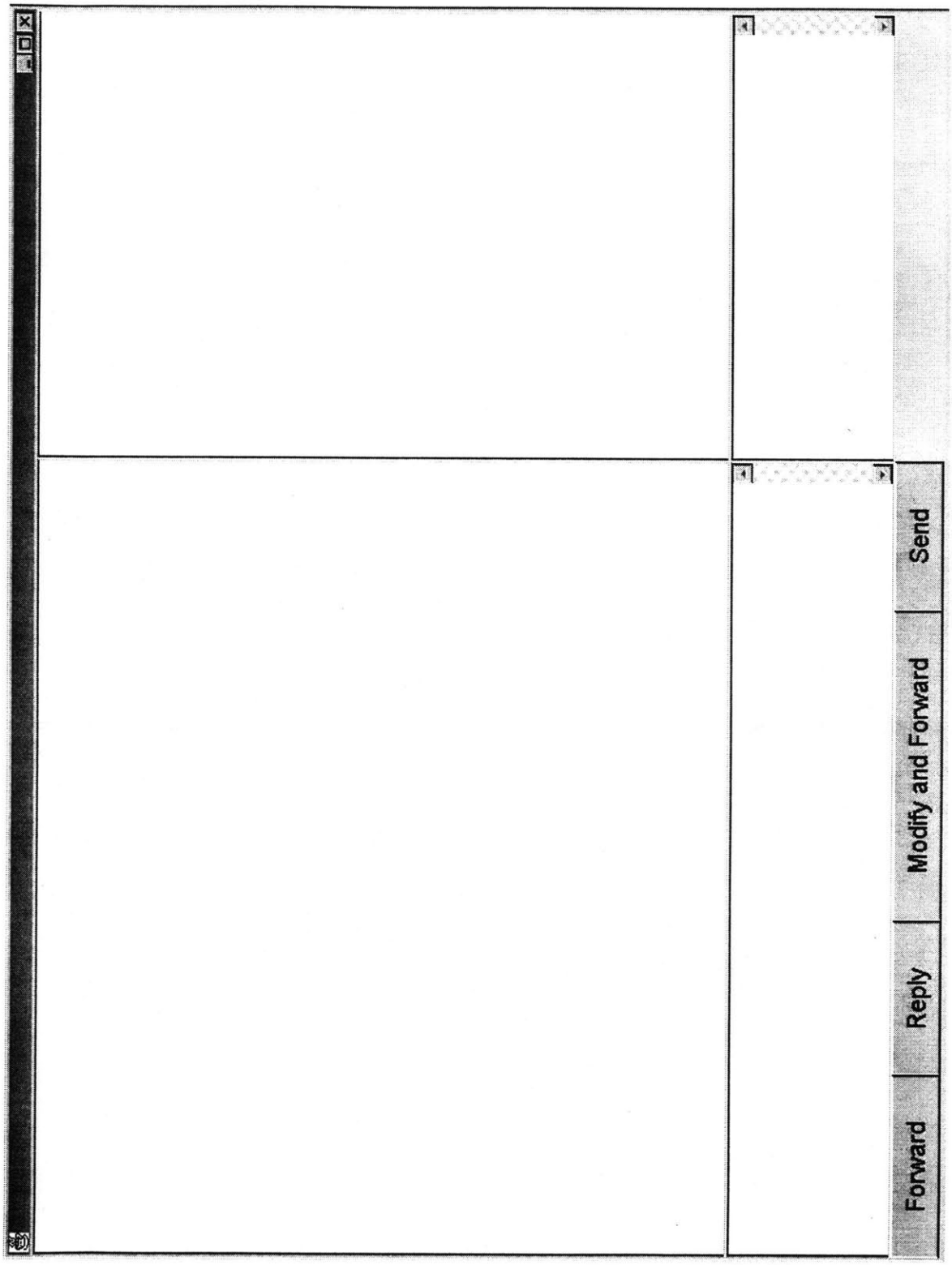


Fig. 5.7 Experimenter Message Console

5.4.1 Visual Layout

The layout of this message console is very similar to the dispatcher version. The main difference is seen at the bottom of the screen where, in addition to the send and reply buttons, there are two others: modify and forward. The use of these buttons is described shortly. Also, there are slight differences in the way message summaries are displayed, which will be discussed.

5.4.2 Incoming Areas

While similar in appearance, the incoming message area serves a somewhat different purpose here. The "incoming" message area is used to display messages that have been sent from one object in the simulation to another and are in the process of passing through the experimenter's station for observation. The one-line summaries are similar in structure but the status marker has only two states instead of three. The messages are either unconfirmed, or confirmed. When the message initially arrives, it is tagged with an unconfirmed marker, which presently is simply the string "<UNCONFIRMED>". When the experimenter clicks on the summary, the entire message text will appear in the lower incoming area. The experimenter can then reply, forward, or modify and forward the message, at which point it becomes confirmed. This is indicated by removing the unconfirmed marker from the message summary. In the incoming message display area, the currently selected message is displayed in the same way as in the dispatcher's message console.

5.4.3 Outgoing Areas

The outgoing area is used to display messages created using the send or reply button. It is not used to display messages that have been forwarded. The rationale is that these messages can already be seen in the incoming area, so even though they are outgoing in a sense, there is no reason to display them again on the right side. The messages that do appear in the outgoing area behave in the standard manner: summaries are displayed on top, and clicking on a summary displays it below.

5.4.4 Console Operations

The console operations available to the experimenter provide a great deal of flexibility in how the experimenter can observe and influence the experiment. Depending on how they are used, his role can range from being a pure observer to being an active participant, playing the roles of work crews or train engineers. The simulator is designed to make it impossible for the dispatcher to distinguish what is pure simulation and what is being "acted" by the experimenter. This allows more complex interactions than would be possible by using only computer simulated elements. This ability to influence the simulation is used in specific cases in the current project, but can be used to a greater extent if desired. It is accessed through the use of the four buttons at the bottom of the experimenter console.

The send button is used to send a new message. The difference between this one, and the send button that the dispatcher uses is that here both the sender and the recipient must be specified (and the sender is not the experimenter). In reality, the experimenter is sending the message. However, the effect is that the recipient thinks that some other object within the simulation sent the message. In some cases, that perceived sender object may exist and be simulated, but lack the ability to decide what messages to send. In other cases, that object may not exist at all. Therefore, its only effect on the simulation is the messages that the experimenter sends on its behalf. One example of this type of nonexistent object is the Engine Repair Crew. There is no program that simulates this. However, by sending messages on its behalf, the experimenter can create the illusion that it exists, and observe the response of the dispatcher. The overall philosophy is that if the dispatcher believes an object exists, it is not necessary that the object actually exist. The perception is sufficient to elicit a response, which can be studied.

It is important to note here that the hierarchical tree used to select a message for the send operation (and the reply operation below) is not the same tree as that seen by the dispatcher. The experimenter has access to all messages, not just the ones used by the dispatcher. In fact, the root level in the experimenter's tree contains a button that leads to

a subtree that is the dispatcher's tree (figure 5.8). Other root level buttons result in trees which can be used to send train engineer messages, work crew messages, etc. This supports the design that the experimenter can potentially pretend to be any part of the simulation.

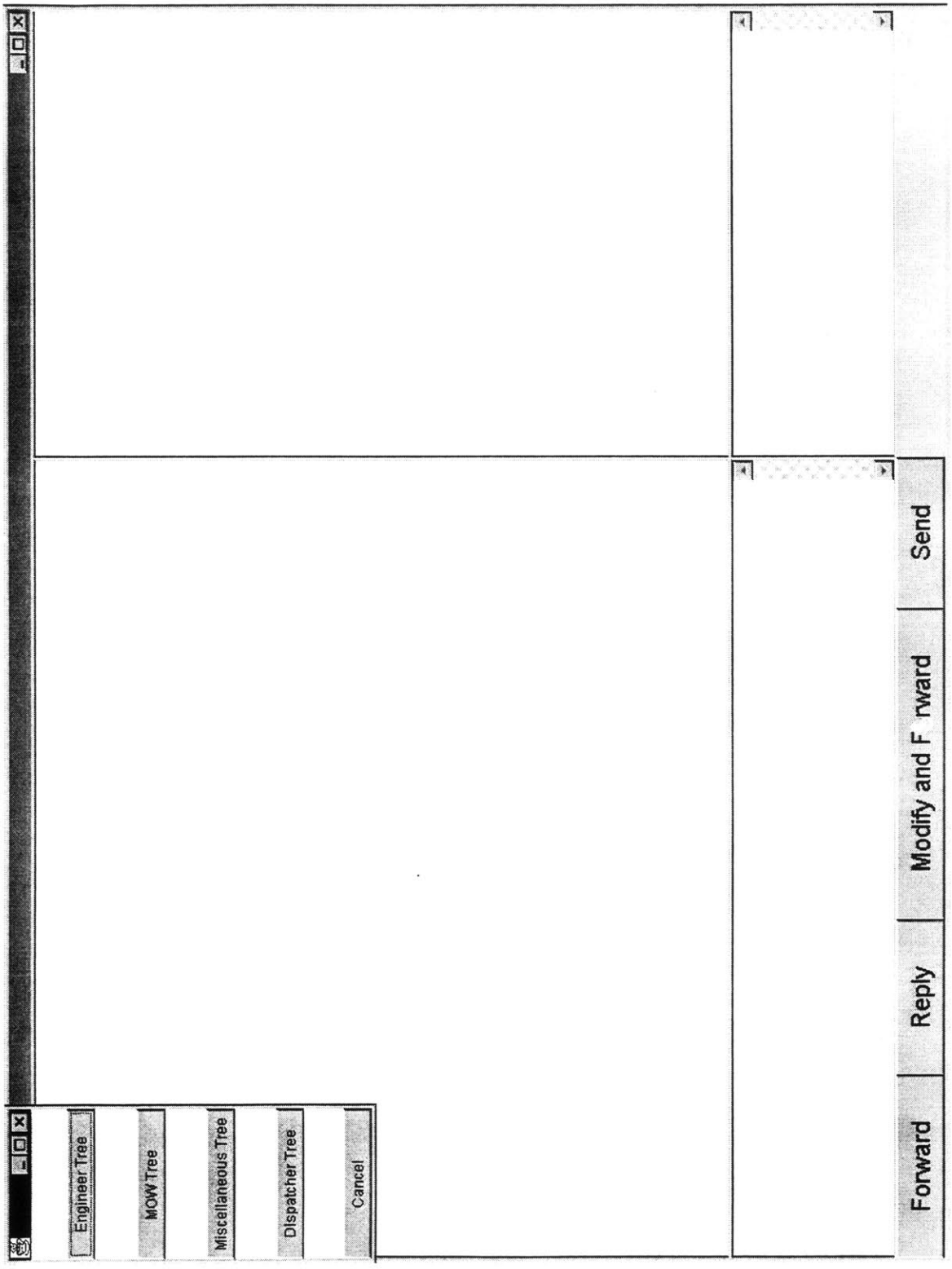


Fig. 5.8 Experimenter Message Selection Tree

The reply button provides another way to play the part of some object. When a message is replied to, it is not forwarded to its destination. Normally, the message would reach the destination, and the simulated object would make some decision, perhaps sending a reply. When the experimenter replies directly, he is essentially making that decision. As one can imagine, there are situations when the decision making process would be too sophisticated for the computer to do it properly. In these situations, the experimenter takes over. He selects a message from the incoming message summaries, clicks the reply button, and selects the message via the message tree, as usual. When the message form appears, the sender and recipient are already filled in, based on the fields of the message to which the reply is being issued. The "to" and "from" fields are simply reversed. The experimenter than fills in the remaining blanks, if any, and sends the message.

The remaining two buttons have very different uses than the send and reply buttons. The first of these is the forward button. This does not result in a selection tree or message form. Instead, the incoming message that is currently selected will be sent to its final destination. It will remain in the incoming summaries section, but the unconfirmed marker placed before the summary will disappear. Any message that is not forwarded (or forwarded modified as described below) will not reach it's final destination. So the forward button serves two purposes. One, it sends the message to the correct object in the simulator. Two, it forces the experimenter to quickly review each message and gives him an idea of what is happening.

Closely related to the forward button is the modify button. This is an interesting feature that may not be used very often in a typical simulation. It allows the experimenter to change a message after it has been sent by its source, but before it is received at its destination. It is strongly recommended that this not be done to messages sent by the dispatcher because it is likely to ruin his sense of cause and effect as far as messaging is concerned. However, in other circumstances, it is conceivable that the pre-programmed behavior of simulated elements may not be satisfactory. They may send the right type of message, but fill in the fields incorrectly because they cannot react to special situations. At times like these, the experimenter has the option of modifying any or all of the fields

in the message. This is accomplished by a message form that appears immediately after the modify button is pressed. Note that there is no selection tree preceding the form. The structure of the form is instead determined by the message that has been selected for modification. After any fields have been changed (or even if no change was made) the experimenter clicks the send button on the message form and the message is sent to its destination. Although very rare, the destination itself can be modified.

6 Experimental Design

6.1 Experimental Setup

For this experiment the simulator was configured to run on five PC's with the following specifications:

CPU	350/400 MHz Intel
Memory	128 MB RAM
Storage	~5 GB Hard Drive
Video	4 MB SVGA Graphics Card (1280x1024 at 65 Hz)
Screen	17 Inch Monitor
Peripherals	Microsoft Intellimouse, Standard 101 Keyboard
Removable Storage	3.5 Floppy Disk Drive
CD	32X CDROM Drive
Network	3C905 or 3C509 series network adapters
Other	XGA Realtime Adapter (SVGA to NTSC signal converter) on two machines

Table 6.1 Hardware Specifications

The most important specifications here are the CPU speed, monitor size, and graphics card resolution. The CDROM is not essential, a smaller hard drive will suffice, and 64 MB RAM will not alter performance. Three of the computers, comprising the dispatchers simulated workstation, were placed in one room. Two of these possessed the SVGA-NTSC adapters; these two were used for the track display, while the third presented the data-link interface screen. The other two computers were placed in another room separated by a door, and provided the experimenter interface.

A clock was placed on the wall facing the dispatcher; this was reset at the beginning of each experiment to reflect the starting time of the scenario. Behind and to the right of the dispatcher a videocamera was positioned to enable a variety of tasks. In the

experimenters room two videocassette recorders were used in combination with the XGA adpater and the videocamera to record the experiment. In the two data-link scenarios (see below) the recorders' video inputs were connected to the two computers equipped with the SVGA-NTSC converters. In the radio scenario, one recorder's video input was connected to the videocamera output. A third monitor was always available to observe any of the three video feeds, but not to record. In all cases, the audio inputs of one recorder was connected to the audio output of the videocamera, while the audio output of the other recorder was connected to a microphone placed near the experimenter's computers.

Because the dispatcher was required to use three mice and one keyboard, extra desk space was provided for writing notes, viewing schedules, filling in D-Forms, and other non-computerized tasks.

6.2 Scenarios

A scenario is the group of files that define a particular simulation. These are data files that are read from disk by the simulator at the beginning of a run, and are used to define the characteristics of the simulator. The aspects of the simulation that can be defined in this way are the number of trains, train behavior, track layout, pre-programmed events, visual layout, station configuration and naming, interlocking configuration and naming, signal placement, initial signal states, initial track states, and initial train states. The means of altering these characteristics are described in the appendix on the simulator architecture.

Two scenarios were designed for this experiment, both equally difficult in terms of the dispatcher workload required in each case. They were designed to force the application of the data-link features of section 4.2.2 to the dispatching tasks of section 4.2.1. The scenarios were designed to be handled by radio communications as well. Under nominal operation, the sequence of events in each scenario is summarized in figures 6.1 and 6.2. In these tables, the first four columns indicate hazardous conditions. The numbers in the

columns indicate which trains, under nominal conditions, will pass these hazardous locations. Shading indicates that the hazard is present, and white that it is gone. The fifth column indicates the bridge operation schedule. The sixth column indicates the data-link messages that will result from the hazards and events in columns 1-5, again under nominal conditions. The last three columns summarize the work events planned in this scenario, and the total number of messages triggered by them. The time for each event is read off the left-hand side of the chart.

	Tresp.		Kids		Bridge	Number of M	MOW	MOW out	Total Mess.
1:00 PM		221							
1:01 PM							SW #1		1
1:02 PM									
1:03 PM									
1:04 PM									
1:05 PM									
1:06 PM									
1:07 PM	100					1			2
1:08 PM									
1:09 PM									
1:10 PM					Disp	2			3
1:11 PM			300				RC #5		4
1:12 PM									
1:13 PM	113							SW #1	5
1:14 PM				443		3			6
1:15 PM							SW #3		7
1:16 PM		223	331			4			8
1:17 PM								RC #1	9
1:18 PM									
1:19 PM									
1:20 PM									
1:21 PM				400		5			10
1:22 PM									
1:23 PM									
1:24 PM							RC #6	RC #2	11,12
1:25 PM									
1:26 PM									
1:27 PM	102	200				6,7	RC #3		13,14,15
1:28 PM									
1:29 PM								SW #3	16
1:30 PM									
1:31 PM									
1:32 PM									
1:33 PM									
1:34 PM							TC #2		17
1:35 PM									
1:36 PM									
1:37 PM		225							
1:38 PM							SW #4		18
1:39 PM									
1:40 PM									
1:41 PM									
1:42 PM	115					8	TC #3		19,20
1:43 PM									
1:44 PM									
1:45 PM							RC #4		21
1:46 PM									
1:47 PM									
1:48 PM									
1:49 PM	117		302			9			22
1:50 PM			333		Disp	10			23
1:51 PM							SW #5		24
1:52 PM									
1:53 PM									
1:54 PM								TC #4	25
1:55 PM									
1:56 PM									
1:57 PM									
1:58 PM	104								
1:59 PM									
2:00 PM									

Figure 6.1 Scenario 1 Script

	Tresp.		Kids		Bridge	Number of M	MOW	MOW End	Total Mess.
4:00 PM									
4:01 PM							SW #1		1
4:02 PM									
4:03 PM									
4:04 PM		240		491		1			2
4:05 PM							RC #2		3
4:06 PM									
4:07 PM									
4:08 PM								RC #1	4
4:09 PM									
4:10 PM					Disp.	2			5
4:11 PM		291	391				RC #5		6
4:12 PM	120					3			7
4:13 PM		242							
4:14 PM									
4:15 PM	191						RC #3		8
4:16 PM									
4:17 PM			360	480		4.5		SW #1	9,10,11
4:18 PM									
4:19 PM									
4:20 PM									
4:21 PM								RC #6	12
4:22 PM									
4:23 PM									
4:24 PM									
4:25 PM							TC #2		13
4:26 PM									
4:27 PM			362						
4:28 PM									
4:29 PM		244				6	RC #8		14,15
4:30 PM			393	493					
4:31 PM									
4:32 PM									
4:33 PM		293							
4:34 PM									
4:35 PM							TC #1	RC #2	16,17
4:36 PM									
4:37 PM									
4:38 PM							RC #7		18
4:39 PM									
4:40 PM				482					
4:41 PM									
4:42 PM									
4:43 PM									
4:44 PM		293				7			19
4:45 PM							RC #4		20
4:46 PM	122					8			21
4:47 PM	193							RC #5	22
4:48 PM			364			9			23
4:49 PM									
4:50 PM					Disp.	10			24
4:51 PM									
4:52 PM							TC #2		25
4:53 PM									
4:54 PM									
4:55 PM									
4:56 PM								TC #2	26
4:57 PM									
4:58 PM									
4:59 PM									
5:00 PM									

Fig. 6.2 Scenario 2 Script

6.3 Subject-Scenario Matrix

The experiment was organized as a set of 12 simulations (one hour each) using 6 dispatchers from CETC as participants. Each dispatcher visited the Volpe Center and participated in two one-hour simulations on the same day. The simulations were categorized as radio, *data-link directed*, and *data-link broadcast*. Each dispatcher participated in two different categories. The radio category involved running a scenario where the radio was used as the communications medium. Data-link directed used the system configured in point-to-point mode, while data-link broadcast configured it in a multicast mode. While the simulator validation can be accomplished with any combination of a radio and a data-link scenario, the full test set is included here for completeness. The setup is summarized in table 6.2.

	Data-link Directed	Data-link Broadcast	Radio
Dispatcher 1	Scenario 1	Scenario 2	
Dispatcher 2	Scenario 1		Scenario 2
Dispatcher 3		Scenario 1	Scenario 2
Dispatcher 4	Scenario 2	Scenario 1	
Dispatcher 5	Scenario 2		Scenario 1
Dispatcher 6		Scenario 2	Scenario 1

Table 6.2 Subject Scenario Matrix

6.4 Experimental Procedures

Two experiments were conducted per day, and apart from the configuration of the simulator, the procedures followed were the same each day. At the beginning of the day, the dispatcher was given an overview of the project and it's goals, of course without disclosing information that could bias the experimental results. They were asked to complete two forms. One provided the information required to compensate them for their time. The other was a statement describing their rights as experimental subjects, which they read and signed. Any general questions about the project were asked and answered at this point.

After this the simulator was placed in the first of three configurations. This first setup was the training configuration. It was a simple scenario with a reduced number of trains and messages, although the track layout was identical to the full-scale scenarios in figure 6.1 and 6.2. The training scenario was used to introduce the dispatcher to the various interface elements of the simulator. After describing these, he was asked to begin routing the trains based on the training schedule. In this first pass, he was not required to handle the data-link messages, which were answered for him. He was allowed to become accustomed to the aspects of the simulator that had corresponding features at CETC. Any questions were answered in the process of routing. Also, key differences between the track display in the simulator and that at CETC were explained. Usually after about half an hour of this, the dispatchers felt comfortable with the non-data-link part of the simulator.

In the next phase, the dispatcher was given an overview of the data-link system. Then the dispatcher was asked to repeat the training scenario, but this time he was required to route the trains and answer the messages. Again, questions that were raised during this phase were answered to the dispatcher's satisfaction. Usually the second phase lasted half and hour as well.

After the training phases were completed, the simulator was reconfigured to run one of the full scenarios. The dispatcher was then provided with the train schedules for the scenario, and allowed to study it for some time. Following this, he performed routing operations in the full scenario, but was not shown the messages. The routing practice was felt necessary to the experiment. If the routing had been completely new to the dispatcher during the experiment, he may have concentrated entirely on working out the optimal paths, rather than viewing and answering the data-link messages. The intention was to maintain a relatively complex train management burden, but to allow the dispatcher to form the equivalent of "routing habits" which require advance knowledge of nominal routes.

Usually within thirty minutes to an hour, the dispatcher would say that he was ready to begin the full scenario. At this point the simulation was restarted and all recording equipment turned on. The clocks were reset to the appropriate time, and we, the experimenters, retired to the second room where the monitoring equipment was located. Here the progression of the simulation was monitored, the message flow checked, and events compared to the scenario's nominal event sequence. All three computer monitors in front of the dispatcher could be seen through either the videocamera, or SVGA-NTSC converters. All activities undertaken by the dispatcher could also be seen via the experimenter screens, which displayed an exact copy of the entire track display.

Halfway through the one hour experiment, the simulator was halted and the dispatcher was given a situation awareness test. After completing this (usually about five minutes) the simulator was restarted and the remaining half hour was completed. The same procedures were followed for the second scenario. After both scenarios were completed, an exit questionaire was presented to the dispatcher, and he was given the opportunity to make general comments and suggestions about the system. Usually, an informal discussion ensued, which was often interesting but not strictly part of the experimental procedures.

In the scenarios involving radio communications, we played a special set of roles. There was no data-link system available in these cases; the dispatcher's third monitor was shut off. All information sent and received by the dispatcher was via walkie talkie receivers. Thus, we were required to play the role of all elements of the simulation that previously had communicated via data-link. This included train engineers, maintenance of way personnel, operators of track-side machinery, and any other parties creating radio "chatter" due to the inherent broadcast nature of the radio.

Interestingly, the data-link system, although unavailable to the dispatcher, still played a vital role in the radio scenarios. Maintaining multiple radio personalities was a difficult task, and the data-link system acted as a teleprompt to let us know when to speak, and what information to relay. Essentially, the messages that would otherwise have been sent

to the dispatchers message console were intercepted by the experimenter message console and used as cues. We spoke the lines as they were displayed. Conversely, when the dispatcher responded over the radio, we translated the verbal communication into a data-link message so that the simulator could respond to the information. The overall process was quite complex, and it required two people to handle uninterrupted communication.

6.5 Metrics

Recalling the objectives of this project, the goal is to investigate the usefulness of the simulator. Carrying out the experiment described in previous sections is therefore a type of validation. If the simulator can provide non-obvious information about the data-link systems, and if this information can be used to draw conclusions about those systems, then it will have proven to be a useful tool. Therefore, in choosing which metrics to study, an emphasis is placed on extracting information about the data-link systems and dispatcher usage habits that could not be obtained without the simulator.

6.5.1 Raw Data

Raw data refers to the data collected during the simulation, either in the form of videotaped actions, or data recorded by the simulator on the computers. In the latter case, each piece of data is a single event occurring at a specific point in time. Individually, these do not provide very valuable information, because there is no context in which to frame the event. When coupled with the continuous videotaped simulation, they become more valuable. Even more information is gained by considering subsets of these events that are related to each other through cause-effect relationships. The question to be answered is to what extent such information can be reliably extracted from the raw data.

The types of events captured in the raw data file are summarized in table 6.3.

Event Type	Information Recorded
Train Move	Movement of a train from one block to another
Interlock Entry	Entry of a train into an interlocking, and delay based on nominal schedule
Interlock Exit	Exit of train from an interlocking
Station Entry	Entry of a train into a station
Station Exit	Exit of train from a station, and delay based on nominal schedule
Route Set	Setting a route to allow travel, and signals used to set the route
Route Reset	Resetting route to default state, and signal used to reset route
Block Track	Protection setup for a track location
Unblock Track	Protection removed from a track location
Message Send	Completed data-link message sent
Message Read	Message viewed by recipient
Message Reply	Reply to a message initiated
Message Initiated	New message initiated
Message Canceled	Previously initiated message canceled before being sent

Table 6.3 Raw Data Outputs

The videotapes made for each simulation run are also considered raw data. In the data-link cases, they act more as a contextual framework in which to frame the recorded events. Also, if there is ambiguity in a recorded action, the videotape may serve to confirm or refute the initial perception. In the radio case, the videotape is a more important source of raw data. While certain actions are still captured more effectively using the simulator, the radio communications can only be recorded on videcassette.

6.5.2 Quantitative Information

By analyzing the data, meaningful information can be extracted. This is the key to validating the simulator, determining what kind of information can be successfully measured. We propose the following quantitative measurements, summarized in table 6.4, then described in further detail.

Data-link	Radio
Average Deviation from Nominal Schedule	Average Deviation from Nominal Schedule
Hazard Notification Ratio	Hazard Notification Ratio
Average Time to Acknowledge Message	Average Time to Initiate Response
Average Time to Initiate Response	
Average Message Creation Time	Average Communications Duration
Total Duration of Messaging Activities	Total Duration of Radio Communications

Table 6.4 Proposed Quantitative Measurements

These metrics were deemed to be useful in studying the tradeoffs between the data-link and radio communications mediums. A discussion of why these were chosen is presented in the complementary report by Malsch. The aim here is to determine whether the simulator can provide the necessary information. The schedule deviation and hazard notification measurements seem to be relatively easy to capture. The others however are rather more complicated because they involve assumptions about what a dispatcher's action says about his state of mind. These assumptions are clearly noted below. Also note that the table above is organized such that measurements listed on the same line are considered to be comparable. In some cases, the same measurement is possible in both radio and data-link cases. In other cases, there is no equivalent, due to the fact that a conversational medium is being compared to a non-conversational one. In these cases every effort is made to compare similar information.

Average deviation from nominal schedule is essentially a way to measure how efficiently trains are being routed. For each scenario, schedules were created specifying the nominal time at which each train was expected to arrive at each interlocking. These schedules were tested by us to ensure that the nominal values were attainable. The simulator measures how far ahead or behind these nominal values each train is running.

Hazard notification ratios are used to indicate the level of safety being maintained by the dispatcher. Each scenario was programmed to create a set of hazardous conditions. During the simulation, the dispatcher was notified of these conditions by a passing train. He was required to forward this hazardous information to other passing trains. Failure to

do so was considered a breach of safety. This is captured quantitatively as a ratio between the number of correct notifications issued and the number of notifications required given the available information. This is admittedly a simplistic way to measure safety. However, work involving conflict probes and monitors is currently being carried out as an extension to this simulator, and this will allow more realistic assessments of safety.

The average time to acknowledge a message is a metric specific to the data-link scenario. It is assumed that acknowledgement occurs at the time the message is first viewed. Therefore the quantity that needs to be measured is the time between receipt of a message and the first viewing of that message. There is no exact equivalent in the radio case because acknowledging a message and initiating the response are the same thing.

The average time to initiate a response is measured in both radio and data-link cases. In the data-link case, it is taken to be the time between acknowledgement of the message and the initiation of the response. In the radio case it is the time between when the initial message is completed and the response is begun.

The average message creation time is measured between initiating a data-link message (be it a response to another message or a new message) and sending that message. The radio counterpart to this metric is the average communications duration. This is taken to be the time between initiating a sequence of radio communications on a specific topic and completing the last exchange on that same topic. Timespans during which one party in the sequence has been told to "standby" do not count toward these measurements. Issuance of standby commands are standard procedure in radio communication. There is a certain amount of difficulty in comparing message creation time with communications duration. Creating a message is a one-time action; a radio sequence is several actions. It is not entirely clear that these are comparable values. However, an assumption was made that if the two communications sequences convey the same information, then the time to execute those sequences are comparable. This was assumed to hold whether the sequence was one-step as in data-link, or a back and forth discussion as in radio.

The total duration of messaging activities is the total time spent using the data-link features, regardless of what is being done. It is intended to measure how much time is taken away from all other activities that the dispatcher is undertaking. Its counterpart in the radio case is the total time of radio communications, i.e. any time at which any audio communication is taking place between the dispatcher and another party. This is an interesting metric because in the data-link case the total duration of messaging will necessarily be far less than one hour in a one hour simulation. The dispatcher cannot use the data-link features, as they now stand, without interrupting other activities. In contrast, the total time of radio communication could conceivably be one hour in a one hour simulation, although in practice it is not.

6.5.3 Qualitative Information

In addition to numerical measurements, an important source of information will be unstructured observation on our part, and unstructured comments on the part of the participant. It is hard to say what qualitative observations will be obtained. It is perhaps better to simply keep an open mind and present a discussion in the results section.

7 Results and Discussion

The results for one dispatcher are now analyzed. Information was extracted from the raw data outputs. In most cases the metrics outlined in section 6.5.2 were successfully obtained. The measurements from both a radio and data-link scenario are presented, except for train delays which are measured identically in both cases. In the cases where there was ambiguity or difficulty in interpreting the raw data, the causes and potential solutions are outlined with an eye toward improving the simulator. In the complementary report by Malsch, these results, and those for the other participants, are used to study the data-link system as it compares with radio communications.

7.1 Hazard Notification Ratio

The hazard notification ratio is a simple measure of how well the dispatcher was maintaining safety by acting as a relay point for information to and from trains. The scenarios were designed to create potentially hazardous situations on the various branches. Trains passing through these zones reported the situation to the dispatcher, and his job was to relay the information to other trains that might traverse the same route. Some of those trains may have detected the hazardous situation had disappeared, and they relayed this to the dispatcher. His job was to again relay this new information to trains that had previously been told to watch for hazards.

Tables 7.1(a-h) below outline three types of information from which the hazard ratio is calculated. All information within the same column is organized chronologically. The first column in each table is the nominal sequence. This represents the set of messages that would have been passed back and forth between trains and the dispatcher if all the trains had been dispatched perfectly. It is based on known train speed profiles, hazard timing and nominal routes. In most cases the actual routing deviates from the nominal, due mainly to unfamiliarity with the schedule, despite the training period.

The second column displays the sequence of communications that should occur given the way that the simulation has evolved. It takes into account delays and mistakes in routing. This column is determined using the knowledge of hazard timing, and the actual movement of the trains, which is recorded and discussed in section 7.5.

The third column is the actual communications sequence that took place. This information was either captured from the videotape of the radio communications, or from the recordings created by the simulator. The hazard ratio is based on a comparison of column two and three. This means that the effect of incorrect routing is removed from the calculation. The intention is isolate the communications sequences, and remove the influence of inexperience with the schedules. This inexperience was a necessary limitation, since our participants, who normally took several days to become accustomed to a schedule, were only allowed a few hours due to time constraints.

In the following tables these symbols are used; in each, nnn is replaced by a train number.

D→# nnn ⊕ : Dispatcher tells train that hazard is present

D→# nnn ∅ : Dispatcher tells train that hazard is no longer present

D←# nnn ⊕ : Train tells dispatcher that hazard is present

D←# nnn ∅ : Train tells dispatcher that hazard is no longer present

Hazard Type: Trespasser on Track		
Location: Branch A		
Communications Medium: Radio		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#120 ⊕	D←#120 ⊕	D←#120 ⊕
D→#191 ⊕	D→#191 ⊕	D→#191 ⊕
D→#122 ⊕	D→#122 ⊕	
D→#193 ⊕	D→#193 ⊕	
D←#122 ∅		
D→#193 ∅		

Table 7.1a

Hazard Type: Trespasser on Track		
Location: Branch B		
Communications Medium: Radio		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#244 ⊕	D←#242 ⊕	D←#242 ⊕
D→#293 ⊕	D→#291 ⊕	D→#291 ⊕
	D→#244 ⊕	D→#293 ⊕
	D→#293 ⊕	D→#244 ⊕

Table 7.1b

Hazard Type: Kids Stoning Train		
Location: Branch C		
Communications Medium: Radio		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#360 ⊕	D←#360 ⊕	D←#360 ⊕
D→#362 ⊕	D→#362 ⊕	D→#393 ⊕
D→#393 ⊕	D→#393 ⊕	D←#362 ⊕
D→#364 ⊕	D→#364 ⊕	
D←#364 ∅	D←#362 ∅	
	D→#364 ∅	
	D→#393 ∅	

Table 7.1c

Hazard Type: Kids Stoning Train		
Location: Branch D		
Communications Medium: Radio		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#491 ⊕	D←#491 ⊕	D←#491 ⊕
D→#480 ⊕	D→#480 ⊕	D→#480 ⊕
D→#493 ⊕	D→#493 ⊕	D←#480 ∅
D→#482 ⊕	D→#482 ⊕	
D←#480 ∅	D←#480 ∅	
D→#493 ∅	D→#493 ∅	
D→#482 ∅	D→#482 ∅	

Table 7.1d

Hazard Type: Trespasser		
Location: Branch A		
Communications Medium: Data-link		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#100 ⊕	D←#100 ⊕	D←#100 ⊕
D→#113 ⊕	D→#113 ⊕	D→#113 ⊕
D→#102 ⊕	D→#102 ⊕	D→#115 ⊕
D→#115 ⊕	D→#115 ⊕	D←#104 ⊕
D→#117 ⊕	D→#117 ⊕	D→#117 ⊕
D←#102 ∅	D←#102 ∅	
D→#115 ∅	D→#115 ∅	
D→#117 ∅	D→#117 ∅	
D→#104 ∅	D→#104 ∅	
D←#115 ⊕	D→#113 ∅	
D→#117 ⊕	D←#104 ⊕	
D→#104 ⊕	D→#117 ⊕	

Table 7.1e

Hazard Type: Trespasser		
Location: Branch B		
Communications Medium: Data-link		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#223 ⊕	D←#223 ⊕	D←#223 ⊕
D→#200 ⊕	D→#200 ⊕	D→#200 ⊕
D→#225 ⊕	D→#225 ⊕	D←#200 ∅
D←#200 ∅	D←#200 ∅	
D→#225 ∅	D→#225 ∅	

Table 7.1f

Hazard Type: Kids Stoning Train		
Location: Branch C		
Communications Medium: Data-link		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#302 ⊕	D←#302 ⊕	D←#302 ⊕
D→#333 ⊕	D→#333 ⊕	D→#333 ⊕

Table 7.1g

Hazard Type: Kids Stoning Trains		
Location: Branch C		
Communications Medium: Data-link		
Nominal Sequence	Correct Sequence	Dispatched Sequence
D←#443 ⊕	D←#443 ⊕	D←#443 ⊕
D→#400 ⊕	D→#400 ⊕	D→#400 ⊕
D←#400 ∅	D←#400 ∅	

Table 7.1h

Now the hazard notification ratio can be calculated by taking the number of notifications that were sent by the dispatcher (third column) and dividing by the number that should have been sent (second column). Sent notifications are indicated by entries starting with D→. The ratios are summarized in the table below.

	Radio	Data-link
Trespasser, Branch A	0.33	0.33
Trespasser, Branch B	1.0	0.33
Stoning Train, Branch C	0.2	1.0
Stoning Train, Branch D	0.2	1.0

Table 7.2 Hazard Notification Ratios

Table 7.2 demonstrates that the simulator was able to capture the information required to make a side by side comparison of basic safety levels. The main difficulty, which was partly anticipated, was creating the second column. This required knowledge of train delays, because this information has a direct effect on frequency and timing of reported hazards. The simulator captured this information by tracking the movement of the train from block to block, through interlockings, and during station stops. It appears that this level of detail is sufficient to construct the tables.

7.2 Average Time to Response

This value is indicative of how busy the dispatcher is, and how quickly he can respond to an incoming message. Early on in the project the problem of comparing the

conversational radio responses to the non-conversational data-link responses was recognized. It was hypothesized that the radio response time could be compared to the sum of the acknowledgement and response time in the data-link case. Table 7.3 below summarizes these values. They give overall times, and times grouped by task type. The task types are work-related, TSRB (Temporary Speed Restriction Bulletin, which is a train related task), and other (bridge operations in these scenarios)

	Overall	Work	TSRB*	Other
Radio	18	21	17	9
Data-link Acknowledgment	49	58	34	46
Data-link Response	21	32	4	12
Data-link Total	70	90	38	58

* Only one response time (62 s) exceeded 10 s in the radio case

Table 7.3 Communication Response Times

The timing values above are all given in seconds. In the radio case, these values were obtained by simply listening to the videotaped run. The simulator provided the data for the data-link case, in the form of four types of records. The first one recorded when a message was received by the dispatcher. The second one recorded when the dispatcher first viewed the message. The third one recorded when the dispatcher began answering the message. And the fourth one recorded when the dispatcher completed the response.

An assumption, which can be reasonably supported, was made in interpreting the data. It was assumed that by viewing the message, the dispatcher was indicating that he read the message as well. There was a slight chance that dispatchers would press the keyboard buttons and scroll back and forth through the messages frivolously. The simulator had no way to distinguish between this and a genuine message viewing. For two reasons, it was clear that this happened very infrequently. One, we did not observe this kind of behavior during the experiments. Two, after completing the experiments the record files did not display evidence of this in the form of a rapid succession of "view message" records.

It is reasonable, when comparing task types, to say that higher average response time indicates lower cognitive priority. The simulator can therefore be used to rank cognitive

priorities. For instance, in both the data-link and radio case, track work related communication seems to be given lower cognitive priority then train related communication. This conclusion drawn from the data was confirmed by discussions with dispatchers who all stated that they would always deal with train related communication before track related communication except in exceptional situations.

The times can also be used to compare data-link with radio communications, which was one of the key requirements to validate the simulator. The data indicates that in the data-link case, the dispatcher waits substantially longer in all cases before responding to a message.

7.3 Average Response Duration

In the radio scenarios, the average response duration was measured by simple observation of the videotaped experiment. The simulator provided the data necessary to calculate the average response duration in the data-link cases. Once again the values being measured in the two cases were slightly different. In the radio case, average response duration included the entire back and forth sequence of communications, confirmation, etc. In the data-link case, the response duration was actually the time taken to complete the message form for each response. It was assumed that the two values could be compared if the information exchanged in each case was the same. From observation of the videotape and the records file this was found to be true in almost all cases. The only exceptions were instances when the dispatcher placed a calling party on standby in the radio case, and completed the communication later. If it was clear that the intervening timespan was not used to address issues related to the original communication, it was discounted from the response duration. The response duration values are summarized in table 7.4; all times are in seconds.

	Overall	Work	TSRB*	Other
Radio	63	102	10	12
Data-link	30	45	9	22

Table 7.4 Average Response Durations

These results demonstrate the simulator's ability to provide information about the communications workload. By measuring the time spent on various tasks, the simulator can help make decisions about what information should be transmitted via data-link instead of radio. For example, if results indicate that the communications time devoted to managing work permissions is cut in half by using data-link, then there is an argument for doing that. In fact, this kind of time savings is apparent in Malsch's comparison of data-link vs. radio. Furthermore, dispatchers have stated that work permission management would be a good candidate for data-link communication. This illustrates how information gained by simulation can be confirmed by follow-up discussions with participants.

7.4 Total Communications Workload

This metric was simply a means to compare how much time the dispatcher spent communicating over the radio and over the data-link messaging system. This information is useful because if the same amount of information can be conveyed in less time, all aspects of dispatching would benefit. Bandwidth would be used more effectively, and dispatchers would have more time to think about their course of action. Note that this measure does not include time that is potentially spent thinking about what information to communicate. It is solely a measure of the time required to communicate it. Again the radio information was gathered via direct observation. The simulator provided the raw data from which the data-link information was extracted. Table 7.5 summarizes overall communications workload, and workload by category of communication.

	Radio	Data-link
Hazard Notification	185	176
Work Permission Management	1117	407
Temporary Speed Restrictions	70	51
Other	48	44
Total	1420	678

Table 7.5 Total Communications Duration

Here again, because two different media of communication are being compared, an assumption was made. It is assumed that the time between initiating a data-link reply and completing that reply is spent constructing the reply. This is in fact known to be false because dispatchers were observed several times completing some other small tasks within this timespan. In essence, the dispatcher was putting the reply form on "standby", similar to the way a standby command was issued over radio communications. In both cases, the dispatcher attended to more pressing matters. It is easy to measure the standby times in the radio case because the command was issued explicitly. It is not possible to accurately measure the corresponding value in the data-link case because it requires knowledge of where the dispatcher's eye is focused, this being indicative of the focus of cognitive attention. Therefore we must conclude that the times listed for total data-link communications duration are necessarily high estimates.

7.5 Train Delays

Train delays were recorded by the simulator to provide an idea of how efficiently the routing tasks were accomplished. This was one of the easiest measurements to take since a nominal schedule provided a baseline, and delays simply required measurement of the time at which trains passed each interlocking. There was no difference in the method of calculating delays between the radio and data-link cases since the trains were simulated elements. To demonstrate this capability of the simulator, the train delays for a data-link scenario are shown in the charts below.

In these charts, delays are given in seconds. It is clear that the number of data points is not consistent from chart to chart. This is because different trains traverse different numbers of interlockings on their scheduled route. The charts are not meant to be directly compared to each other, but rather to provide a picture of the progression of each trains through its nominal route. Each bar represents the train's delay as it passes an interlocking on its route, with the first one visited placed on the left side of the chart. Positive delays indicate that the train is behind schedule, negative delays ahead of

schedule. These charts can be compared to the train schedules in Appendix B and the territory map in Appendix C to spatially locate each interlocking.

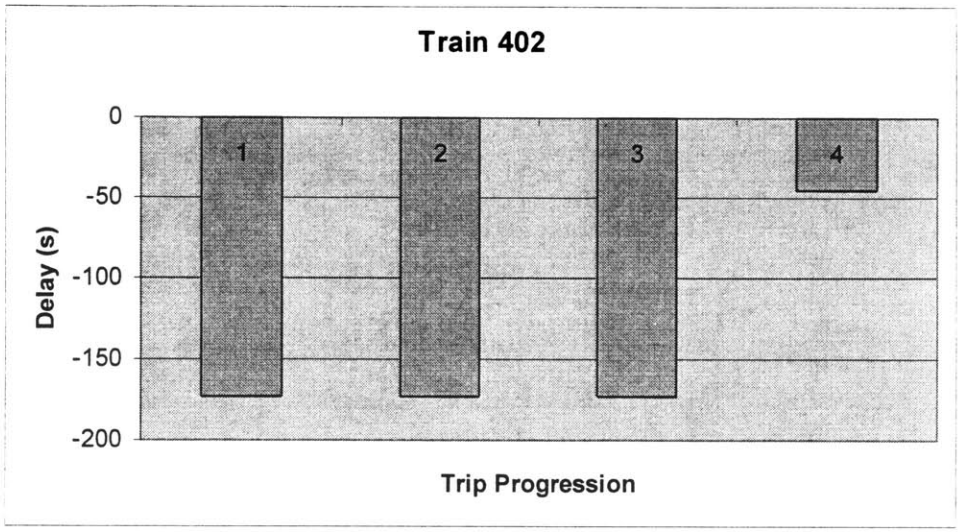


Fig. 7.1b

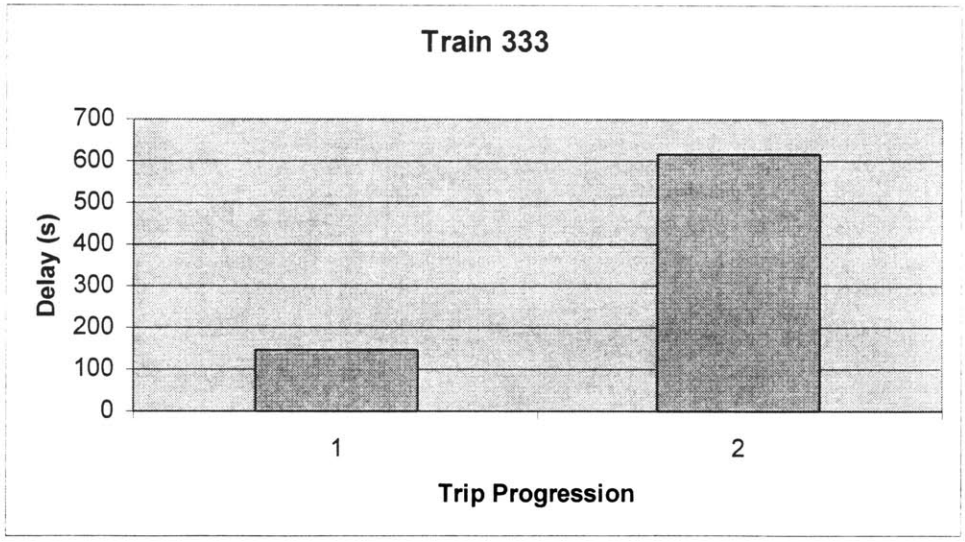


Fig. 7.1c

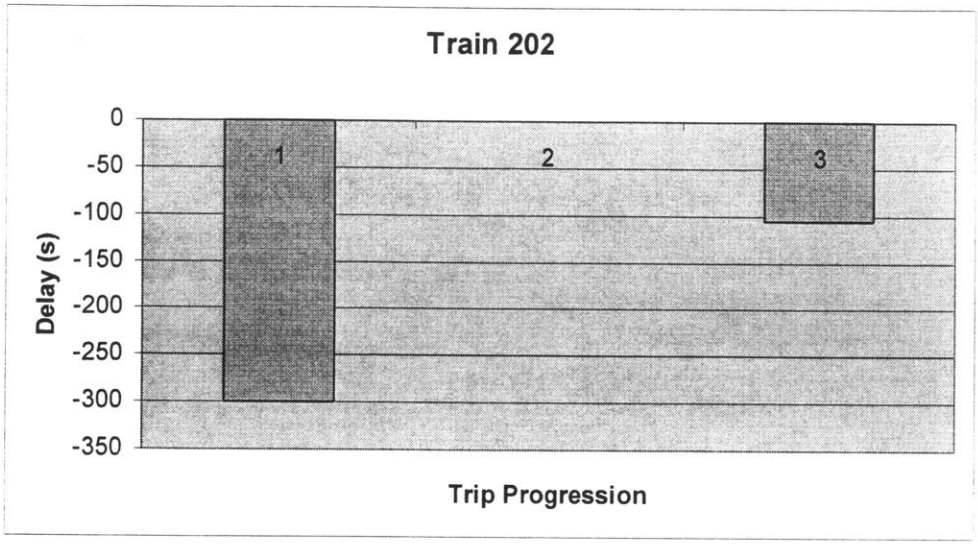


Fig. 7.1d

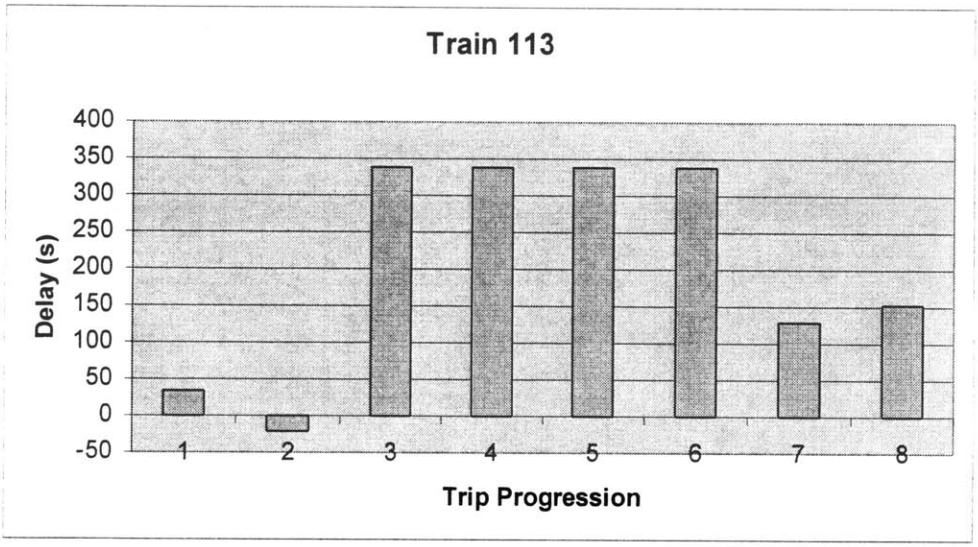


Fig. 7.1e

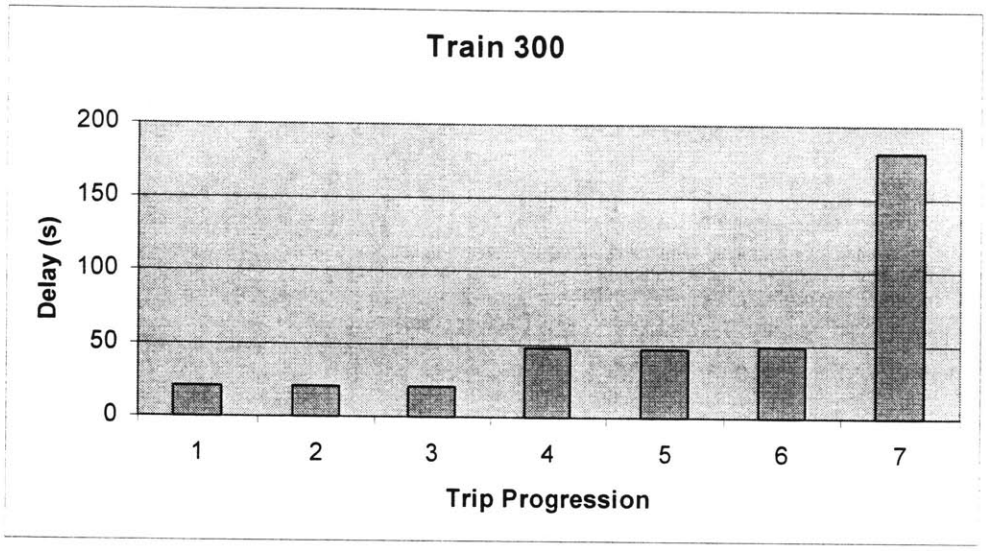


Fig. 7.1f

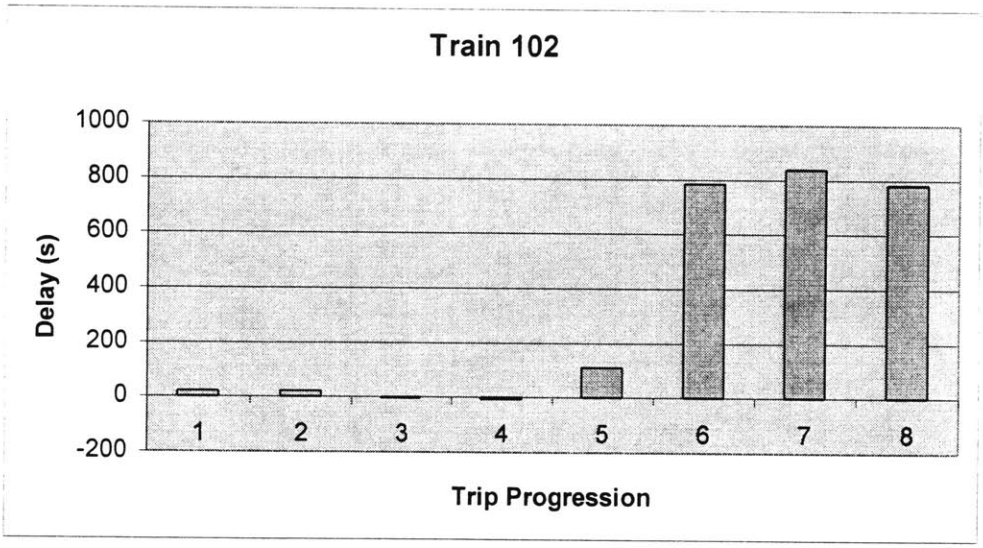


Fig. 7.1g

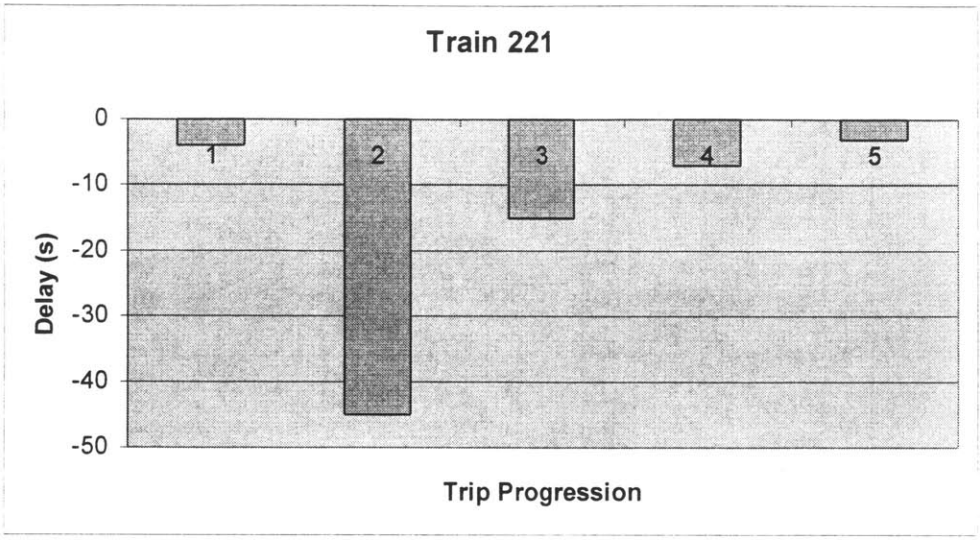


Fig. 7.1h

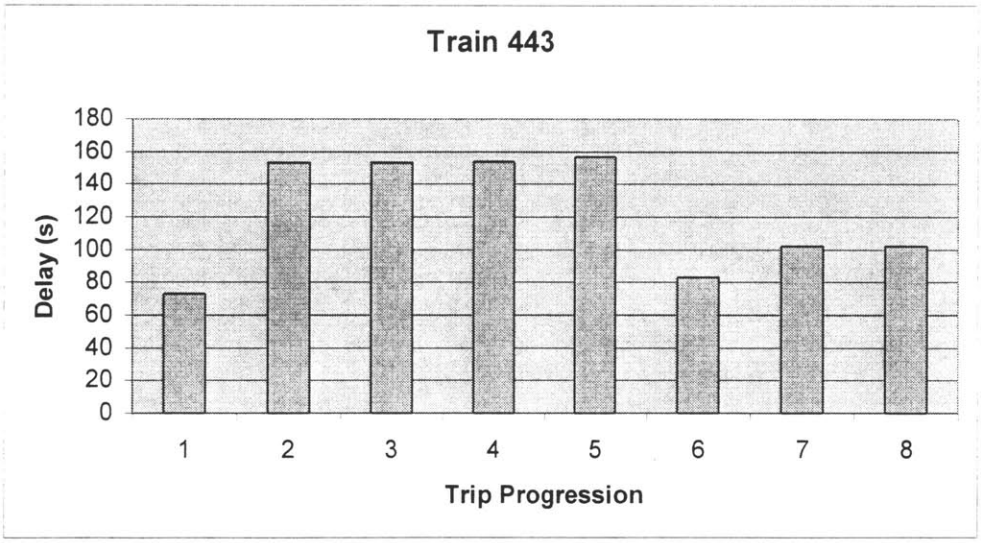


Fig. 7.1i

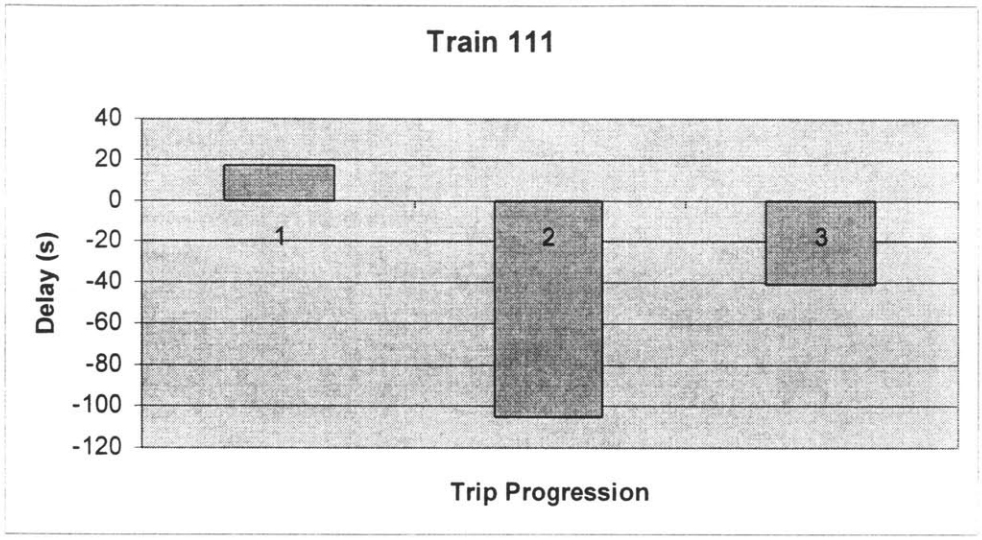


Fig. 7.1j

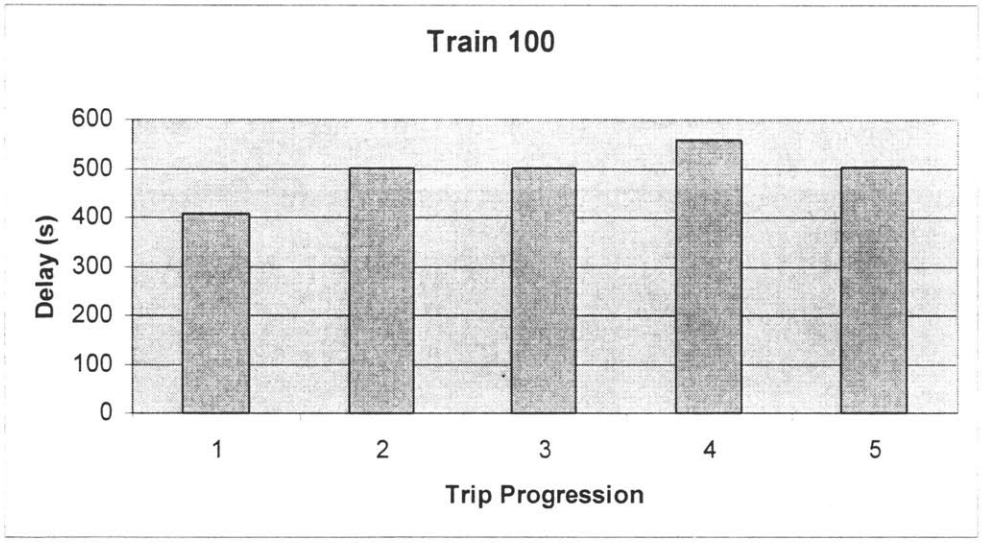


Fig. 7.1k

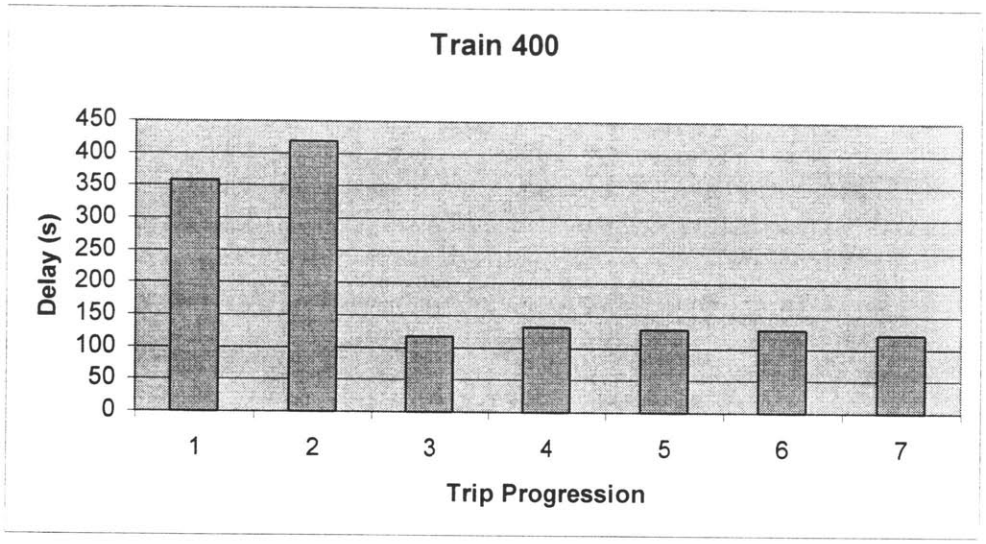


Fig. 7.1l

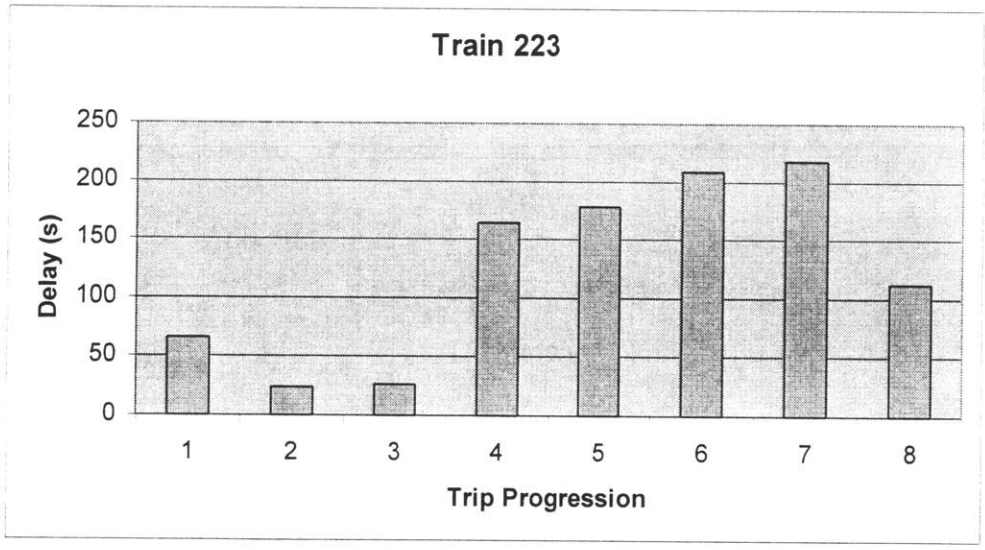


Fig. 7.1m

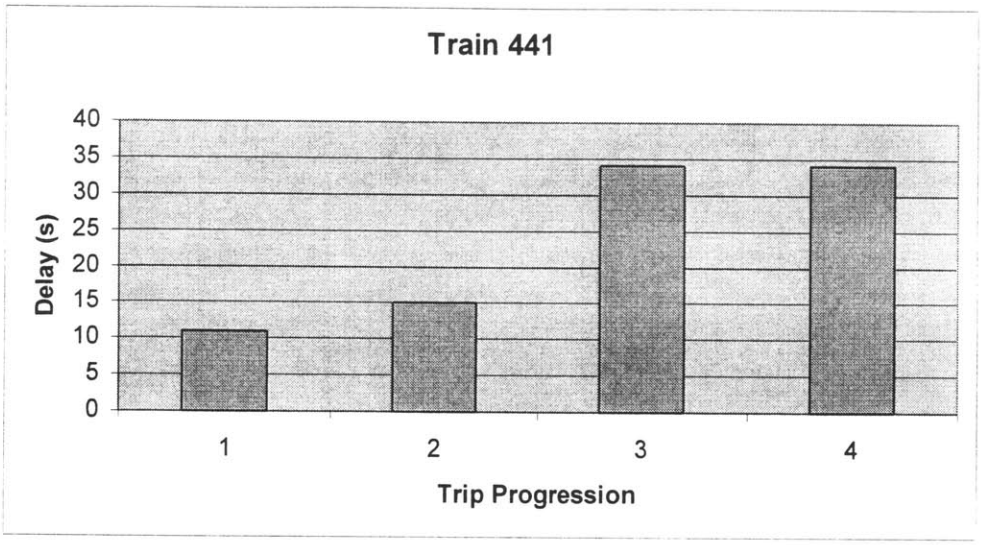


Fig. 7.1n

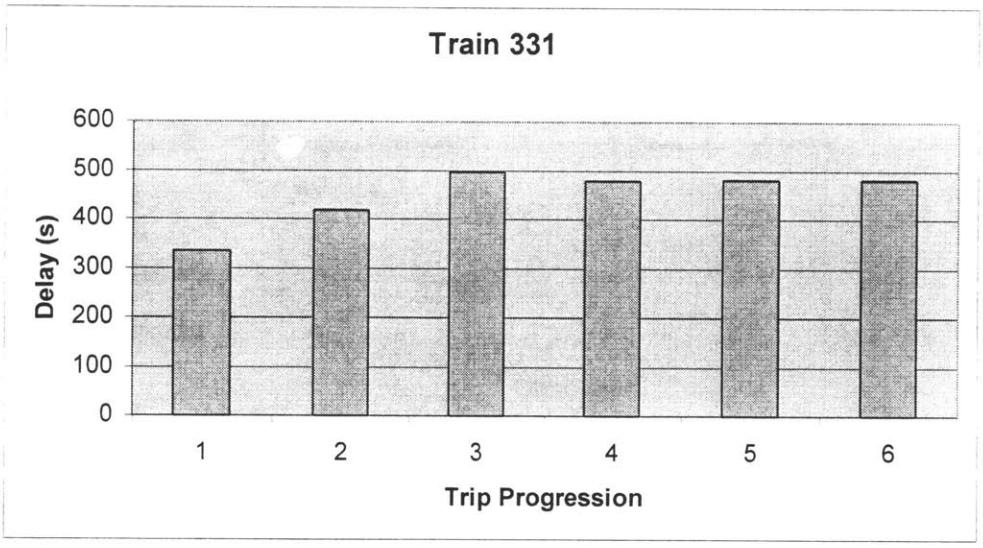


Fig. 7.1o

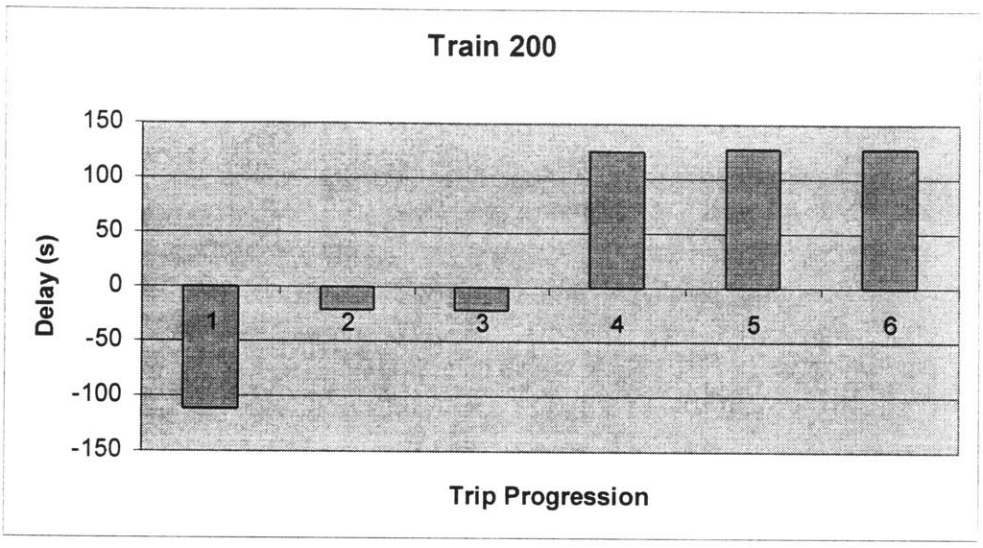


Fig. 7.1p

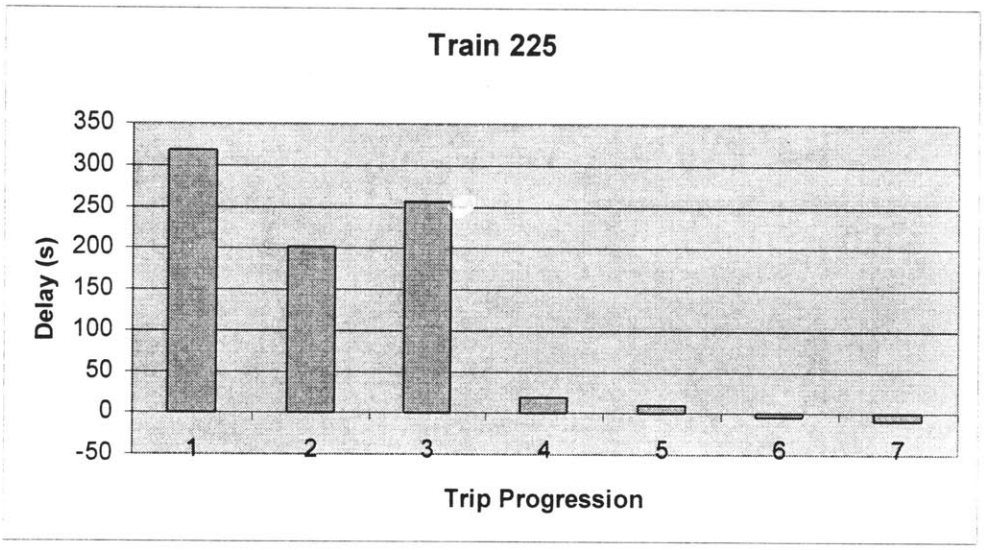


Fig. 7.1q

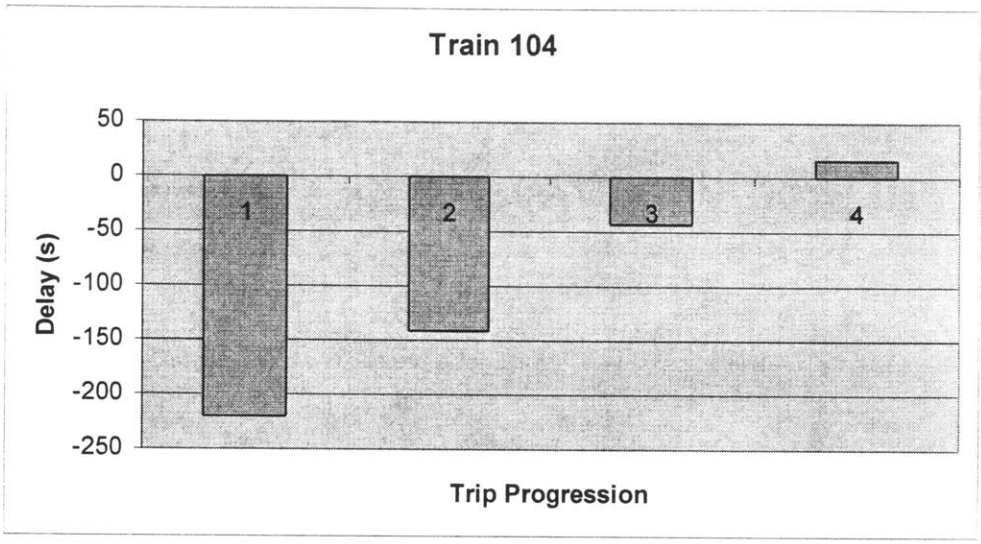


Fig. 7.1r

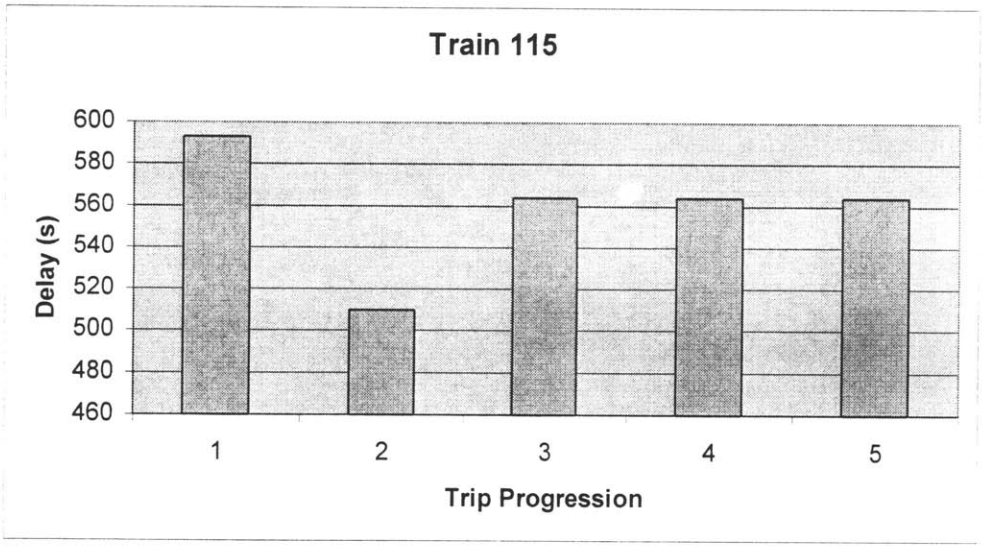


Fig. 7.1s

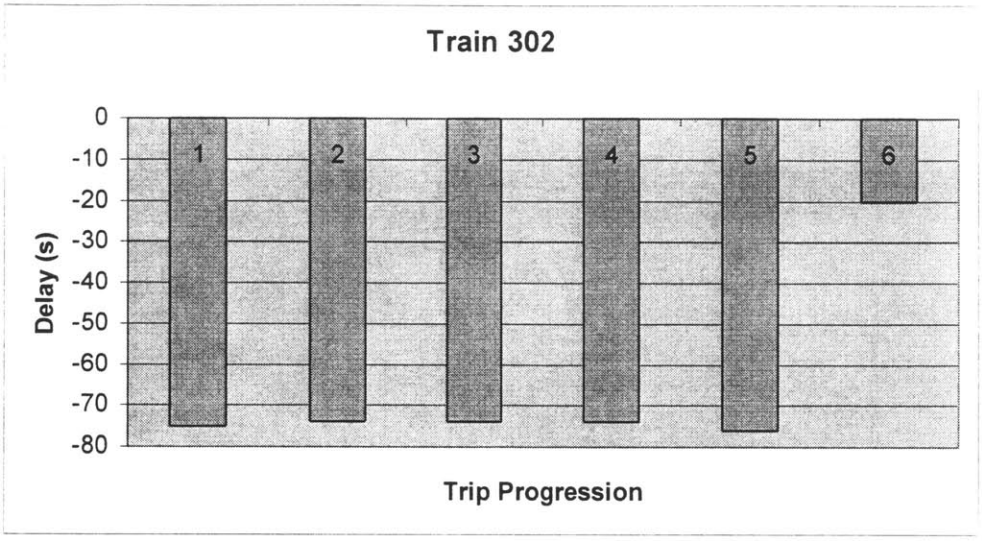


Fig. 7.1t

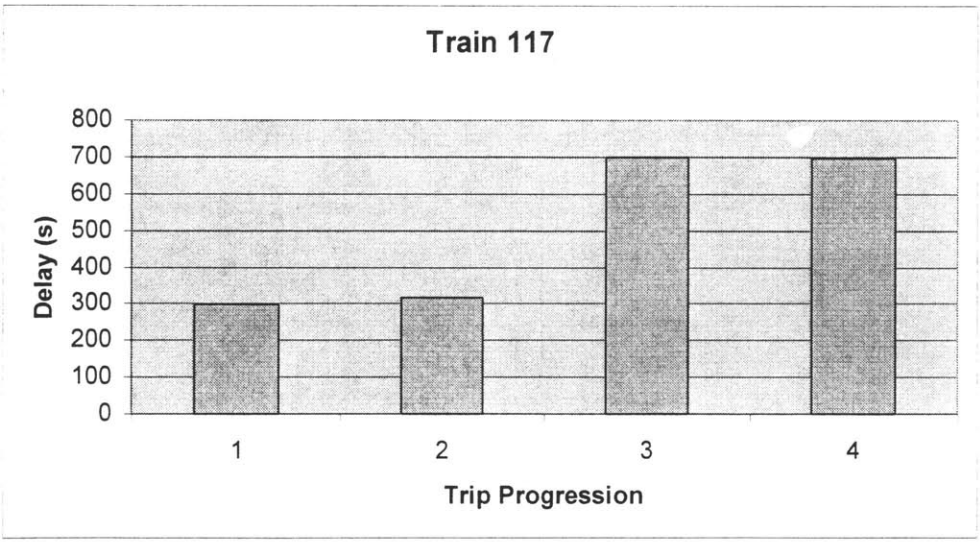


Fig. 7.1u

The train delay charts, taken as a group, give a rough indication of how well the dispatcher is routing the trains, or alternatively, how difficult the nominal routing task is. This can also be viewed on a per branch basis, for instance by looking at all 200-series trains. Viewed individually, many of the charts display a common characteristic. They tend to have abrupt jumps in the delay values. These are interesting because they indicate one of two things. If the jump is positive, it means the train was waiting before the

interlocking for a significant time. Usually this indicates that the interlocking was not cleared for the train to pass. This can be confirmed by referring to the raw data file, which records when each interlocking was cleared. If the jump is negative, it means the train made up time after leaving a station.

While the delay charts are useful in tracking routing efficiency in a data-link vs. radio experiment, they can also be valuable when studying line capacity, which is another potential use for this simulator. Theoretical models can be tested by scripting a scenario and studying the delay charts of each train.

7.6 Qualitative Observations

The qualitative information gained by using this simulator was in a way the most useful because it was not restricted to a pre-determined set of measurements. Some of it resulted from observations made during the simulation, and some from comments made by dispatchers before and after the simulation.

Because the participants in this project were dispatchers who currently work at CETC dispatching center, their reaction to the simulator interface was especially interesting. On a very promising note, they all took the simulator seriously. There was always a possibility that they would treat it entirely as a game, because it was not realistic enough in appearance or in operation. If this had been the case, then it would invalidate most of the information that the simulator provided, because presumably the dispatchers would not put the same amount of effort into the simulated tasks as they would on their job.

It is also interesting from a human factors perspective to understand why they did take it seriously. Each participant was told, at least once, that this was roughly based on the CETC system, but greatly simplified. However, most of them made comments indicating that they thought the track display screens were more or less consistent with what they work with daily. It appears therefore that the visual appearance of the simulator was more important than its operational realism. If this is true, then future dispatching

simulators should place a premium on visual realism, without of course sacrificing operational realism.

Most of the dispatchers made some comments that were strikingly similar. They all said that the data-link messaging system would be greatly improved if a set of audio cues were triggered when messages of different priorities or category arrived at the console. This perhaps reflects their radio dispatching background. It is a suggestion that should be incorporated into future versions of the simulator or into other simulators testing data-link systems.

Most of them also commented on the assignment of priorities to the various messages, saying that they would have assigned them differently. The simulator architecture was designed to allow the priorities to be changed very easily, in anticipation of this kind of comment.

Most dispatchers also commented that the workload was far in excess of what they were accustomed to in CETC. However, upon questioning them, they said the reason was not that each individual task took longer than at the CETC, but that there were far too many tasks to complete. This reaction was anticipated. In fact, the scenarios were designed so that dispatchers would make some mistakes, thereby providing a means of comparison between data-link and radio. This topic is discussed in greater detail in the Malsch's discussion on scenario design.

Some observations were made by us, the experimenters, during the dispatching runs. Like human-in-the-loop aspect of the simulator, the experimenter-in-the-loop architecture worked well, and it proved indispensable in the end. Several situations arose that were unexpected, and which the simulator was not designed to handle. One typical example of this occurred a few times on the right side of the territory where trains were entering from the imaginary adjacent territory. Sometimes, the dispatcher would be very busy and would neglect to route those trains. When one train remained blocked behind an interlocking and another train entered the simulation on the same track, there was a

danger of collision. This resulted both from the dispatcher's heavy workload, and because he was unable to view a global territory map as he could at the CETC. In these cases it was our job as experimenters to try and avoid these collisions. If they became unavoidable, we would create a communications sequence that made sense given the context. This kind of behavior was not pre-programmed into the simulator and required an experimenter-in-the-loop architecture. Most importantly, the experimenter-in-the-loop proved to be a transparent model, which means the dispatcher participant was not aware which portion of his stimuli was being created by the computer and which portion originated with us. We were pleased with its performance and would recommend it as a useful simulation model.

8 Conclusion

The focus of this work was to design, implement, and validate a real time railroad dispatching simulator, as well as the associated simulation model; this task was successfully accomplished and the resulting simulator provides an environment in which to test data-link theories.

The simulation model was based on standard human-in-the-loop principles, supplemented by the addition of a variable level of flexible experimenter interaction. This modified model, called experimenter-in-the-loop simulation proved to be a valuable way to simulate a complex, multiple human, multiple machine environment. It allowed human experimenters to intervene in situations that required a substantial amount of sophisticated behavior that could not be produced by simulated elements. It also proved transparent to the experimental participants.

The simulator was validated as an experimental tool by applying it to the study of a simple data-link system. It provides a level of flexibility that allows it to simulate situations that would be considered unsafe in the real world, and would therefore be difficult to study. Experimental participants, experienced dispatchers from Amtrak's CETC center, reacted favorably to the simulator. Their comments indicated that the simulator is visually realistic. On the other hand, they indicated that it lacked the same operational complexity of CETC. The simulator architecture detailed in this report was designed to be flexible and to allow modifications based on comments such as these.

Future applications of the simulator include further testing of data-link systems, one of which is currently under development. Additionally, it can be configured to study theories about line capacity, meet/pass efficiency, and traffic management.

Appendix A Simulator Architecture

It is not essential to read this section on the simulator software architecture to use the simulator either as an experimenter or a dispatcher. It is geared more toward those who want to understand the existing software at a developer's level, so that they may extend its capabilities. Therefore, it is assumed that the reader is already familiar with the basic vocabulary of software development, particularly object-oriented software development.

Because the system is rather large, its architecture is discussed in terms of layers, one built on top of the other. The method of layers is a very common way to describe software systems, with the lower layers being the basic building blocks, with a very general purpose. Upon these are built the higher layers, which are typically more specific in their purpose. Most of the time, the lowest level layer is the hardware layer, while the highest level layer is the interface between the user and the software. Most often, the majority of communications are between a layer and the one directly below it, or with other layers at the same level, but sometimes communications may span several layers.

The layer distinctions adopted to describe this simulator are as follows. First comes the hardware layer, which includes the computers, network cards, network, monitors, etc. Upon this is built the transport layer, which is the means of packaging data and exchanging it between computers. Next is the virtual machine layer, which is essentially the Java Virtual Machine (JVM), and some supporting software. Above this are several layers that are considered to be on the same level; these are the timing, recording, data, and display layers. On top of these lies the simulation layer and messaging layer; again, both are considered to be at the same level. Finally, at the highest level is the interface layer.

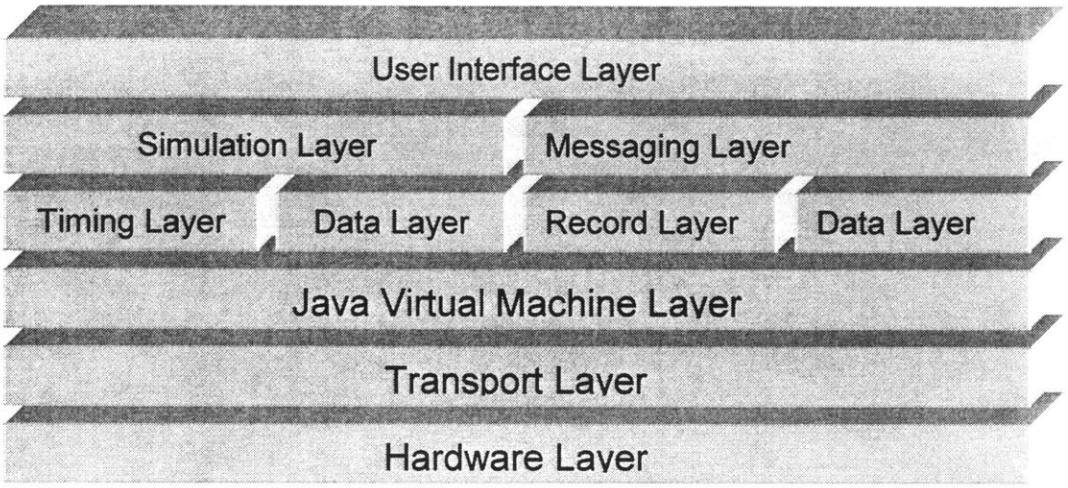


Fig. A.1 Layered Architecture of Simulator

The rationale behind using this layering scheme to describe the architecture is that extensions to the software will most likely only involve the interface and simulation layer, and perhaps the messaging layer. It is unlikely that any layer below these will need to be modified in a substantial way, unless dramatically different capabilities are required. Thus, over half of the code need not be touched, and this makes the job of extending the simulator much easier.

The rest of this section will describe in detail each of the layers. While the lower layers are discussed at some length, more emphasis will be placed on the higher layers since they are specific to this simulator. For the purposes of discussion, some terminology has to be agreed on. The words used most often that are a frequent source of confusion are defined as follows:

JVM: an environment in which java code is executed
Local: existing on the same JVM
Remote: existing on a different JVM
Local Access: requiring no communication outside the JVM

Remote Access: requiring communication between JVMs

Synchronized Access: essentially, "one at a time" access to a shared resource

Centralized: located in a single location within the simulator

Localized: located at many points throughout the simulator

A.1 Hardware Layer

This simulator currently runs on a group of Pentium machines. Their clock speeds are 350MHz and 400MHz. There is also a slower machine of the 486DX class running at 66MHz. In all, there are six machines, which happen to correspond directly to the six parts of the experimenter and dispatcher stations described above. Of course, other types of machines could be used. The recommended setup (due to performance reasons) includes four Pentiums with clock speeds of 300MHz or higher, and two Pentiums with clock speeds of 100MHz or higher. It will become apparent in the sections that follow, that it is possible to make up for one slower than average machine, with another faster than average machine, due to the component based design of the higher layers. Also note that, because the software components are written entirely in Java, it should be possible to entirely replace the hardware layer with one of equal performance, but base on another chipset (for instance, using Sun Sparcs instead of PC's). However, this has not been tested, and there would likely be some minor bugs to work out.

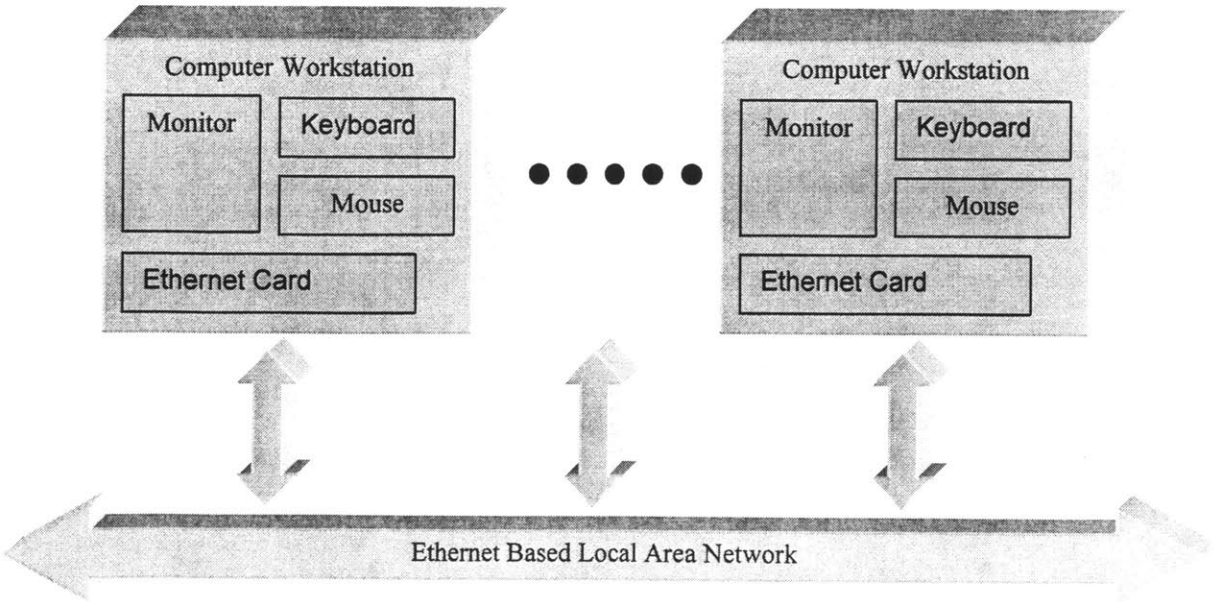


Fig. A.2 Hardware Configuration

Each computer is connected to a local area network via an ethernet card capable of 10BaseT data transfer rates. The cards used in the current setup are either 3C905 or 3C509 series cards from 3Com.

Inputs are accomplished via standard mice and keyboards. Each computer is connected to a 17 inch SVGA monitor. Hard drive space is not really an issue since the simulator software itself takes only two megabytes of space, while the Java Virtual Machine occupies another few dozen, depending on the version. Memory capacities on the machines range from 64MB to 128MB. It would be sufficient to have 64MB on all machines, and although it has not been tested, the simulator will most probably work fine with as little as 16MB of RAM per machine.

A.2 Transport Layer

This layer is actually made of three sublayers, called IP (Internet Protocol), TCP (Transmission Control Protocol), and RMI (Remote Method Invocation). Strictly speaking, RMI is part of the JVM, but since it conceptually is closer to a transport mechanism, it is included here instead.

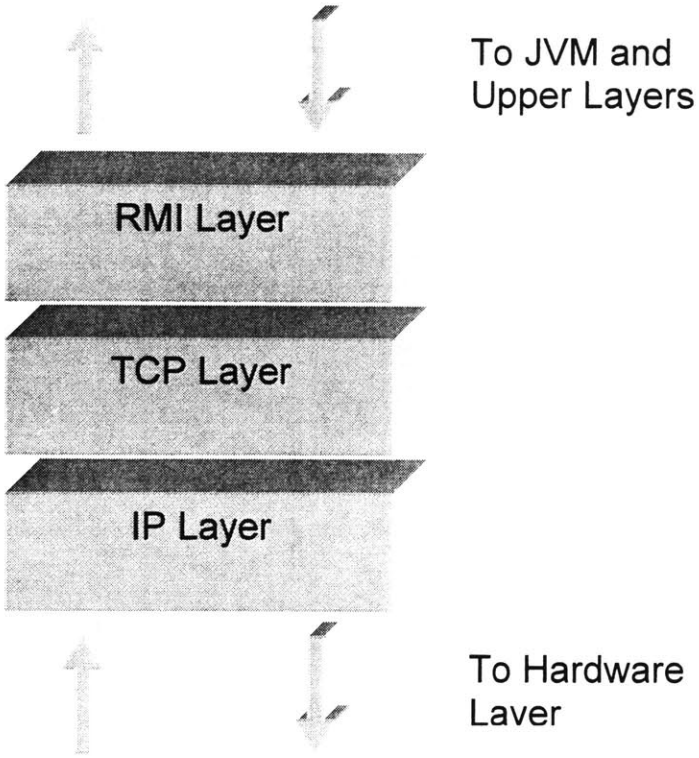


Fig. A.3 Transport Layer Detail

IP and TCP are standard protocols in the world of networking, so they will not be discussed too deeply here. It is sufficient to say that IP is at the lowest level. It is a quick and unreliable way to send bits of data across the network. It is rare that IP is ever used directly without building another layer on top of it. One such layer is TCP, which is somewhat slower, but guarantees delivery of data to some extent. Very many, detailed discussions of these protocols can be found on various web sites.

Upon these two is built RMI, a transport protocol developed by Sun Microsystems as part of their Java programming language [3] [11] [12]. RMI is more than simply a transport protocol; it also allows the remote calling of methods (functions in objects) from one machine to another. This is a great advantage to a simulator such as this, because it is essentially a group of interacting components spread across several machines. Without using RMI, it would be necessary to develop a complex messaging protocol that would

remotely invoke the appropriate methods. This is time consuming and potentially error prone.

On the other hand, RMI is slow. In some cases, it appears to be almost and order of magnitude slower than TCP/IP. A significant portion of the effort in designing the higher level layers went into thinking about how the advantages of RMI could be leveraged while maintaining a minimum level or performance.

In any case, RMI requires a support tool, called the RMI registry. This must be run prior to running any software that makes use of the RMI protocol. The program itself can be found in the /jdk1.1.x/bin directory (see the directory structure section on where to find files). There are some issues to keep in mind when using this registry. There must be an entry in the "autoexec.bat" file which reads "set classpath=c:\<project directory>" where <project directory> is the root directory for the files associated with the simulator. That happens to currently be "c:\kawa\projects." Additionally, there is a bug in the registry which will sometimes cause the error message "Invalid class found in stream" or something similar. If, while extending the simulator, no bugs can be found in the source code, it is worthwhile to shut down the registry and restart it; this may solve the problem.

For a further description of RMI, refer to the web site www.javasoft.com.

A.3 Virtual Machine Layer

This simulator requires that the JVM be installed on all the computers. This installation of compatible versions of the JVM comprises the virtual machine layer. The simulator was developed using the 1.1.5 – 1.1.7 versions of the JVM. Any of these versions in any combination can be installed. With few or no modifications, versions 1.2 may also be used. It was not used for the initial development since it was still in beta release at the time the project began. In any case, the features specific to version 1.2 are not used.

This layer is, in essence, the java programming language API's, so it is necessary to run the programs. But it also provides a useful decoupling mechanism between the hardware and the simulator software. The simulator does not directly access any hardware, but does so indirectly through the JVM. This may not seem helpful at the moment, but it makes the job of porting the simulator to another platform very easy, if ever this is required. The necessary changes to the code would be minimal, and there would also be very few changes to the data files (described in the following sections).

A.4 Timing Layer

This is the first of the layers built from the ground up specifically for this simulator. The purpose of this layer is to provide a basic timing system with which various components can synchronize themselves. This ensures that events that are happening at the same time in reality are actually happening at the same time as far as the simulation goes. It provides a means for "stopping time", speaking of course of the simulation timeline. There are two halves to the timing functionality: the central timer, and the synchronized remote timers.

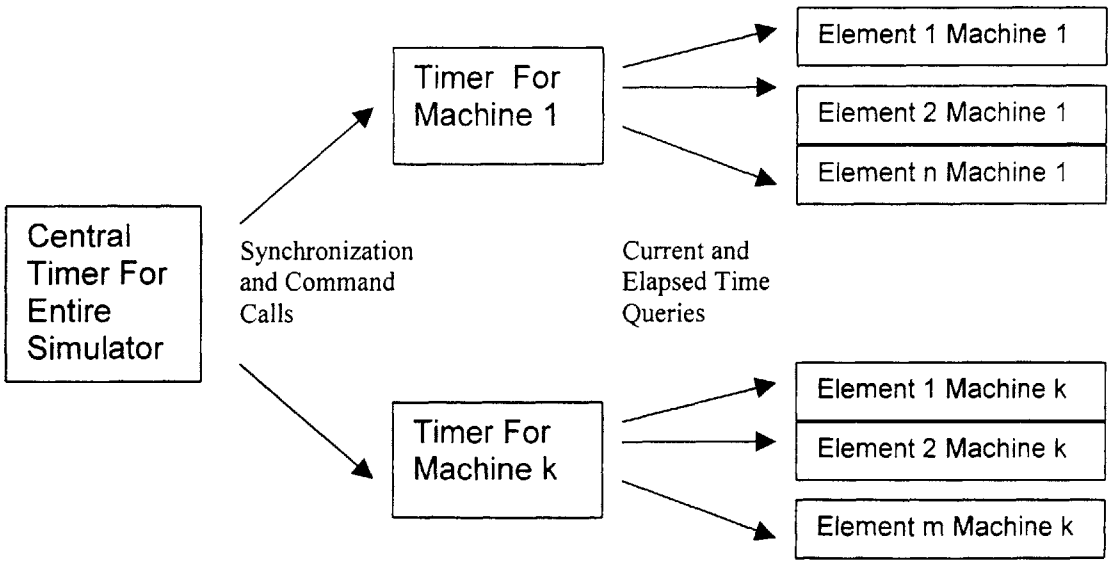


Fig. A.4 Timing Mechanism

A.4.1 Central Timer

The functionality of the centralized timer is implemented by an instance of the timeServer class. It maintains an internal reference to the system clock, which tracks time in terms of the number of milliseconds elapsed since a fixed date (which seems to be midnight GMT, 12/31/69). The time server maps a time t1 in this fixed real time frame to an arbitrary time t2 in the simulation time frame. This mapping is made at the beginning of the simulation. During the simulation, the central timer can be halted and resumed; this can be done repeatedly. The current and elapsed time can also be queried. The current time is simply the start time (t2) plus the elapsed time. The important thing to note is that the current and elapsed time are in the simulator's time frame. To illustrate the distinctions, assume that the simulator is started when the real world time is 4:00 PM, and the simulation starting time is arbitrarily set to 1:00 PM, denoted as (4:00/1:00). Now, if the simulator is halted at (5:00/2:00) for 30 minutes, then when it is resumed the time pair will be (5:30/2:00). This shows how the simulation time frame does not move forward during a halt.

Because the simulation time frame does not move forward, neither does any object synchronized with the central clock. In effect, the entire simulation freezes for as long as is necessary. This ability can be used to conduct situation awareness tests for the participants in the experiment, provide a break in an exceptionally long test run, etc.

A.4.2 Synchronized Remote Timers

There is only one central timer, but there is any number of remote "mirror images" of this timer. There is at least one, and probably several, on each computer. They exist so that there is no need to continually refer to one timer, a method that could become slow and relatively inefficient. These remote versions behave in exactly the same way as the central version. Whenever a command is issued to the central timer (be it start, halt, or resume), the command is copied to each of the remote timers.

There is an issue of time lag in transferring these commands. Although this lag is no more than about a second, it was eliminated in the following way. When the start, halt, or resume commands are issued, they are not assumed to take effect immediately. They are issued with a "standoff" time, which means they will execute after a specified number of seconds. This means that the command can be issued centrally and relayed remotely, but no action is taken until the standoff time has passed. Then, all the commands are executed simultaneously.

A.5 Recording Layer

The recording layer is a means by which the activity of the simulation can be recorded to disk in real time, then analyzed later. The recording mechanism is arranged in the form of one global repository for each machine, to which all parts of the simulator can write records. In the interest of speed, the objects writing these records do not write directly to the record store, but to a fast temporary buffer. Within a few seconds, the recorder will clear the buffer and write these to a file. At the end of a simulation, there is one file on each machine with the ".rec" extension, and a name specified at the beginning of the simulation. It holds the raw recordings of the simulation. These are not readable by humans, but can be converted into structured text files using a secondary utility implemented by the recordTools class. Before discussing the file structures, a description of how records are created is helpful.

When an event occurs in the simulation that needs to be recorded, a few lines of code are executed to create the record. An object of one of the subclasses of the record class is instantiated, and any data required to define the record is passed to the constructor. The only piece of data that is required by all records is the timestamp, which is obtained from one of the remote timer objects described above. Other data varies from record to record. The record object is then sent to the recorder. When it decides to write the record to file (which is very soon after it is given the record) it will use the serialization mechanism of Java to write the record to disk as an object.

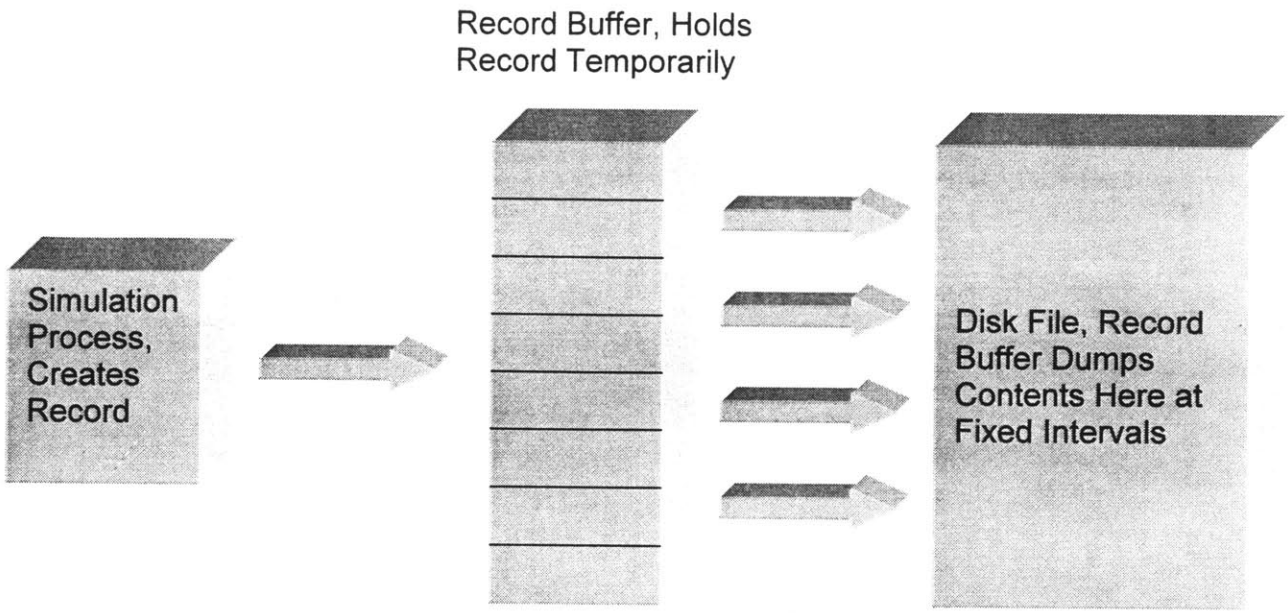


Fig. A.5 Recording Mechanism

This object will appear in the file as text interspersed with other symbols specific to Java's serialization protocol. In general, the file will not be readable. However, a secondary file can be created by using the recordTools class. It will take the original record file and reread the record objects stored there into memory. Then, it will ask the records to render themselves into a human readable format (every subclass of record must implement functionality to accomplish this; it is the only requirement of subclassing the record class). Then, the human readable text is written back into a file with the same name as the original record file, but this time with a ".txt" extension.

The structure of this file is linear, and ordered chronologically. Each entry contains a timestamp, the type of record, and a textual or numeric description of any data contained in the record. The format is easy to review, but the files can potentially become very long. For simulations longer than about one hour, it may be necessary to process the record files even more, separating types of records into different file. This feature is not presently implemented.

One last feature that is necessary is the merging of record files. Because there is one file for each computer, there will be several record files when the simulation is finished. Each is chronologically ordered, but they must be combined so a complete picture of the simulation progression can be drawn. The recordTools class implements a tool that allows the merging of two record files. More files can be successively merged into the resulting file, until all the files have been merged into one.

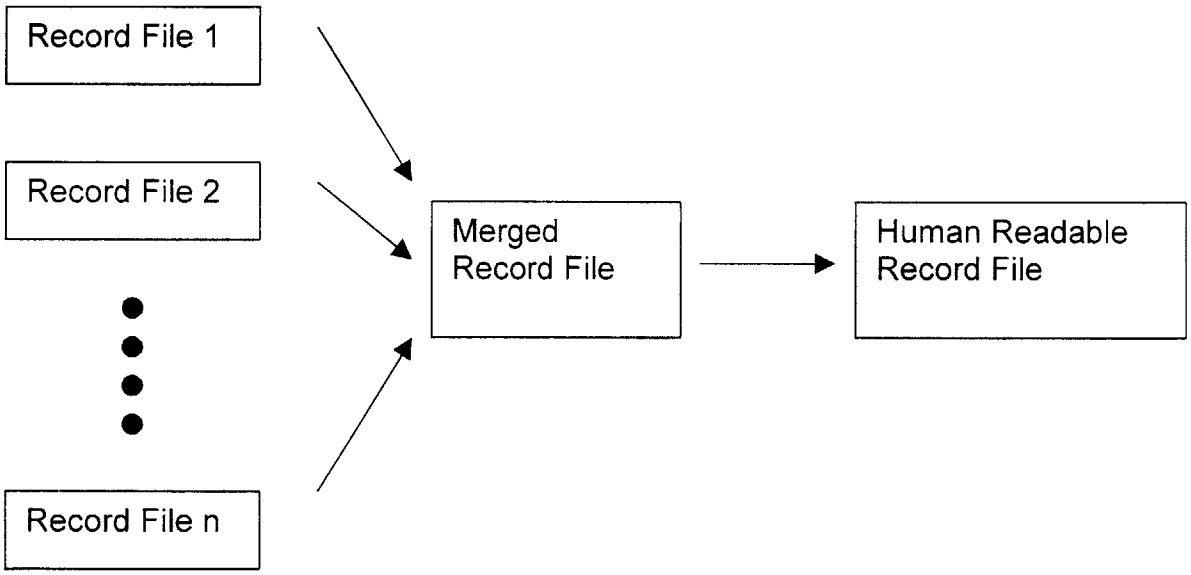


Fig. A.6 Record File Post-Processing

A.6 Data Layer

This layer is one of the most crucial parts of the simulator, but also one that is not likely to be modified much in the future. It was also one of the most difficult parts of the simulator to program. The difficulty arose because the elements of the simulator needed fast, local access to consistent centralized data. That is what the data layer provides. Before describing how it does this, it is useful to understand why it is necessary.

A.6.1 Simulation Requirements

Much of the data referenced in this simulator is used to model the activities of objects in real time. Therefore, the data are accessed very often, by many different objects. This

precludes a totally centralized system, where all the data are stored on one machine and accessed remotely when it is needed. Even without the RMI protocol, this kind of access pattern would be too slow. Moreover, the need to synchronize access to data (prevent different objects from modifying the same data at the same time) would likely result in a noticeable delay if many objects tried to get at the same data.

At the other end of the spectrum is a totally decentralized model where the data are copied as many times as necessary such that every object gets it's own copy. This becomes impractical as well because when one object makes a change to its data, the change must be reflected to all other copies; after all, the copies really describe the same object. If each object has it's own copy, it becomes almost impossible to make all the necessary updates and guarantee consistent data.

A system is required that combines relatively fast access to read and write data, while maintaining data consistency at all times. This system must also be transparent to the objects accessing the data; they should be able to treat it as if it was the only copy available, when in fact it is not. The data layer fulfills these competing requirements reasonably well. It is based on a few simple components. There is a centralized data store which retains "master copies" of data objects. There are any number of local data stores that maintain "mirror copies" of these objects. And the objects that want to be processed by the data layer must derive from a root class called localWorldObject. All other data are derived from the root class remoteWorldObject

A.6.2 Central Data Store

The class localGroupServerImpl implements the functionality of the central data store. This central store is never used directly to access data, but it serves several other important purposes.

The central store is aware of all local data stores. When a new local store is created, it contacts the central store to register itself. By doing this it is requesting that it should be notified of any change that is made to the centralized data.

The central store always maintains a copy of all the data available to the simulation. At the beginning of the simulation much of this data is read from disk. When local data stores first register themselves, they are given all this data. It is even possible that a local store could be created during the simulation and recieve the data then.

A.6.3 Local Data Store

Typically, there is one local data store per JVM. Access to the data in this store is global within the JVM, so there is no need to maintain special references to the store. When they are set up initially, they register them selves with the central store, as mentioned. Whenever a piece of data is changed, the change is reflected through the central store to all the other local stores. The mechanism of how this is done is described in the next sub-section.

The overall effect of this system is twofold. The data can be accessed quickly, without any reference to the central copy. But if a change is made in the local copy, the central copy must be notified. The system lends itself to applications where data is read very often, but modified less frequently. That is exactly the type of application that this simulator is.

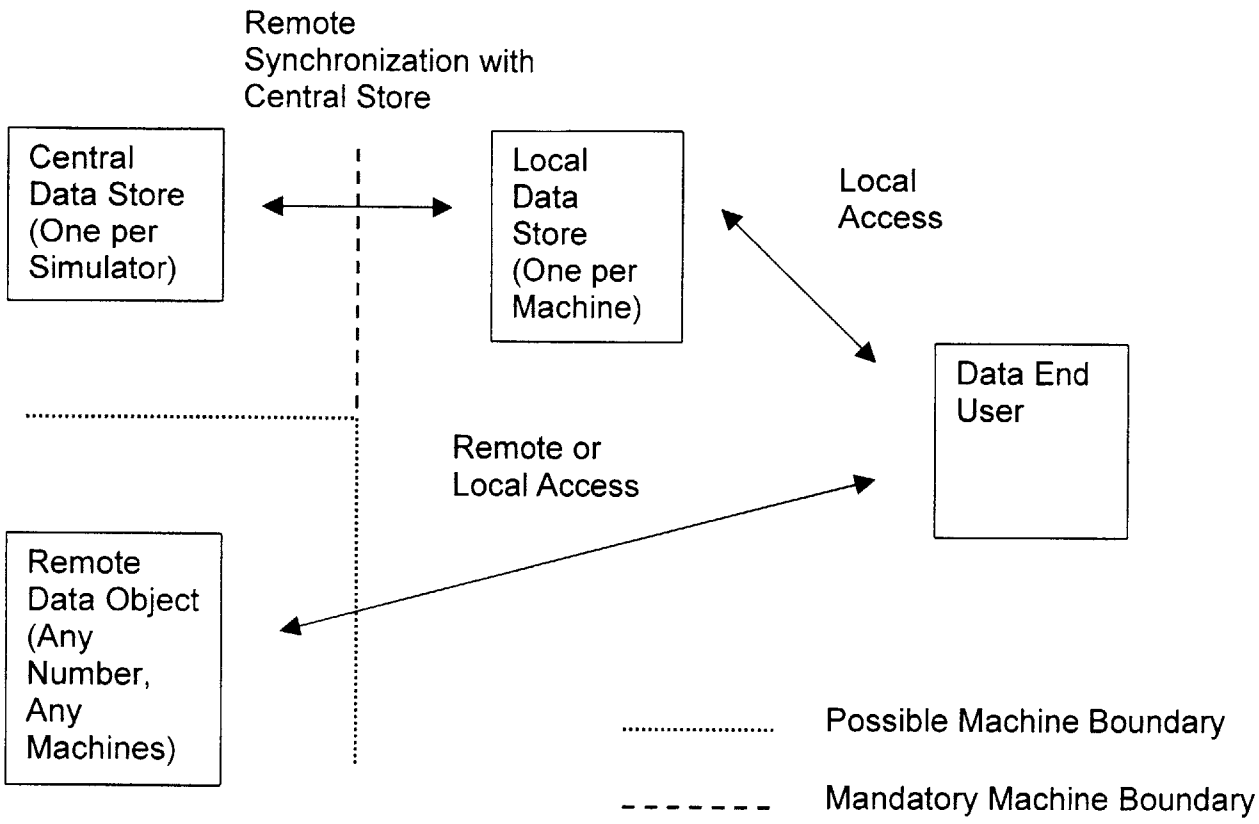


Fig. A.7 Data Layer Schematic

A.6.4 Local Objects

The term "local objects" is used to denote data objects that exist as multiple copies, several locally, and one centrally. This is in contrast to remote data, which exists as a single copy over the entire simulator and is accessed remotely when needed. All local data is implemented by classes derived from the class localWorldObject. The most important feature of these classes is their ability to automatically make an update call to the local data store when any part of their structure changes. For example, the local object used to model track makes an update call when the speed limit of the track changes as a result of aspect restrictions.

The update call is what initiates the data consistency mechanism across the entire simulator. The local data store becomes aware that a data object has changed. It then contacts the central store and notifies it of this fact. The central store then contacts all the other local stores and notifies them. Finally, they make the change to their copies of the data.

This whole process is transparent to the process that initially made the change, since the object being changed initiates the update call, not the process itself. The only requirement is that all the local objects used in the simulator (the existing ones, and the ones that may be created in the future) are well defined. This means that when there is a means provided to change data in the object, the update call is also made automatically. If this simple rule is not obeyed, then data consistency can no longer be guaranteed throughout the simulation, and the resulting data will be invalid.

A.6.5 Remote Objects

The complementary type of object is a remote object. It exists only in one place. In general, local objects are synonymous with the passive objects described in section 3.1, while remote objects are synonymous with active objects. Passive/active are functional descriptions while local/remote are architectural descriptions of the same things.

Because a remote object exists in only one place, its data must be accessed over the network. Thus, it is not practical to access the data frequently. An example of a remote object is a train (it is also an active object). It maintains a state variable to describe it's speed, heading, mass, etc. While some of these variables may change rapidly, such as the speed, other objects in the simulation will not be aware of this because they do not sample the values at frequent intervals.

The general rule to follow is that objects that need to be used by many other objects very frequently should be made local, but objects that access other objects frequently but are not themselves accessed frequently can be made remote.

A.7 Display Layer

This layer is responsible for automatically visualizing changes made to a simulation element, if that element's has a visual representation. An example of this automatic screen update is the switch. When a switch changes state, as the result of a routing operation, the screen image must be updated because the switch will probably connect different tracks then it did previously. The display layer accomplishes this update. Similar to the data layer, the display layer operates on the basis of a centralized information relay point working together with several local display points.

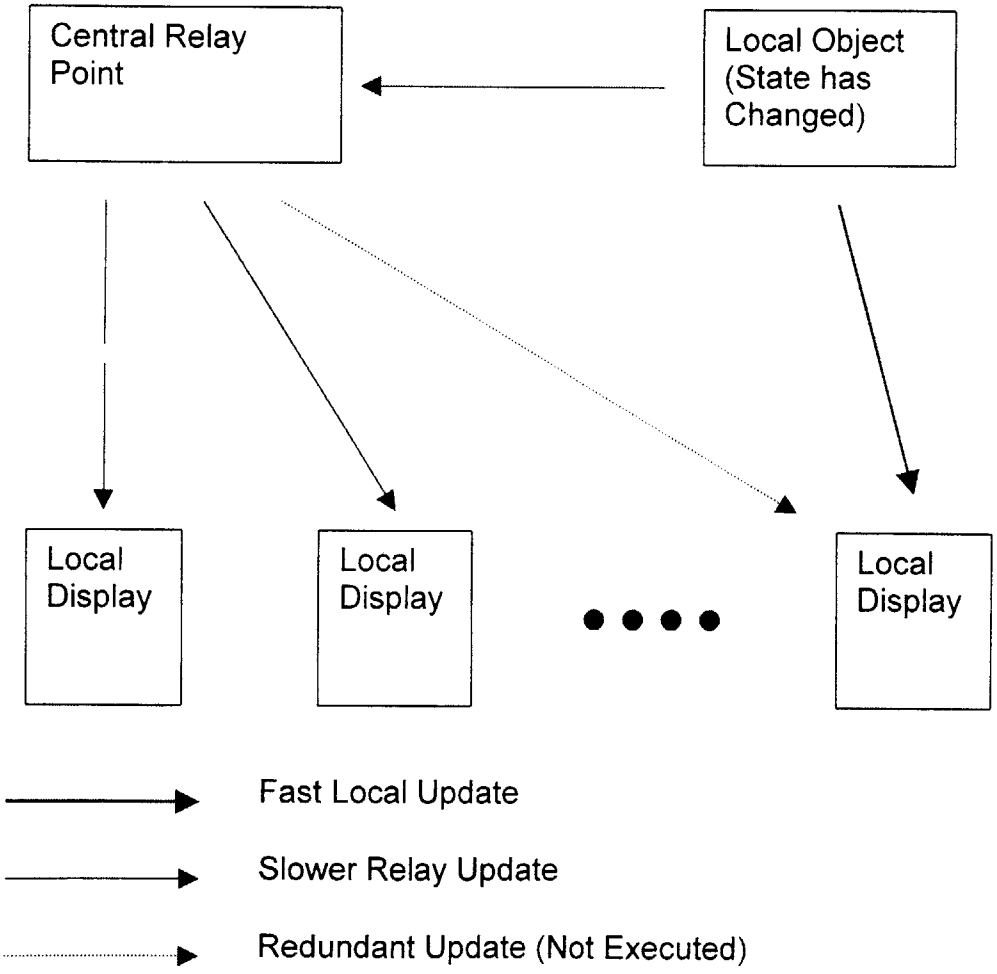


Fig. A.8 Display Layer Schematic

A.7.1 Central Relay

Whenever a local display is initialized, the first thing it does is register itself with the central relay point, of which there is only one. Later, when the visuals are updated on one machine, the change can be reflected to all the other machines. This is necessary because there is always the possibility that two local displays will overlap to some extent and display the same data. In these cases, the visual display must be consistent to avoid confusion.

When a change is made to an object such as a switch, the switch object automatically initiates two types of updates. One has already been mentioned: the update call to the local data store. The second takes care of the screen update.

This screen update itself has two parts. This first is the local update portion, which is described in the next section. The second is the central update portion. This notifies the central relay point that a visual change has been made on one of the displays. The central relay will then automatically reflect the change to all the local displays.

A.7.2 Local Displays

As mentioned in the previous section, there are two halves to a visual update. The second half is carried out locally. When an object changes its visual appearance, it notifies any displays that coexist locally on the same JVM as itself. This occurs independently of the central relay point. Thus, there seems to be two ways that a local display is updated. However, only on of these methods is used for any given display. If the display coexists locally with the object that has changed, it is updated locally. Otherwise, it is updated via the central relay point.

Incidentally, the local displays are encased in a larger window. This window directly implements the display screens of the dispatcher and experimenter terminals.

A.8 Simulation Layer

At the next higher level above the preceding four layers lies the simulation layer. It makes extensive use of all four of those layers to accomplish it's task. This is essentially to continually modify the states of simulation elements according to a set of rules and data files. This is accomplished in two ways, through state based simulation and event based simulation.

A.8.1 State Based Simulation

State based simulation is the process of altering the state of a simulated object by considering its current state and applying a set of rules to modify that state. It is independent of the time. To illustrate, the simulation of train dynamics under the force of the train's engine is state based. Its current velocity, engine traction force, local curvature, friction coefficient, etc. can be used to derive the same data at a later instant in time. It does not matter what the time is because the train will behave the same way as long as the state variables are identical. The next state values are obtained using a rule set, which in the case of the train, is the set of dynamical equations describing the train.

A.8.2 Event Based Simulation

This type of simulation is based on time. Certain events are programmed to occur at certain times, regardless of the state of their surroundings. This is accomplished by using event files, which are data files that contain definitions of events. Besides the time of occurrence and type of event, other data may be specified in each entry. The data from these files is read into a set of event objects, each an instance of a class derived from a root event class called scheduledEvent. Throughout the simulation, the events are executed at the specified times.

A.8.3 Combined Simulation

Event based and state based simulation can be combined to create situation based simulation. In this type of simulation, an action will be taken if a certain state is

encountered, similar to state based simulation. However, the state that triggers these actions will be created for only limited time spans, using event based simulation. In other words, things will happen in particular situations that are come and go. This type of simulation is very flexible, but it also suffers form one drawback: there is no way to guarantee that the state created will trigger the action.

As an example, consider this situation. An event is used to set the state of a piece of track to indicate that there are trespassers near the track. This is done at time t1, and remains there until time t2. The intention is that a train that passes that track will inspect the state of the track, determine there are trespassers there, and initiate an action. However, there is no way to guarantee that the train arrives at that track between t1 and t2. In reality, a situation based simulation can only create the probability that something will happen, but cannot ensure that it does happen.

A.9 Messaging Layer

The messaging layer has a very simple purpose: it allows communications via a standard messaging system between any two objects that declare themselves to be potential message recipients. A good analogy is a telephone directory. If you list yourself in the directory, you indicate to other people a means to contact you, and they may do so. The messaging layer works similarly. It provides a means for an arbitrary object to become "listed", and it provides a repository to store those listings, as well as a means of accessing them.

A.9.1 Becoming a Message Recipient

Notifying the system that a particular object can receive messages is a very simple procedure. The class of which the object is an instance must simply implement the messageable interface. Then, at some point, the object must register itself with the message registry.

A.9.2 Message Registry

The message registry is a listing of all the objects that have indicated that they can receive messages. It acts as a kind of switchboard. To illustrate, assume that object A wants to send a message to objects B. Also assume that object B implements the messageable interface. In this case, object A would contact the message registry, specify the name of object B, and supply a message. The registry would then retrieve the listing for object B (containing it's location) and send the message to it. Object B could then react to the message in any way it saw fitting.

A.9.3 Messages

The actual information exchanged via the messaging layer is described by a large set of classes that are derived from the root class message. This class can be extended in almost any way desired, and use to transmit an arbitrary amount and variety of data, while organizing that data in a standard manner. The description of how this works is complicated without making frequent references to the source code. Therefore, please refer to the source file message.java for a more complete description.

A.10 Interface Layer

This is the highest level layer and it is the one with which the user directly interacts. The visual layout and operation of the layer has already been described in considerable detail in section 4. The means by which objects are selected was only briefly mentioned. That facility is described in detail in this section.

Any object that has a visual representation can be selected by clicking on that image on the screen. A train can be selected by clicking on its ID number, a track can be selected by clicking on its image, etc. By selecting the object, it is made available to any task that requires an object as a parameter, for instance the execution of an operation, or the completion of a message.

The framework that allows the selection of objects using their on screen images (screen objects), even if these images move, is based on a system of zoning. This is discussed in the context of how a display screen is broken into zones, as well as how a screen object associates itself with one of these zones.

A.10.1 Screen Partitioning into Zones

Each display area is first partitioned into a regular grid. This is the most course zoning used and the rectangular zones involved are never used directly to locate features on the screen.

A.10.2 Categorical Sub-Partitioning

Each of the coarse rectangular zones is further partitioned into categorical zones, each of which handles certain types of objects. For instance, track zones are used to locate track images, while train zones are used to locate train ID numbers. These zones are a bit abstract since they do not actually have any physical dimensions. Rather they are simply groupings for the further sub-zones.

A.10.3 Spatial Sub-Partitioning

Within each categorical zone, there are multiple spatial zones. These divide the area within each course grid zone into non-overlapping shapes, each of which is considered to be a spatial zone. These are the smallest zones, and they are not subdivided any further.

A.10.4 Mapping a Zone to an Object

While the previous three sections have discussed three types of zoning, a zone is, strictly speaking, a fully qualified triple. That is, a fully qualified zone must specify the spatial zone, the categorical zone, and the screen zone.

Zones are mapped to screen objects under the assumption that there is only one screen object for each spatial zone. When a screen object is given to a display (when the local

display registers with the central relay, see section A.7) it is asked to create a list of zones. If it is not a screen object that wants to be selectable from the display, it is free to return a null set of zones. Otherwise, it must return the set of zones that will encompass its location on the screen.

These zones are then stored in the display, and each fully qualified zone points to the object that created it. When a user clicks on the display area, a test is done to see if the click falls within a zone, and if so, the zone is used to access the associated screen object. It is then considered to be selected.

At this point, it makes sense to explain the need for the three levels of partitioning. There are on the order of 1000 screen objects in a typical simulation, meaning than any one display will display several hundred. If all of them create only a single zone to locate themselves, it would require a check of hundreds of zones each time the mouse was clicked. This is impractical.

To solve this, the zones to be checked are narrowed down in a hierarchical manner. First, the mouse click is located inside one of the top level screen zones. This is an O(k) operation because the screen zone names are obtainable directly from the position of the click. Then, within a single screen zone, the categorical zones are checked. This is also an O(k) operation because the category of object that needs to be selected is known ahead of time (e.g. route clearing operation requires the selection of a poke point). Lastly, the mouse clicked is located within one of the spatial zones. This requires a check on about 10 zones on average, which is computationally manageable.

When screen objects move, they are asked again what zones they wish to be mapped to. The screen object is responsible for updating the position and/or type of zones it associates itself with so that it will remain selectable in it's new position.

A.11 Directory Structure

There are two main directory hierarchies involved in the train simulator at present. While the source files can be configured to use other directories, it is simpler to just keep the structure as it stands now.

The first branch starts at c:\jdk1.1.x where x anything between 5 and 7 inclusive. Only the subdirectories pertinent to the simulator are listed, although there are others. It is organized as:

+ jdk1.1.x
    - bin (the directory containing the compiler, rmi-compiler, and interpreter)
    - docs (the directory containing the Java documentation)

The second branch starts at c:\kawa, and deals with the simulator source, data, and record files. It is organized as:

+ kawa (the directory containing the IDE used for this project)
    + projects (root directory for all source, data, record files)
        + railsim (all source code excluding message set)
            - messages (source code for message set)
        - railterm (train data files)
        - railserver (physical, visual, and event data files)
        - records (recordings of simulator)

A.12 Data File Formats

The various data files used by the simulator adhere to fixed formats. There are currently four categories of data files: physical information files, visual information files, event files, and train files.

A.12.1 Physical Information Files

The purpose of these files is to provide the simulator with physical descriptions of the passive objects that constitute the simulation environment. Examples of this type of data include the dimensions of track segments, the organization of interlockings and stations, etc. Within this category of file, there are always at least two files used by the simulator. The reason for this is maintaining compatibility between this simulator and another simulator the models the train engineers perspective. The only difference between the two files is that the first one is restricted to listing objects that are found in the other simulator, while the second can list all objects including ones specific to this simulator. In either case, the format is the same, based on the following pattern. The '<' and '>' symbols are not actually typed in the data file; text that appear between them should be considered variables that can take on different values.

<class name> <instance number>
<tag 1> <value 1>
...
<tag n> <value n>

This structure can be repeated any number of times, but there must be a line containing only the word "end." For each instance of this structure, the information is interpreted in the same way. The class name specifies one of the subclasses of the localWorldObject class. A new instance of the class is created, and the data is read from, one data member at a time, until the new instance has been initialized. Each tag is a character string, and each value can by anything (text, numerical, or other formats). The newly instantiated object will interpret the data as it is programmed to. To see what tag is mapped to what data member of the each class, it is simplest to just read the documented source files.

A.12.2 Visual Information Files

The visual information files serve to define how the objects found in the physical information files will appear on screen. In some cases, the physical objects have no

screen representation. For those that do, this file provides the data necessary to display them. For each physical information file, there is one of these files, with the same name, but a ".dis" extension. They are based on the following structure.

obj <full object name>
<tag 1> <value 1>
...
<tag n> <value n>

This structure can be repeated as many times as needed, and again, the last line must contain only the word "end." The tag-value pairs work in the familiar way, but the object is instantiated differently. The physical object found in the first file type will declare a screen object to represent it on screen. It will assign a name to this screen object based on a rule (see the documented source files for how this is done for each object). This name goes in the <full object name> field of this data file. It is not necessary that the order of the screen objects match the order of their respective physical objects. As long as each screen object declared by a physical object is defined in this file, the order is not important.

A.12.3 Event Files

These files define the events used to create the simulation capability described in section A.8.2. There are various types of events, all derived from scheduledEvent, that can be defined in this file. See the source files for a description of each event. In the event file, there is one entry, structured as follows, for each event that will occur.

<event class>
<tag 1> <value 1>
...
<tag n> <value n>

The event class specifies which event is to be created. There is no need to distinguish different instances of the event, because they are not referred to directly. An instance is simply created, and initialized with the data defined by the tag-value pairs. Again, see the individual event source files for a description of what tags and data is required in this file. The newly created event is then placed in the event scheduler, which executes it at the time specified.

A.12.4 Train Files

The train files are used to define the initial positions and schedules for the trains in the simulation. They do not contain data for any other objects. Each entry starts with:
<train class> <train number>
where the train class is currently lumpedTrainImpl, but in the future could be any other train class derived from the root class trainImpl. The train number is a unique identifier, but this is not the same as the train's identification number. The detailed structure of the data that follows is specified in the source files lumpedTrainImpl.java, trainImpl.java, and schedule.java.

Appendix B Train Schedules

The train schedules for both scenarios are included in this appendix. These describe the nominal paths the trains travel, specifying both locations and times. Along with the territory maps and the train delay charts, these schedules provide a picture of how trains actually moved, where they were held up, and where they made up time.

Train Schedule for Scenario 1

Br.	Dir.	Train #	Ptl	Terminal		Interlocking		Interlocking		Interlocking		Interlocking		Interlocking	
BRANCH A	IN		1	1:26 PM	A1	1:23 PM	A2	1:21 PM	A3	1:14 PM	A4	1:07 PM	A5	1:00 PM	A-t3
			2	1:46 PM	A1	1:43 PM	A2	1:41 PM	A3	1:34 PM	A4	1:27 PM	A5	1:20 PM	A-t4
			3	2:17 PM	A1	2:14 PM	A2	2:12 PM	A3	2:05 PM	A4	1:58 PM	A5	1:51 PM	A-t2
	OUT	111	1	12:39 PM	A1	12:42 PM	A2	12:44 PM	A3	12:51 PM	A4	12:58 PM	A5	1:05 PM	A-t2
		113	3	12:58 PM	A1	1:01 PM	A2	1:03 PM	A3	1:10 PM	A4	1:17 PM	A5	1:24 PM	A-t1
		115	2	1:27 PM	A1	1:30 PM	A2	1:32 PM	A3	1:39 PM	A4	1:46 PM	A5	1:53 PM	A-t2
		117	3	1:34 PM	A1	1:37 PM	A2	1:39 PM	A3	1:46 PM	A4	1:53 PM	A5	2:00 PM	A-t1
BRANCH B	IN		6	1:42 PM	B1	1:38 PM	B2	1:32 PM	B3	1:27 PM					NS
			4	2:20 PM	B1	2:16 PM	B2	2:10 PM	B3	2:05 PM					NS
	OUT	221	3	12:50 PM	B1	12:54 PM	B2	1:00 PM					B4	1:05 PM	BL-t1
		223	6	1:06 PM	B1	1:10 PM	B2	1:16 PM					B4	1:21 PM	BL-t1
		225	5	1:27 PM	B1	1:31 PM	B2	1:37 PM					B4	1:42 PM	BL-t2
BRANCH C	IN		7	1:21 PM	C1	1:17 PM	C2	1:13 PM	C3	1:10 PM	C4	1:05 PM	C5	1:01 PM	No
			7	2:00 PM	C1	1:56 PM	C2	1:52 PM	C3	1:49 PM	C4	1:44 PM	C5	1:40 PM	No
	OUT	331	8	1:08 PM	C1	1:12 PM	C2	1:16 PM	C3	1:19 PM	C4	1:24 PM	C5	1:28 PM	No
		333	4	1:42 PM	C1	1:46 PM	C2	1:50 PM	C3	1:53 PM	C4	1:58 PM	C5	2:02 PM	No
BRANCH D	IN		8	1:35 PM	D1	1:33 PM	D2	1:25 PM	D3	1:21 PM	D4	1:17 PM	D5	1:10 PM	D-t1
			9	2:20 PM	D1	2:18 PM	D2	2:10 PM	D3	2:06 PM	D4	2:02 PM	D5	1:55 PM	D-t2
	OUT	441	8	12:42 PM	D1	12:44 PM	D2	12:52 PM	D3	12:56 PM	D4	1:00 PM	D5	1:07 PM	D-t2
		443	7	1:04 PM	D1	1:06 PM	D2	1:14 PM	D3	1:18 PM	D4	1:22 PM	D5	1:29 PM	D-t2
Yard	IN	225EQ	5	ar. 1:15 PM		To terminal									
	OUT	300EQ	7	ar. 1:30 PM		To Yard									

Station		Interlocking		Interlocking		Interlocking		Station		Interlocking	F.D.	Train #	Dir.	Br.
12:50 PM	A6	12:56 AM	A7	12:49 AM							1	100	IN	BRANCH A
1:19 PM	A6	1:09 PM	A7	1:02 PM							2	102		
1:50 PM	A6	1:44 PM	A7	1:37 PM							3	104		
1:16 PM	A6	1:18 PM	A7	1:25 PM									OUT	
1:29 PM	A6	1:31 PM	A7	1:38 PM										
1:56 PM	A6	1:58 PM	A7	2:05 PM										
2:04 PM	A6	2:06 PM	A7	2:13 PM										
			B6	1:26 PM	B7	1:19 PM	BR-t2	1:17 PM	B6	12:50 PM	6	200	IN	BRANCH B
			B6	2:04 PM	B7	1:57 PM	BR-t3	1:55 PM	B8	1:50 PM	4	202		
1:10 PM	B6	1:11 PM	B6	1:13 PM	B7	1:20 PM	NS-t3	1:22 PM	B8	1:24 PM			OUT	
1:23 PM	B5	1:24 PM	B6	1:26 PM	B7	1:33 PM	NS-t3	1:35 PM	B8	1:37 PM				
1:45 PM	B5	1:46 PM	B6	1:48 PM	B7	1:55 PM	NS-t3	1:57 PM	B8	1:59 PM				
											7	300	IN	BRANCH C
											7	302		
													OUT	
1:09 PM	D6	1:03 PM	D7	1:01 PM	D8	12:58 PM					7	400	IN	BRANCH D
1:54 PM	D6	1:49 PM	D7	1:46 PM	D8	1:43 PM					9	402		
1:18 PM	D6	1:18 PM	D7	1:21 PM	D8	1:24 PM							OUT	
1:32 PM	D8	1:32 PM	D7	1:35 PM	D8	1:38 PM								
										ar. 1:15 PM	5	225EQ	IN	Yard
										ar. 1:30 PM	Yard	300EQ	OUT	

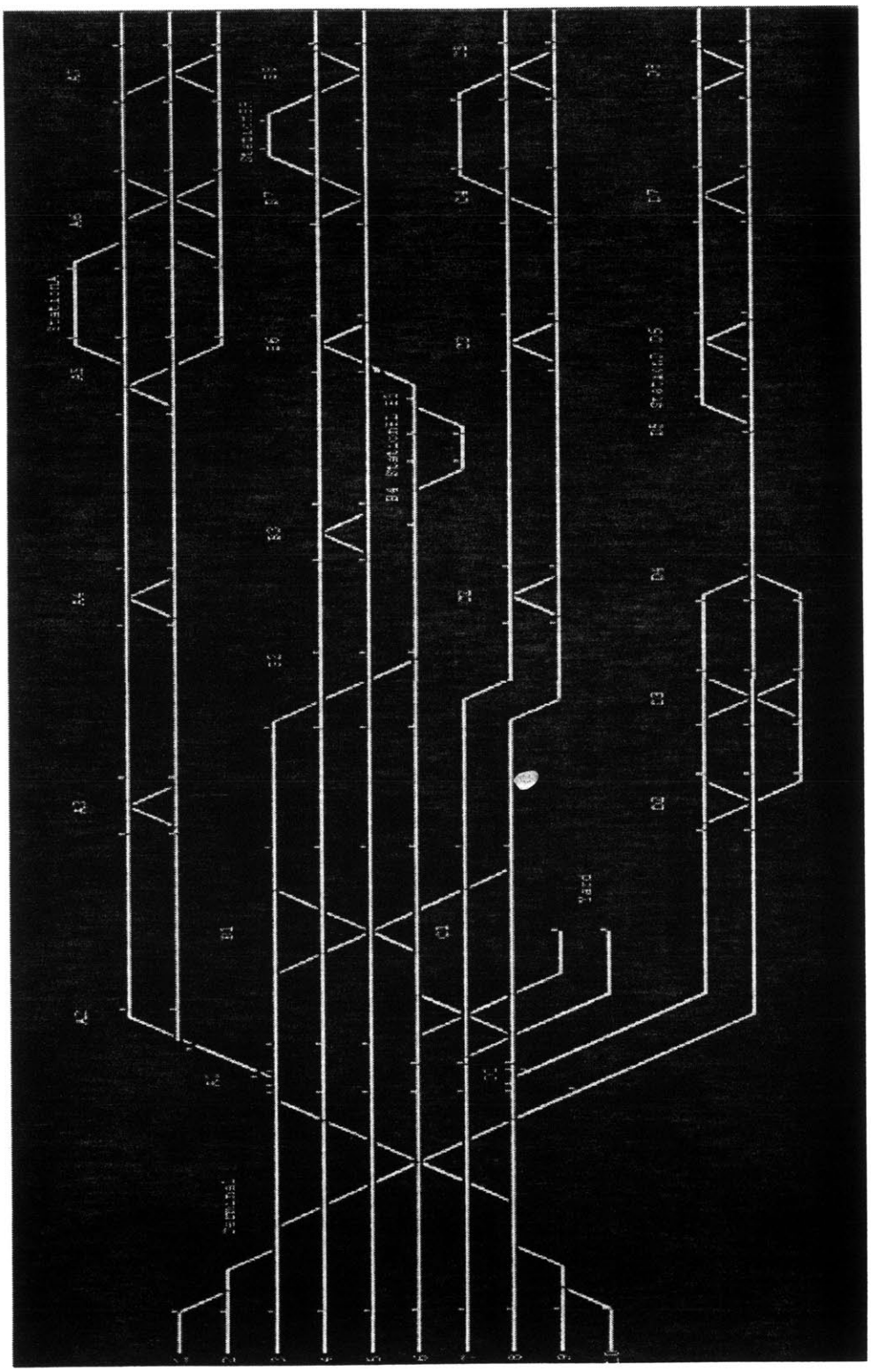
Train Schedule for Scenario 2

Br.	Dir.	Train #	Ptf.	Terminal		Interlocking		Interlocking		Interlocking		Interlocking		Interlocking	
BRANCH A	IN		1	4:31 PM	A1	4:28 PM	A2	4:26 PM	A3	4:19 PM	A4	4:12 PM	A5	4:05 PM	A-t3
			2	5:05 PM	A1	5:02 PM	A2	5:00 PM	A3	4:53 PM	A4	4:46 PM	A5	4:39 PM	A-t3
	OUT		1	5:32 PM	A1	5:29 PM	A2	5:27 PM	A3	5:20 PM	A4	5:13 PM	A5	5:06 PM	A-t3
		191	3	4:00 PM	A1	4:03 PM	A2	4:05 PM	A3	4:12 PM	A4	4:19 PM	A5	4:26 PM	A-t2
		193	2	4:32 PM	A1	4:35 PM	A2	4:37 PM	A3	4:44 PM	A4	4:51 PM	A5	4:58 PM	A-t1
BRANCH B	IN		3	4:19 PM	B1	4:15 PM	B2	4:09 PM	B3	4:04 PM					NS
			4	4:28 PM	B1	4:24 PM	B2	4:18 PM					B4	4:13 PM	BL-t1
	OUT		6	4:44 PM	B1	4:40 PM	B2	4:34 PM	B3	4:29 PM					NS
		291	4	4:01 PM	B1	4:05 PM	B2	4:11 PM	B3	4:16 PM					NS
		293	5	4:23 PM	B1	4:27 PM	B2	4:33 PM					B4	4:38 PM	BL-t2
BRANCH C	IN		8	4:27 PM	C1	4:23 PM	C2	4:19 PM	C3	4:16 PM	C4	4:11 PM	C5	4:07 PM	No
			7	4:37 PM	C1	4:33 PM	C2	4:29 PM	C3	4:26 PM	C4	4:21 PM	C5	4:17 PM	No
	OUT		8	4:58 PM	C1	4:54 PM	C2	4:50 PM	C3	4:47 PM	C4	4:42 PM	C5	4:38 PM	No
		391	8	4:03 PM	C1	4:07 PM	C2	4:11 PM	C3	4:14 PM	C4	4:19 PM	C5	4:23 PM	No
		393	7	4:22 PM	C1	4:26 PM	C2	4:30 PM	C3	4:33 PM	C4	4:38 PM	C5	4:42 PM	No
BRANCH D	IN		9	4:31 PM	D1	4:28 PM	D2	4:21 PM	D3	4:17 PM	D4	4:13 PM	D5	4:06 PM	D-t2
			10	4:54 PM	D1	4:52 PM	D2	4:44 PM	D3	4:40 PM	D4	4:36 PM	D5	4:29 PM	D-t1
	OUT		9	5:20 PM	D1	5:18 PM	D2	5:10 PM	D3	5:06 PM	D4	5:02 PM	D5	4:55 PM	D-t2
		491	8	3:54 PM	D1	3:56 PM	D2	4:04 PM	D3	4:08 PM	D4	4:12 PM	D5	4:19 PM	NS-t1
		493	9	4:20 PM	D1	4:22 PM	D2	4:30 PM	D3	4:34 PM	D4	4:38 PM	D5	4:45 PM	D-t2
Yard	OUT	360EQ	8	ar. 4:40 PM		To Yard									
		120EQ	1	ar. 4:40 PM		To Yard									

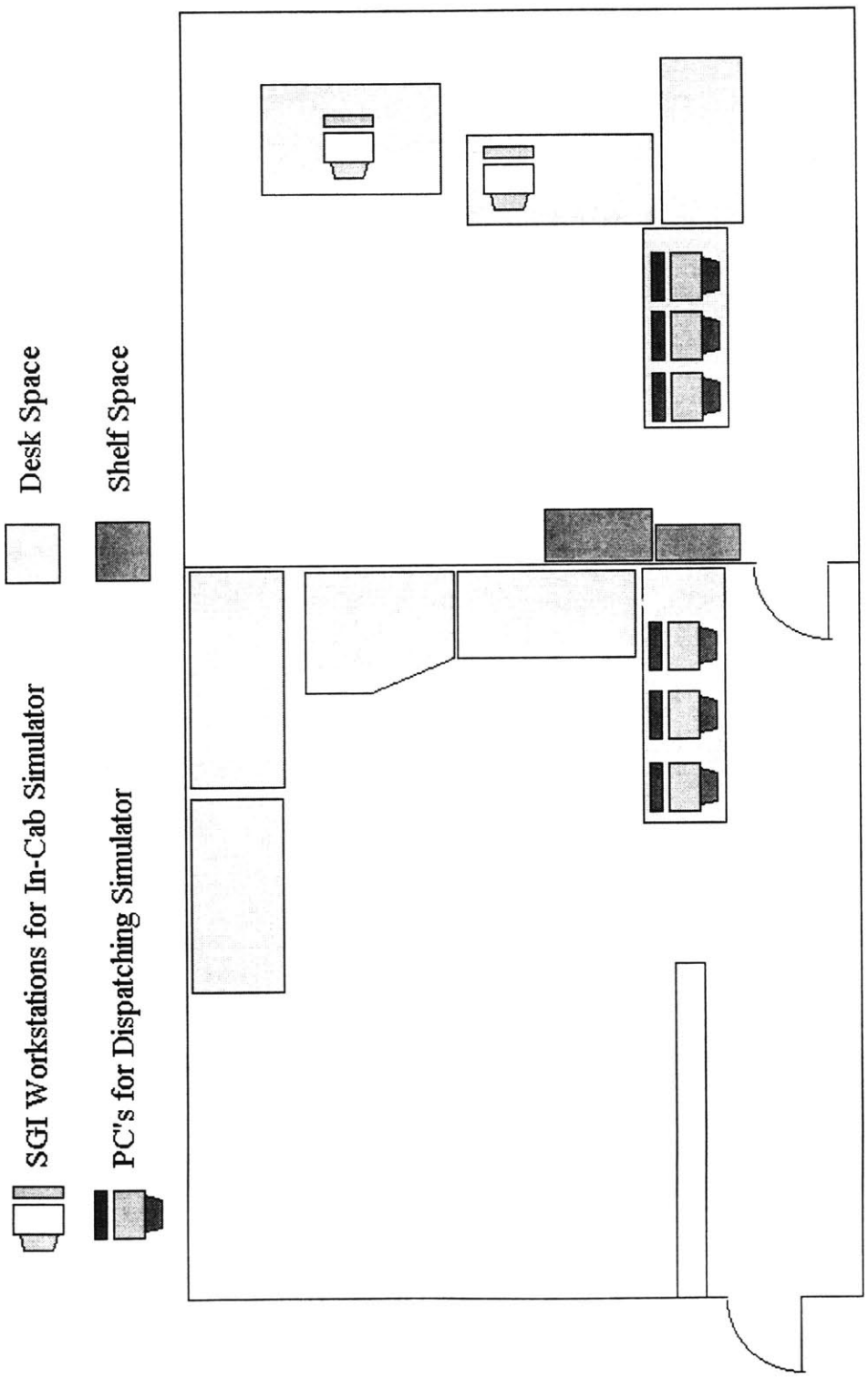
Station		Interlocking		Interlocking		Interlocking		Station		Interlocking	F.D.	Train #	Dir.	Br.
4:04 PM	A6	3:52 PM	A7	3:52 PM							1	120	IN	BRANCH A
4:38 PM	A6	4:31 PM	A7	4:24 PM							2	122		
5:05 PM	A6	4:57 PM	A7	4:50 PM							1	124		
4:35 PM	A6	4:37 PM	A7	4:44 PM									OUT	
5:00 PM	A6	5:02 PM	A7	5:09 PM										
			B6	4:03 PM	B7	3:56 PM	BR-t2	3:54 PM	B8	3:42 PM	3	240	IN	BRANCH B
4:12 PM	B5	4:09 PM	B6	4:08 PM	B7	4:01 PM	BR-t3	3:59 PM	B8	3:45 PM	4	242		
			B6	4:26 PM	B7	4:21 PM	BR-t3	4:19 PM	B8	4:10 PM	5	244		
			B6	4:17 PM	B7	4:24 PM	BR-t2	4:33 PM	B8	4:35 PM			OUT	
4:43 PM	B5	4:44 PM	B6	4:46 PM	B7	4:53 PM	BR-t3	4:56 PM	B8	4:58 PM				
											8	360	IN	BRANCH C
											7	362		
											8	364		
													OUT	
4:05 PM	D6	3:58 PM	D7	3:55 PM	D8	3:52 PM					9	480	IN	BRANCH D
4:28 PM	D6	4:19 PM	D7	4:16 PM	D8	4:13 PM					10	482		
4:54 PM	D6	4:49 PM	D7	4:46 PM	D8	4:43 PM					9	484		
4:20 PM	D6	4:20 PM	D7	4:23 PM	D8	4:26 PM							OUT	
4:50 PM	D6	4:50 PM	D7	4:53 PM	D8	4:56 PM								
										ar. 4:40 PM	Yard	360EQ	OUT	Yard
										ar. 4:40 PM	Yard	120EQ		

Appendix C Territory Map

This map shows the territory in which the simulation was carried out. This is a fictitious track layout, but it was created to appear realistic to a dispatcher. The displays were created by simply taking screen snapshots without any trains.



Appendix D Plan View of Setup



References

[1] Amtrak CETC System Documentation. Amtrak Railroad, internally published.

[2] Santanu Basu. Guide to the System Architecture and Operation of the MIT/Volpe Train Dispatcher Simulator. Project Report, Volpe Transportation Systems Center. Cambridge, MA, 1999

[3] Luc Cassady-Dorion. Industrial Strength Java. New Riders Publishing. New York, NY, 1997

[4] Edward John Lanzilotta. Dynamic Risk Estimation: Development of the Safety State Model and Experimental Application to High-Speed Rail Operation. PhD Thesis. Massachusetts Institute of Technology. Cambridge, MA, 1995

[5] Carl D. Martland. Modeling Railroad Line Performance. Railroad Applications Special Interest Group Newsletter. 1995

[6] Patrick Naughton. Java, The Complete Reference. O'Reily. New York, NY, 1996

[7] Emilie Roth, Nicolas Malsch. Understanding How Train Dispatchers Manage and Control Trains. Volpe Transportation Systems Center Report. 1998

[8] Brotherhood of Locomotive Engineers Web Site (www.ble.org)

[9] Cato Institute Web Site (www.cato.org/pubs/pas/pa012.html)

[10] Federal Rail Administration Web Site (www.fra.dot.gov)

[11] Developer.com Web Site (www.gamelan.com)

[12] JavaWorld Web Site (www.javaworld.com)