Development of middleware applicable to various types of system structure in train-traffic-control systems

Y. Kakumoto1, E. Nisijima1, H. Suizu2, & N. Komoda3
1Systems Development Laboratory, Hitachi Ltd., Japan
2Mito Transportation Systems Product Div., Hitachi Ltd., Japan
3Graduate School of Information Science and Technology, Osaka University, Japan

Abstract

COTS (commercial off-the-shelf) products have recently become sufficiently reliable and powerful to be utilized in medium- and smaller-scale train-traffic-control systems. Such control systems have to be developed at lower cost and within shorter periods than in the past. We propose middleware that achieves a standard software architecture applicable to various system structures for train-traffic-control systems. This middleware has a reliable structure and presents operational data for the entire area to application modules in a consistent form, independently of system structure. This standard software will eventually be applicable to various types of system structure. These features can contribute to the realization of more reliable systems and to lower periods and costs of development.

Keywords: train-traffic-control system, middleware, reliability, productivity.

1 Introduction

Train-traffic-control-systems handle route control through operating signals and points for many types of railway systems, including elevated (monorail), underground, and surface railway systems [1]. Whether a centralized or decentralized structure etc. is suitable depends on the scale of the system. In general, train-traffic-control systems have been for the exclusive use of particular railway systems. Although software developed for one system has been applied...
to another in some cases, this has been very difficult. The need for reduced development costs has led to an increase in the use of COTS (commercial off-the-shelf) products such as personal computers and standard embedded controllers, operating systems, and communications links (e.g. Ethernet) in control systems, and medium- and smaller-scale train traffic control systems need to be developed at lower costs and within shorter periods than in the past. One aspect of achieving this is to realize software that is consistently applicable to various system structures.

The highest consideration in designing a train system is reliability, including non-stop operation. Train-traffic-control systems are generally implemented as real-time systems, in which program modules communicate with each other through data in global memory area and mutual exclusion is applied during access to this data. Since this sometimes leads to deadlocks, however, a structure that avoids the need for mutual exclusion is preferable in terms of reliability. The second most important requirement is for a consistent operational data management process that is independent of the system structure. The number and placement of machines differs with the structure. The autonomous decentralized system with multicast communications [2] and distributed object technology [3] are well known as techniques that provide positional transparency. However, these techniques only conceal the physical structure. To realize the consistent operational data management, we also need to remove the need for application programs to be aware of the logical system structure.

We propose middleware that achieves the kind of standard software architecture that is necessary for application to various system structures. The middleware itself has a reliable architecture, in which a single program provides both a received-data buffering function and a data hand-over function. It also handles low-level editing of the data (i.e., consolidation and consistency-checking of data) and presents the data to application program modules as overall control area information. This relieves the application modules of the need to be aware of the overall system structure, so each only has to monitor or control its own specific area. These features can contribute the realization of reliable systems and reduced costs and periods of development.

2 Train-traffic-control systems

2.1 Basic system structures

A train traffic control systems tracks trains by receiving state information from equipment such as blocks, signals, point switches, etc., and controls the equipment to safely select the correct train routes. It also controls the announcement of arrival and departure times through displays and other means. Figure 1 shows various types of system structure for developed train-traffic-control systems. System components of the central subsystem include an operational management machine, a diagram server, monitoring machines and a projector. The management machine manages diagrams of the day’s operations and checks command information from the monitoring machines. The server
holds diagrams of future train-traffic patterns and record information on the current system state. Station subsystems include interlocks and guidance-information devices. Route-control machines, that is, the systems within the dotted rectangles, output route-control and guidance data to these devices. The route-control machines also handle the train-tracking and route-control functions. The network within each of the subsystems is Ethernet-based. The train-line network is a fibre-based or other type of wide-area network (WAN). The management and route-control machines are dual systems, i.e. two machines provide redundancy and are alternately in service. The server and monitoring machines are personal computers. The scale of the train system and operational conditions determine whether the system structure is centralized or decentralized. In a centralized system structure, there is only one route-control machine, which is situated in the central subsystem. This single route-control machine controls the whole line. A system structure in which each station subsystem has a route-control machine is referred to as “station-decentralized”. In this case, each route-control machine controls its own subsystems. A system structure in which some of the route-control machines are in the central subsystem is referred to as “center-decentralized”. The rough criterion for system structure is that centralization is applicable to medium- and smaller-scale systems while decentralization is applicable to large-scale systems.

Figure 1: Various system structures for train traffic control systems.

2.2 Basic functions

Figure 2(a) shows the basic function of train-traffic control. The route-control machine receives the equipment data such as the status information of signals, points and blocks from the interlocking device and tracks trains on the basis of
the latest equipment data. It then selects train routes and maintains train routes by sending control data to the interlocking device and guidance data to the guidance device. Access to operational data such as the equipment data and tracking data requires mutual exclusion if the access produces a change to the data, because the receiving and control functions are executed independently.

Functions of the modules in the route-control machine differ greatly according to whether the structure is centralized or decentralized. The route-control machines of a decentralized structure have to cooperate with each other. As is shown in Figure 2(b), the transfer of train-tracking data among the route-control machines is such that each gets the information on trains entering into its own control area. Cooperative data-exchange functions are included in application modules such as those for equipment-data reception and train tracking. A centralized system does not require such cooperation, because a single route-control machine controls the entire operational area. Functional differences of system structure also exist in the central subsystem machines. The management and monitoring machines always refer to the operational data on all equipment and trains. In the decentralized case, the machines have to consolidate operational data form the distributed route-control machines and generate operational data for the entire line. In the centralized case, data consolidation is not necessary because the machines only receive data from a single route-control machine.

![Figure 2: Function of software in the route-control machine.](image)

### 3 Needs

#### 3.1 Flexibility

As was mentioned above, the individual machines of a decentralized structure not only get operational data from the route-control machines but also have to
consolidate the data and check it for consistency. Some application modules of the conventional decentralized system include such editing functions. However, these functions are unnecessary for a centralized system. To apply the same application modules to various types of system structure, we have to separate the editing functions necessary for a decentralized structure from the application modules. The operational data must also be presented to the application modules in a consistent format, i.e. operational data for the overall train line must be such that each application module simply has to monitor or control its own specified control area while not having to be aware of the overall system structure. By developing such functions as middleware, we achieve a software architecture that is applicable to the various possible system structures.

3.2 Reliability

High quality in terms of reliability, such as a capability of non-stop operation, is essential for train systems. This requirement, of course, still applies when COTS products are used. The proposed middleware should thus support functions for reliability. In real-time systems, including train traffic-control systems, certain task modules are executed in response to random-time triggers. These task modules usually apply mutual exclusion to data that might be accessed by other modules [4][5]. Mutual exclusion introduces the possibility of deadlock [6]. Thus, in terms of system reliability, it is desirable that the software structure eliminates the need for mutual exclusion.

4 Proposed middleware

4.1 Approach

Figure 3 shows the total software architecture including the proposed middleware. The middleware modules handle data reception, data collection, data editing, and the output of updated data. The main modules are those for data collection and editing. The data-collection module achieves reliability for this software architecture. The data-editing module provides an interface with operational data that is consistent and independent of system structure. These essential functions are described under (1) and (2) below.

(1) How the data-collection function eliminates mutual exclusion
The data-reception modules are event triggered while the main functional modules are cyclically triggered in the conventional system, so the timing of execution is different. Mutual exclusion is therefore necessary when these modules concurrently access the same operational data. In the proposed architecture, the data flows for the data-reception (event-triggered) and data-copying functions (cyclically triggered) are integrated in a single task module, the data-collection module. Furthermore, the data-copying function triggers sequential execution of application modules such as those for train tracking, route setting, etc. Namely, after execution of the data-collection module (data copying) (①), execution of the data-editing modules starts (②).
data, the data-editing module presents the data to the required application modules, which it then starts up (③). When the data-editing module is again started by an application module (④), it notices the completion of application execution and then starts the updated-data-output module (⑤). This module then sends the data as updated by the application modules to other machines. Mutual exclusion is not necessary because these modules are not simultaneously executed.

(2) A consistent data interface for independence of system structure
The data-editing module has consolidation and consistency-checking functions for the train-tracking data presented by the data-collection module. Namely, this module has the function of always presenting the equipment data and tracking data in a consistent form, i.e., as the overall train-line operational data, to the application modules. The editing module thus provides a consistent interface and relieves individual application modules of the need to explicitly gather information from the overall system structure; each module only has to monitor or control its own specified area of the line.

### Figure 3: The proposed software architecture.

#### 4.2 Data-collection module
The data-collection module has two functions: buffering of the data in event-triggered execution and presentation of the received data to the data-editing module in cyclically triggered execution. This task module’s special feature is the integration of both functions.
(1) Buffering of received data
Certain data-reception module is executed in accordance with the kind of the received data. Each reception module immediately starts up the data-collection module on receiving data. When the data-collection module is executed by any of the data-reception modules, it buffers the received data in a specified buffer area. The buffer areas for train-tracking and equipment data are overwrite buffers in which assignment of data blocks is per machine unit. The received data is written over the existing data in the corresponding data block. The command data is registered in a FIFO (first-in first-out) buffer because processing must be in the order of reception.

(2) Data hand-over function
To hand over buffered data to the data-editing module, the data-copy function is executed when the time-management module cyclically starts up the data-collection modules. This function copies or shifts the buffered data into the reference table area and starts the data-editing module. The data structure in the reference table area is the same as in the buffering area.

The above two functions of this single task module (i.e. the data-collection module) are not executed at the same time, so that the relevant buffer and reference table are accessible without mutual exclusion. This achieves a reliable software structure.

4.3 Data editing module
In the case of a decentralized structure, the data-editing module consolidates the equipment and tracking data, checks it for consistency, and then presents it to application modules in the form of overall operational area information. Editing of the equipment and tracking data is described below.

(1) Equipment-data editing
The data-editing module basically merges the local-area data from the route-control machines into over train-line-equipment data and renews areas of this data as required.

(2) Train-tracking-data editing
An example of consolidation and consistency checking of tracking data in a decentralized structure is described. Different control machines are responsible for the left and right sides as shown in Figure 2 (b) in a decentralized structure. Now we suppose that a train is now on the boundary between the two control areas in a decentralized structure. We may have a situation where both machines are sending tracking data that indicates different positions for this train. In this case, the operational safety logic gives precedence to the tracking data form the machine on the side towards which the train is travelling [7]. This function is referred to as the most important form of consistency checking. The data-editing module presents train-tracking data consolidated from the train line as a whole to the application modules.
The above two functions are programmed into the data-editing module so that each application module only has to monitor or control its own specified area and is not explicitly concerned with operational data for the train line as a whole. In the case of a centralized structure, the editing module does not necessarily consolidate the operational data because a single route-control machine controls the whole line.

5 Reference interface for application modules

Figure 4 shows how control applications refer to operational data on the train line as a whole in the route-control machine. In the case of the decentralized structure, the control application extracts tracking data on trains in its local control area from the overall data and overwrites data thus extracted within a private area. Equipment data is handled in the same way. The control application tracks trains and renews train positions on the basis of equipment data thus renewed. It then sets train routes and outputs the corresponding route-control data. The proposed middleware consolidates the overall operational data by merging the locally renewed tracking data and the data form other control machines into new overall operational data, and then presents this to the control application. In the case of a centralized structure, the private data area and overall control information area contain identical data, because a centralized route-control machine controls the entire train line. However, the data-reference interface of the control application is the same regardless of the system structure. Namely, the control application extracts the necessary data from the overall control-information area and overwrites data within a private data area. This makes standard application programs applicable to different system structures.

Figure 4: Data links between application and middleware modules of the route-control machine.
6 Evaluation of the proposed middleware

We applied our new approach to a train-traffic-control system and verified the order of execution of program tasks and the operational performance. The system has 10 stations. The route-control and management machines are embedded systems with 90-MHz COTS microprocessors and COTS real-time operating system is installed. Figure 5 shows the transitions in the execution of program tasks, and was obtained by analysing the actual task-execution log. The left side of the figure depicts transitions in cyclically triggered execution. Firstly, the time-management task acts as the trigger event. The data-collection, data-editing and application 1, 2, 3 tasks are then executed in sequence. While the data-editing task is in progress, it is pre-empted by another task. This means that the context of the CPU has switched to a higher-priority task, in this case, the mutual monitoring task for dual system. On completion of the mutual monitoring task, the context of the CPU returns to the data-editing task. On completion of the control-output task, the data-editing task is again started up so that it can be informed of the completion of application-task execution. After this has been done, the updated-data-output task commences. In this traffic-control system, the route-control machine receives equipment data from the interlocking device once per second. The right side of Figure 5 shows task transitions in the reception of equipment data. On receiving data, the equipment-data-reception task initiates the data-collection task. The other type of event-driven task sequence is in response to the reception of command data from management machines. The task transitions in this case are the same as in the reception of equipment data. This verification of program-task transitions through analysis of the task-execution log demonstrates that the proposed architecture operates as designed. Furthermore, we found that the total execution time per cycle in cyclically triggered execution is only about 100 milliseconds. The system thus presents no problems in terms of performance on our target platform.

Figure 5: Time schedule of task modules.
7 Conclusion

We proposed middleware as standard software applicable to train-traffic-control systems in various structures. The proposed middleware ensures that consistent overall train-line operational data is always presented to the application modules, regardless of system structure. Each application module thus only needs to monitor or control its own specified area, that is, it is relieved of the need to explicitly take any action in terms of the overall system structure. This makes standard software consisting of the proposed middleware and application modules applicable to various system structures. The middleware leads to a 50% reduction in test periods and a 15% reduction in software-development periods. The middleware also has a deadlock-free structure for reliability. When implemented in the actual train-traffic-control system, the software ran efficiently and as designed. We thus verified the utility of the proposed architecture. Realizing a hierarchical structure of this type would generally require extra resources or program steps (e.g. decentralized type route-control machines would require control-data areas other than those for their own control data). Nevertheless, progress in hardware performance leads us to conclude that the merits in terms of software productivity, quality and reliability are greater than the demerit in terms of resources. With further progress in development environments and hardware performance, we can expect the middleware approach to spread to many industrial fields, and not just railway systems.

References