A Software Testbed for Advanced Projects in Real-time And Distributed Computing

Richard A. Brown*

Department of Mathematics
St. Olaf College
Northfield, MN 55057
rab@stolaf.edu

1 Introduction

The person in the street takes real-time computing for granted, even if he or she does not know that term. Computers may not be trusted by everybody, but the public nevertheless assumes that they are used to control complex processes such as space shuttle motion and automobile carburetion. In practice, real-world applications are very difficult. Some of the computer science issues that must be faced include "hard" time deadlines, fault-tolerance, formal specification, verification that a system satisfies its specification, and coordination of distributed computing resources [4]. In addition, there are innumerable complications when attempting to model a physical process. We take the latter as a major reason why real-time computing is often omitted from upper-level work in undergraduate computer science programs.

Few undergraduates have either the computational capability or the opportunity to design and implement a significant real-time process-control application of computing. Yet, given the significance of such problems outside of academia and the recent flurry of research activity in the area of real-time computing (see, for example, [1, 2, 3 and 6]), it is increasingly desirable for our students to encounter computing problems involving real-time constraints, distributed computing and related issues, in advanced courses and/or independent projects.

The `Trainset` simulation presents a substantial process-control problem, namely motion of idealized trains along a track, that represents various difficulties in real-time programming while stripping away gratuitous complication. `Trainset` was designed both for student projects in real-time courses and for research in formal methodology. It is the work of the Reliable Real-time research group at Cornell University, directed by Prof. Fred B. Schneider. The software requires only ordinary workstation equipment to use, and provides a C-language library of routines that may be used to interact with the simulated trains. The software provides support for distributed computing and fault-tolerance, and the simulation lends itself to formal specification. These features make it accessible and available for a wide range of advanced student projects in various fields within computer science.

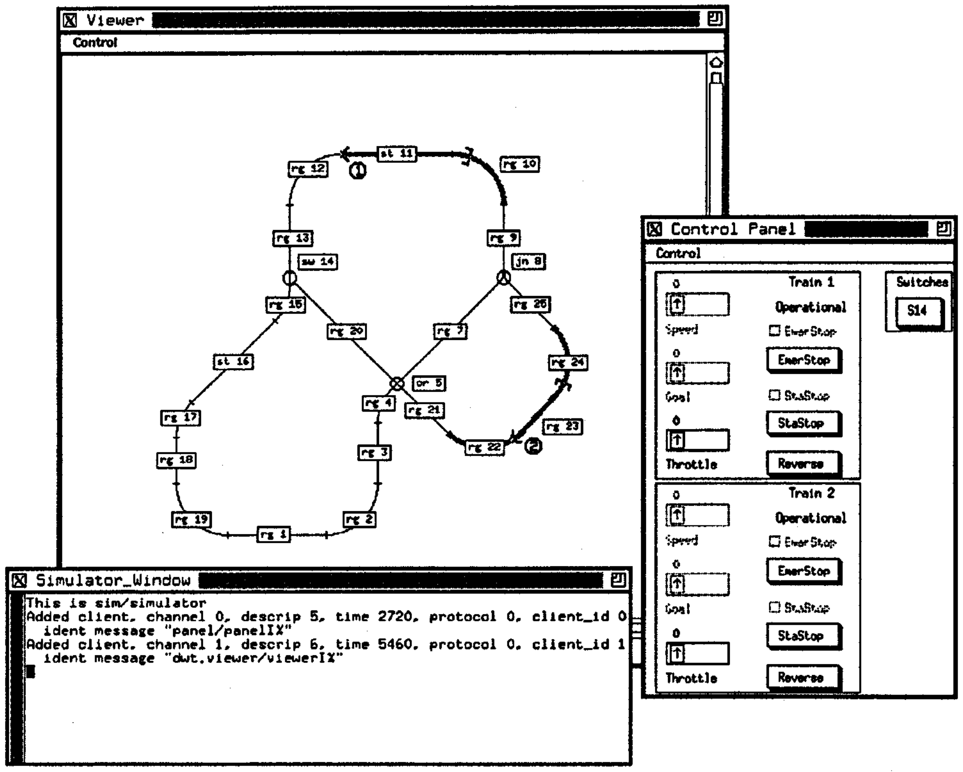
2 The Trainset software

`Trainset` is a real-time simulation of a railroad. The software consists of a simulator, an interactive graphics editor for defining railroad layouts, graphics monitor programs for displaying the state of the railroad and manually controlling it and a launch utility program for coordinated startup of the various programs. The railroad layout editor enables a user to draw track configurations of virtually any desired shape, populated with one or more trains of various lengths. The graphics monitor programs (described below) are easily operated, and can be used alone with a simulation even by inexperienced users to get a sense of the difficulties in controlling a railroad by computer.

The simulator provides a programmer's interface, called the *Automatic Control Interface* (ACI), that consists of a library of routines and data structures for exerting control actions (*actuators*) and gathering data about the current state of a railroad (*sensors*).

`Trainset` has been implemented in C language on Digital Equipment Corp. Unix (Ultrix) workstations using DECwindows/X-11 and either TCP/IP or DECnet.

*This material is based on work supported in part by the Office of Naval Research under contract N00014-91-J-1219 to Cornell University.



```
This is sim/simulator
Added client, channel 0, descrip 5, time 2720, protocol 0, client_id 0
  ident message "panel/panelIX"
Added client, channel 1, descrip 6, time 5460, protocol 0, client_id 1
  ident message "dwt.viewer/viewerIX"
```

Figure 1: Sample railroad layout

```
/*
 * Code segment that initiates acceleration of
 * a train T, allows that acceleration to take
 * effect for 25 seconds, then travels until a
 * block B is reached (or timeout occurs)
 */

printf("beginning acceleration\n");
Accelerate(T, 20.0);
printf("continuing acceleration...\n");
Sleep(25.0);

printf("polling until block %d is reached...\n",
       B);
poll_timeout_flag = 0;
InitTimer(POLLING_TIMEOUT, 0,
          set_poll_timeout_flag);

while ((occ = GetBlockOccupancy(B)) == OC_FREE
       && !poll_timeout_flag)
  Sleep(POLLING_INTERVAL);

if (occ == OC_ERROR) {
  printf("error getting block occupancy!\n");
  exit (1);
}
if (poll_timeout_flag) {
  printf("polling timed out after %f sec\n",
         POLLING_TIMEOUT);
  exit (1);
}
CancelTimer();
/* assertion:  occ == OC_OCCUPIED */

printf("performing emergency stop\n");
EmergencyStop(T);
```

Figure 2: An example using the Automatic Control Interface (ACI).

The ACI library has been ported to Digital's VAXELN real-time kernel. An effort is underway to port the software to other Unix platforms and graphics systems.

When the Trainset software is started up, three windows open on the display, as shown in Figure 1. One of these, the Simulator Window, is a terminal window that displays messages from the simulation. The remaining two windows make up the graphics monitor. The Viewer window shows the current positions of the railroad tracks and trains in a simulation. The Control Panel window has controls and indicators, including pushbuttons and slide bars. It is a graphical interface for manually controlling trains and switch blocks.

The sample railroad layout displayed in the Viewer window of Figure 1 has two trains and 24 *blocks*, or track segments. Each block has maximum and minimum speed limits (not shown). In Figure 1, block cr 5 is a *cross block* that allows intersection of track paths; block jn 8 is a *join block* for merging two paths; and block sw 14 is a *switch block* for interactively selecting between two paths.

The control panel comprises one subwindow for each train and a pushbutton per switch block for toggling between the straight and turned settings. Each train's subwindow includes indicators labelled Speed (current train speed) and Goal (present goal speed desired for that train); a Throttle control for setting a new goal speed; labels showing the name of the train and its state (Operational, Derailed or Collided); pushbuttons for performing an emergency stop, reversing the train's direction or performing *station stop*, i.e., a stop at the next *station block* (such as block st 11); and indicators for the emergency-stop and station-stop features.

A tutorial introduction to the software is provided in the Trainset documentation.

The ACI routines fall into five categories.

- The GetDownload procedure establishes a network connection with a running railroad simulator and receives a report of the state of the railroad being simulated.

- ACI *commands* enable a program to change switch-block positions, accelerate or decelerate a train for a specified time period, set a new goal speed, change the direction of a (stopped) train, perform an emergency stop or enable/disable station stopping.

- ACI *queries* enable a control program to obtain limited discrete state information about a railroad after a download has been received. They can report whether or not a block is occupied, what a switch block's position may be (straight, turned or changing positions), whether a train is operational or not and whether a train is moving or not.

- ACI *voting service* procedures implement a command arbitration facility in the simulator for use in fault-tolerant control programs.

- ACI *utility* procedures provide high-level general purpose timing facilities.

The example control program segment in Figure 2 illustrates the use of the ACI. The GetDownload procedure is assumed to have been issued before this code segment is executed. The functions Accelerate and EmergencyStop are ACI commands; GetBlockOccupancy is an ACI query, and Sleep, InitTimer and CancelTimer are ACI utility procedures.

Note that the Accelerate command in Figure 2 requests that a 20-second acceleration begin. That command does not cause any suspension of the control program. An explicit 25-second Sleep command is issued in order to give the acceleration plan enough time to complete. If a subsequent command affecting acceleration were issued before the 20-second acceleration was complete, it would override that earlier Accelerate command.

As an illustration of real-time computing issues, consider the state of the system at the end of Figure 2. Observe that it is *not* correct to conclude that block B is occupied at that moment, even though the variable occ has the value OC_OCCUPIED. This is because network latency, the method of polling and the execution time of control-program instructions all introduce delays that occur after the variable occ is assigned.

3 Some features of the Trainset railroad simulation

Trainset railroads are idealized versions of real railroads. While Trainset railroads are simpler than their real-life counterparts, the simplifications are ones that do not make it appreciably easier to write programs to control the railroad. For example, in Trainset the acceleration of a train is always one of four constant values ACC, 0, $-ACC$ and $-A_{emer}$ (except during a station stop). All secondary effects such as friction are therefore neglected. However, the simulation incorporates other real-time difficulties such as uncertain knowledge of speed, indirect control of a realistic process and "hard" time deadlines.

For example, after the initial download, no precise knowledge of train speed or position is provided by the simulator to an ACI control program. Information about the current speed and position must be deduced from knowledge of the initial state, of commands that have been issued (such as a command to set a certain goal speed) and of queries that have been made (such as the train motion query, which returns only whether a train is moving or stopped).[1]

Acceleration and deceleration latency illustrates indirect control of a realistic process. For instance, `Trainset` trains do not stop immediately when an emergency-stop command is issued. As in real railroads, one must plan ahead: If insufficient distance is allowed for a stop, then a collision or derailment may result. Also, once a simulated train has derailed or collided, it remains non-functional for the remainder of that simulation. This cost of failure illustrates the "hardness" of time deadlines.

Developing a reliable program that simply moves the trains quickly without violating speed limits on individual blocks is itself a significant problem [5].

`Trainset` has also been used in advanced student projects that do not emphasize real-time computing. One project in computer network programming focused on the simulator's voting mechanism and distributed computing capabilities. It involved replicating an ACI demonstration program provided with the `Trainset` distribution. In addition, several students have "come up to speed" in computing by contributing implementations and ports of the `Trainset` programs to new computing environments and windowing systems.

4 Summary and future work

The `Trainset` software provides a software testbed that is available for advanced programming projects in computer science. `Trainset` presents a simplified yet challenging real-time process-control problem that can be approached as such by users with various levels of sophistication, from the novice "getting a feel" for the difficulties of real-time problems to the researcher investigating formal methods in real-time computing. The software may also be applied in student projects on topics including computer networking and distributed computing, with or without a real-time emphasis.

The software system is currently implemented in C language on DEC equipment. Work in progress includes porting the software to additional computing platforms.

5 Acknowledgements

I am indebted to Fred Schneider, director of the RR project and conceptual architect of the `Trainset` software system, who arranged for financial support that enabled me to visit Cornell, and who has contributed to my professional development in many ways.

Jacob Aizikowitz and Tony Lekas were the primary implementors of early versions of the `Trainset` software.

[1] In contrast to ACI control programs, the graphics monitor programs receive much more complete information about current speed and position.

6 References

1. Cooling, J. E., *Software Design for Real-Time Systems*, Chapman and Hall, 1991.

2. Fuhrt, B., et al, *Real-Time Unix Systems: Design and Application Guide*, Kluwer Academic, 1991.

3. Hooman, J., *Specification and Compositional Verification of Real-Time Systems*, Springer-Verlag, 1991.

4. Krishna, C.M. and Lee, Y.H., "Real-time systems," guest editor's introduction, IEEE Computer, May 1991.

5. Marzullo, K., Schneider, F. and Budhiraja, N., "Derivation of sequential, real-time process-control programs," in *Foundations of Real-Time Computing: Formal Specifications and Methods* (A.M. van Tilborg and G. M. Koob, eds.), Kluwer Academic, 1991, 39-54.

6. van Tilborg, A. M. and Koob, G. M., editors, *Foundations of Real-Time Computing*, Kluwer Academic, 1991 (two volumes).