ADVANCED SOFTWARE DESIGN AND STANDARDS FOR TRAFFIC SIGNAL CONTROL

By Darcy Bullock,1 Associate Member, ASCE, and Chris Hendrickson,2 Member, ASCE

(Reviewed by the Urban Transportation Division)

ABSTRACT: Improved traffic management and control systems are widely reported to be cost-effective investments. Simply retiming signals can provide significant benefits by reducing vehicle stops, travel times, and fuel consumption. The installation of advanced traffic management systems (ATMS) can provide even greater savings. However, many hardware and software obstacles have impeded the actual implementation of advanced traffic management systems. The general hardware and software limitations of current traffic signal control technology are reviewed. The impact of these deficiencies is discussed in the context of three example applications. Based on this discussion, the paper identifies several computing issues that should be addressed in order to reduce the effort involved with integrating existing traffic control devices. Adoption of standard industrial control computing platforms and development of new communication and software engineering models are recommended.

INTRODUCTION

Introducing advanced software environments and standards for traffic signal control may have several advantages. First, configuration of a system controller for a particular intersection could be simplified, thereby reducing time and programming expertise required. Second, communication options may be increased and made more flexible, thereby aiding corridor-level control technologies. Third, traffic signal control might take advantage of the hardware production economies and experience in the industrial market, thereby reducing the cost of controllers. Finally, flexible software platforms would allow “standard” devices to be used in new applications, such as weigh-in-motion systems, reversible lane installations, and intelligent vehicle-highway systems.

In the following sections, we review the limitations of existing traffic control software environments and present some selected applications requiring more sophisticated controllers. Requirements for a new generation of control systems are proposed, and these requirements are compared with the current practices for real-time control. A final section suggests an evolutionary process leading to improved traffic control systems.

PROBLEMS WITH CURRENT TRAFFIC SIGNAL CONTROL
SOFTWARE TECHNOLOGY

The current technology for traffic signal control and configuration is based on standards and protocols that are more than a decade old (Chase and Hensen 1989). This technology has a number of associated problems:
• Existing standards are microscopic in nature. They detail electrical interfaces, standard signaling practices, physical dimensions, and rigid, inflexible microprocessor platforms.
• The interfaces to controllers are overly complex and difficult to manipulate. This problem has been recognized and many companies provide microcomputer-based software for configuring controllers. This software allows better editing facilities, but preserves the underlying complex interface. Little has been done to improve the transition from conceptual design to actual implementation.
• The existing microcomputers used for traffic control devices are not powerful enough to accommodate the increased memory and processing demands required for more sophisticated control.
• Finally, budget constraints and low-bid selection often dominate decisions when evaluating any signalization project. To stage improvements to signalization systems, the hardware must not only be modular, but it must support common software and communication protocols.

The traffic control community is not isolated in its quest for modern, easy-to-use systems. Many of the same issues facing traffic engineers currently face engineers in manufacturing and industrial control. Organizations involved in these areas have sought to address system integration issues by adopting standards. In developing such standards, it is always difficult to balance vague standards (such as specifying a communication port) and specific standards (such as specifying an RS-422 serial port supporting a specific protocol). If standards are precisely detailed but leave room for flexible implementations, they benefit the user community by ensuring compatibility between vendors and promoting competitive bidding. However, if they are loosely detailed and do not provide an adequate foundation for applications without additional proprietary features, they leave room for vendors to craft custom features that are not uniform from vendor to vendor. These issues can be illustrated by examining two of the most common standards in traffic control: the National Electrical Manufacturers Association (NEMA) and the California Department of Transportation (Caltrans) type-170 standard.

The NEMA standard specifies details for the mechanical and electrical nature of all the connectors on a controller unit. (In this comparison we will only consider the 1988 NEMA standard, designated TS1. The proposed NEMA TS2 standard addresses some deficiencies in the TS1 standard, but as of this writing it has not been officially released or adopted.) In concept, the modular connectors attached to one controller conforming to the NEMA standard can be disconnected and another vendor's controller (also conforming to the NEMA standard) can be reconnected. This is true if the controller is operating at an isolated intersection and the software configuration is disregarded. However, the NEMA standard is electrical in nature and does not specify how controllers are configured. The problems inherent in the existing NEMA standard are widely recognized. In particular, the NEMA standard has not adequately addressed system coordination, time base control, preemption, uniform code flash, communications, or diagnostics ("Migration" 1990). Consequently, each manufacturer has implemented custom features in a unique way on its controller. These proprietary features counteract the spirit of interchangeability of the NEMA TS1 standard, and as a result, technicians installing a new unit must understand new configuration software. Further, connecting a standard NEMA controller
to detectors, conflict monitors, and other controllers requires the use of one conductor for each signal. If all the connections were used, the unit would have 322 conductors leading into the controller ("Migration" 1990). This type of networking is not cost-effective in view of modern communication technologies, and it provides a multitude of potential failure points. This communication problem has been recognized by various vendors and resulted in additional proprietary communication and printer ports.

The Caltrans 170 controller specification addresses standardization from a portable software perspective by precisely detailing a 6800-based microcomputer. These Caltrans 170 controllers are general-purpose computers with diverse applications ranging from pump control to networked traffic control systems. In this architecture, a general-purpose, rack-mountable computer is fitted with traffic control firmware. In concept, this standard is very effective at providing a software migration path for controllers because of the flexibility afforded by a general microcomputer. However, software is rigidly tied to the existing hardware architecture because no standard operating system or kernel is defined (Chase and Hensen 1989). The reality of the type-170 standard is that the application software produced for Caltrans 170 controllers must be crafted using low-level code and cross compilers. Much of the application software developed for the Caltrans 170 controller is capable but difficult to configure. Minor modifications to accommodate nuances of unique intersections often must be contracted with software shops or simply ignored. This often results in subpar performance because minor changes can be too difficult or expensive to implement and maintain.

Reviewing the NEMA TS1 standard and the Caltrans 170 standard demonstrates the need for uniform electrical interfaces and flexible software platforms. However, neither standard provides an adequate framework for more sophisticated hardware platforms or software engineering methods. Development of software standards for configuring controllers, network installation, and system integration, should be examined in order to reduce software maintenance costs. In fact, Chase and Hensen (1989) noted, "Software used in today’s control system architectures often plays a more crucial role than does the hardware."

**EXAMPLE APPLICATION PROBLEMS**

Basic traffic control functions are adequately performed by the existing controllers, but when sophisticated control is needed these standard systems are not adequate. To demonstrate the need for flexible software environments, this section reviews three case studies.

A common approach used for implementing intersection control with NEMA devices is to specify parameters defining a sequence of phases. Accompanying menus are used to define corresponding cycle length, split, and offset values. A myriad of menus can be used to activate features ranging from dual ring operation to periodically reporting controller status over telephone lines. This menu-based configuration tends to be very structured, hindering the flexibility of traffic engineers if the control software has not been designed to deal with a particular situation. For example, Marshal and Berg (1990), in a recent study evaluating railroad preemption capabilities of NEMA traffic signal controllers, noted the following:

- Features present on the controller will dictate what control strategies are possible.
• Implementation of some features does not allow enough flexibility to create safe preemption designs.
• Flexibility is the key to preemption hardware because each installation will have its own requirements.

Their findings appear to be a direct result of the rigid software implementation architecture selected by the manufacturers. Given the frequency of railroad preemption, it is disappointing that safe preemption cannot be achieved under all conditions.

In addition to simply activating or deactivating a particular function from a menu, it is often desirable to customize a controller's operation beyond what the manufacturer had envisioned. For example, a recent field test of the optimized policies for adaptive control (OPAC) had to rely on a personal computer (PC) instrumented with a counter cards and parallel I/O. The PC was used for acquiring detector data and switching selections. The PC was interfaced to a NEMA controller using force-off and hold logic. This installation required development of custom assembly language interfaces and a Fortran-based implementation on the PC ("Evaluation" 1989). Aggregate field tests results showed a 9% decrease in delay when the OPAC strategy was used instead of actuated control. Despite this successful demonstration, the study reported it was infeasible to implement the OPAC strategy on existing controllers due to slow speed and lack of memory. Consequently, the authors of "Evaluation" (1989) recommended the development of a new controller capable of running such software.

A recent Caltrans report evaluated future platforms for developing "sophisticated" traffic control systems (Quinlan 1989). The report cited more than a dozen applications requiring advanced controllers and described several existing, custom traffic systems ranging from proprietary Pascal/MS-Dos–based weigh-in-motion (WIM) systems to a reversible lane control systems running under OSENGINE. Many problems were encountered due to the custom nature of these systems. For example, the German-built WIM systems were out of service for as long as six months, and the developer of the reversible lane system declared bankruptcy before completion. The lack of traffic control hardware standardization among various projects has made it expensive for Caltrans to take over operation and maintain these proprietary systems. Quinlan concluded it was important to develop an advanced controller based on industrial standards to facilitate implementation of future project requiring sophisticated control and communications.

This group of example applications is extremely diverse, ranging from relatively minor adjustments of traffic controllers to complex distributed reversible lane control systems. One problem common to all these examples is a standard mechanism for adapting or creating software. The train preemption evaluation presents a very clear example on why a flexible software configuration environment is needed. The OPAC example illustrates the deficiency of both a hardware and software environment for developing maintainable systems. Finally, the Caltrans report recommends more advanced hardware, but stops short of making similar software recommendations (Quinlan 1989). This leaves the problem of integrating diverse traffic control and monitoring systems for effective traffic management. Furthermore, future intelligent vehicle-highway systems will demand a homogeneous highly capable software environment to ensure maintainability.
REQUIREMENTS FOR NEW SOFTWARE ENGINEERING TECHNOLOGY

Simply inventing a new generation of controllers, with the latest, fastest processors, huge quantities of memory, and fast networking, will not guarantee major traffic control improvements. To ensure effective traffic control, systems must be easy to assemble, configure, and, most importantly, maintain. The significance of maintaining the software operating traffic systems is illustrated by the substantial benefits associated with retiming signals (Kessmann et al. 1985). Current notions on what constitutes maintainable hardware are very well established. However, attempting to define maintainable software is much more difficult and falls under the broad classification of software engineering.

It is useful to begin by defining what is meant by software engineering. One useful definition is: "The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines." (Pressman 1987; Naur and Randall 1969). This definition has particular relevance to the traffic control community because it explicitly addresses financial, reliability, and application issues. This perspective considers the entire life cycle of software systems: idea, specification, coding, and maintenance. By viewing the task of configuring traffic control devices in this manner, the economics associated with maintenance are clearly identified. Application of software engineering concepts provide a structured framework for development, coding, and documentation. This avoids the problems associated with ad hoc development and defines metrics for evaluating the software before it is developed. Of course, imposition of a structured development does not eliminate the need for creativity in the overall software concept.

Since traffic control systems have evolved from electromechanical systems into microprocessor systems, it seems logical that the same engineering concepts applied to traffic control systems in the past can now be applied to new software-based traffic control systems. However, hardware and software engineering concepts are fundamentally different, which often leads to confusion on what sound engineering principles should be applied. For example, hardware systems are typically constructed by assembling standard hardware modules. In contrast, software systems are seldom constructed by assembling software modules (Pressman 1987).

Another important difference between hardware and software engineering is the fundamental difference in failure curves. For example, hardware devices typically have a failure curve similar to that shown in Fig. 1. This U-shaped curve typically has a relatively high initial failure rate. Following this burn-in period, the failure rate is very low until life expectancy is reached. As the life expectancy of the device is approached, the failure rate of the device begins to increase, indicating the device is wearing out. Because software has no moving parts that burn out or wear out, the failure rate for software is often idealized as a curve asymptotically approaching zero (Fig. 2). However, actual software systems undergo a great deal of modification and change. This leads to a typical failure curve similar to the scalloped curve shown in Fig. 2. The idealized failure curve is never realized because changes tend to decrease the overall integrity of the system as time goes on.

Because of the significant failure rate associated with software systems, the maintenance of software systems often consumes between 50% and 70% of a software organization's budget (Pressman 1987). With typical maintenance costs of this magnitude, it appears that the development of a struc-
tured software engineering methodology could return significant benefits by simply reducing maintenance costs.

Lawson (1990), discussing the software development for several real-time systems, noted, "A sound problem-relevant philosophy is the key to achieving successful implementation of complex computer based systems. Software engineering methods and tools will naturally flow from this foundation." This leads us to present several issues that should be addressed by the implementation philosophy.

- Traffic controllers need to be configured by traffic engineers and a standard representation should be developed for configuring them. Currently, manufacturers' control strategies, symbols, and terms differ from those provided in ITE recommended practices (Marshal and Berg 1990). Integration and standardization of this representation with stochastic simulation and design programs will provide better tools for effective strategy design and timing.
- The configuration architecture needs to be open and extensible, with provisions for incorporating interfaces to special-purpose systems used...
in advanced traffic management systems and advanced traffic information systems. For example, changeable message signs, driver information systems, vision systems, sonar systems, ramp metering, and incident detection should be accommodated by a new architecture.

- Existing electromechanical and solid-state controllers cannot be discarded when new controllers with modern technology are introduced. Rather, older existing control zones must be capable of interfacing with new control zones. Proprietary circuits and custom microprocessor systems used to bridge incompatible equipment should be replaced by highly configurable, distributed controllers conforming to standard software metrics.

- Advances in communication technologies have made networks ranging from twisted-pair cabling to fiber-optic strands superior alternatives to the electrical interconnection mechanism defined in the NEMA TS1 standard. Obviously, the varying costs, ranges, capabilities, and capacities of the various network alternatives cause different media to be selected for different applications. Software realizing the benefits of networking must also be abstract enough to migrate to more sophisticated media as needs change.

- System costs must remain competitive with existing units. Advances in hardware manufacturing have caused software costs (development, maintenance, enhancements) to account for a significant portion of a systems costs. Hence, systems will have to provide clear engineering benefits (reduced development, lower maintenance, flexible extensible architecture) to warrant the additional cost of hardware and software.

- The traffic control community needs to look toward a more sophisticated programming environment tailored to the specific problems facing traffic engineering. One approach used successfully by process control software vendors is a function block approach (Elwart and Martin 1990). Control algorithms could be configured by connecting various function blocks using icons in a CAD-type configuration program on a personal computer. Once a control strategy has been completed, it could be simulated and validated on the personal computer. The blocks (represented as data structures) defining a strategy can then be downloaded to small, inexpensive local controllers over a communication link. This approach holds considerable promise for the traffic control community, provided an appropriate block language and representation can be developed.

Recognizing the importance of software engineering in traffic control systems, it is important to develop quality software engineering techniques that reduce costs associated with steep learning curves and minimize software maintenance costs. No longer is it acceptable to purchase hardware and then try to design software to meet a region’s needs. Rather, the software must first be carefully selected and engineered to provide the maximum flexibility with the least amount of maintenance. This criterion can be interpreted very widely (from assembly language coding, to merely entering cycle, split and offset). Most likely, regional and local traffic control needs are best served by a compromise between these two flexibility extremes.
Previous sections outlined problems with current traffic controllers, cited requirements for new control system software, and reviewed applicable technology and trends in industrial control. Keeping in mind the existing huge investment in traffic control systems and the fact that most of the systems do work, an evolutionary process toward new software environments and standards might be followed. However, an evolutionary process does not preclude introducing new controllers, new programming methods, or new networking concepts, as long as adequate provisions are made for interfacing with existing equipment. Following these guidelines, an outline detailing the desired software and complementary hardware is described. The most important point of this system outline is that the software and hardware technologies described already exist and have been implemented in several other domains. The points that follow represent some required steps to take advantage of these technologies.

1. Intuitive representation: The single most important advance needed in the area of software engineering for configuring traffic control devices is an intuitive representation. Such a representation would allow the engineer to design control algorithms using graphical icons and data exchange with simple graphical connections. This function block, or object-oriented approach, was pioneered by the Foxboro company in the early 1970s and has evolved to a point where there is almost a one-to-one correspondence between process and instrumentation diagrams (P&IDs) and the graphical implementation of the control system (El-wart and Martin 1990). The engineering tasks for porting this type of system to traffic engineering require the definition of suitable functions blocks, in combination with updating the ITE recommended practices.

2. Integrating simulation and control: An approach often used by large system integrators in other industries involves staging and simulating an entire control system. The technology currently exists to construct entire computer simulations with displays indicating microscopic intersection performance and macroscopic network performance. However, these simulations are often done using traffic control algorithms that differ from those implemented. If geometry and network topology are included in a standard representation, then stochastic modeling and simulation programs can be used directly with the defined control system to evaluate performance and potential impact of proposed improvements. Conceptual models could be constructed by the traffic engineer and adjusted to improve system performance. Confidence would be gained in the performance of the control system, and economic viability could be justified before the system is purchased and installed. Considering the conservative posture of most public works departments, this capability would be more important as systems become more complex and intimidating.

3. High-level networking: Varying needs and financial constraints dictate that networks of varying complexity be supported. Consequently, the traffic control community needs to develop an abstract network representation for defining communication channels that is media independent and based upon rigorous communication standards. Industrial networking and protocols may provide an effective standard or starting point.

4. Integration with NEMA standards: The NEMA TS1 standard has been widely criticized for not addressing critical issues, but the contributions it has made by standardizing electrical connections should be recognized. The pro-
posed NEMA TS2 standard also make significant contributions by recognizing the need for high-speed communication ports, reducing the number of hard connections, and recognizing the need for better user interfaces. Because the NEMA standard is predominantly an electrical standard and does not rigorously define the communication protocol to be used by the communication ports or the software interfaces for configuring the units, it must be supplemented with software and networking standards.

5. Integration with industrial hardware: The past decade has seen tremendous growth in the power, quantity, and quality of industrial computation. Industrial platforms are becoming more powerful and highly configurable. Because industrial systems employ much, if not all, of the technology required for traffic control, it seems likely that traffic control manufacturers may benefit by entering into an original equipment manufacturer (OEM) arrangement with controller companies in order to develop a more powerful and flexible traffic control device. The VME bus system, suggested by Caltrans, appears to be a likely candidate at the current time (Quinlan 1989). This approach can provide significant benefits for traffic control. First, industrial computers are produced in much larger volumes and marketed in a very competitive environment. This permits the traffic control community to benefit from the latest hardware technologies on stable platforms without the full burden of development costs. Second, if the low-level networking and distributed I/O were built upon commercial systems, unusual control situations could be handled with industrial computers and more easily integrated into the system.

APPENDIX. REFERENCES


