Roadway Traffic Control Software

Darcy Bullock and Chris Hendrickson

Abstract—Roadway traffic control is an important practical example of real time control applications, but effective roadway control is hampered by a variety of organizational, financial and technical considerations. One major hurdle is the current reliance on outmoded field hardware and software. A systematic approach to traffic engineering software development could provide significant advantages with regard to software capability, flexibility and maintenance. Improved traffic controllers will likely be essential for many of the proposed intelligent vehicle highway systems (IVHS) applications. This paper describes the roadway traffic control problem generally and introduces a computable language that can be used for constructing real time traffic control software. This computable language is designed to be configured by a graphical user interface that does not require extensive software engineering training to use, yet provides much more flexibility and capability than possible by simply changing program parameters. The model is based upon the function block metaphor commonly used for constructing robust and efficient real time industrial control systems. The software model has been implemented in C on an open architecture traffic controller (OATC) hardware platform and demonstrated under simulated conditions for applications such as signalized intersection control, ramp metering, and communications with existing traffic control devices. System users can construct applications from a library of function blocks. The paper describes a demonstration application to freeway ramp metering control in Sacramento, CA.

I. INTRODUCTION

ROADWAY traffic control relies upon a large number of distributed microprocessors responsible for intersection signals, freeway ramp meters, traffic volume monitoring, communication and various other tasks. Effective traffic control can have significant impacts on urban congestion, air pollutant emissions and fuel consumption. Nevertheless, modernization and improvement of roadway traffic control is hampered by inadequate field computing hardware and software [6]. Existing controllers are based on decades-old control strategies; [4] and [10] describe typical techniques. This paper presents a software model that is intended to foster significant improvements in the scope and opportunity for effective roadway traffic control. In describing the software model, the special characteristics of the roadway control problem are reviewed. An example application is described for freeway ramp metering implemented as a demonstration on Route 52 in Sacramento, CA.

Most distributed control systems are comprised of several hierarchical levels of control. Ideally, roadway traffic control systems are comprised of three distinct control layers: local, supervisory, and strategic (Fig. 1). The local controllers are used to monitor inductive loop vehicle detectors, to sequence signal phasings, and to archive a limited amount of historical data. To mitigate the impact of local controller failures, an individual controller is typically used to control a single intersection or freeway entrance ramp. Coordination of adjacent controllers and synchronization of arterial routes is implemented using supervisory controllers called "on street masters" or "field masters." A single supervisory controller may be connected to a dozen or more local controllers in a particular zone or along a corridor. A third level of control provides strategic guidance to the "field masters." At the present time, most strategic planning is provided by human operators in traffic operation centers.

Local and supervisory controllers are typically based on one of two different hardware platforms. One family of controllers, commonly called NEMA units, are built with connectors conforming to standard mechanical and electrical connectors. Suppliers compete based upon the internal hardware and software. Since the units have standard connectors, an agency can migrate to another controller by unplugging one unit and plugging in a new one. However, due to additional proprietary sockets added to the NEMA TS1 (Fig. 2) units and non-standard communication protocols, this interchangeability is not realized in practice. The 1988 NEMA TS1 standard has recently been updated (NEMA TS2 Type 1 and NEMA TS2 Type2) to address shortcomings of the NEMA TS1 standard and incorporate a 40 character, 16 line display for interacting with the controller. However, the software on all NEMA controllers remains proprietary and cannot be ported by the customer. Since most public agencies are required to solicit competitive bids, engineers and technicians must learn and maintain controllers based upon several different software models. Furthermore, if a particular installation requires software modifications, manufacturer participation is required. This is often unavailable, administratively very difficult, or prohibited by competitive bid regulations.

A second family of controllers, referred to as the Caltrans Type 170 controllers (Fig. 3), are built to provide both standard connectors and portable software. The philosophy
of this standard is to develop a very precise specification for a traffic control microcomputer. Suppliers are selected periodically through competitive bidding. This standard has been tremendously successful for the past twenty years and has seen several improvements over the years, including adding a serial port, adding additional memory, and changing ROM sizes. A distinguishing feature of 170 controllers is their program module. This program module is an insertable card with a ROM that stores the traffic control program. This module can be removed from one manufacturer's 170 controller, inserted in another controller, and the software will run without modification. Instead of relying on embedded user interfaces (as in the NEMA controllers), the 170's are typically configured by connecting a PC to serial port for downloading a control strategy. Alternatively, configuration codes can be keyed in on a hexadecimal keypad.

In general, larger states such as California and New York have preferred the Caltrans 170 platform because they could control the software development and competitively purchase functionally identical units. However, the Caltrans 170 standard is beginning to age. First, the software is written entirely in assembly language. The complex nature of assembly language development precludes all but the largest cities and states from maintaining a software staff for making software configuration changes other than changing parameters. Second, no operating system is employed. Routine chores such as task scheduling and communication semaphores must be re-invented. The "home-grown" executives that have evolved preclude sharing of new control strategies since the strategies are not based upon any standard operating system. Third, the 170 hardware is based on decades old designs with a relative slow processor and limited memory. It is unclear how much longer manufacturers will support the embedded 6800 microprocessor.

To address these deficiencies, the state of California is in the process of developing a specification for a new controller that would address many of these issues [16]. The initial Caltrans 170 specification detailed every component of a special purpose microcomputer. In contrast, the current development focuses on adopting an open architecture platform composed of existing industrial computing components. The new specification will likely involve a 3U VME chassis, a 68000 family of CPU boards, the OS-9TM operating system, and a collection of modular I/O boards. This Open Architecture Traffic Controller (OATC) provides several important hardware advances, but has not adequately addressed the software development obstacles commonly encountered by traffic engineers. The following sections of this paper i) briefly summarize issues in roadway traffic control, ii) describe an adaptation of a synchronous data flow model to traffic engineering, iii) discuss verification issues, and iv) present an application of the model to freeway ramp metering.

II. SOME CHARACTERISTICS OF ROADWAY TRAFFIC CONTROL

Identical volumes of roadway traffic may be accommodated at substantially different levels of service. A somewhat simplified example of the flow characteristics of a roadway appears in Figs. 4 and 5 showing the relationship between volume (or flow) and speed and volume and density respectively. Due to the variation in inter-vehicle headways at different speeds, any travel on the roadway will lie (approximately) on this curve, with free flow travel having low density, low volume and high speed. Beyond a certain density level and critical speed (marked as \( k_0 \) and \( U_0 \) in Fig. 5), both traffic volume and speed decrease. This is the regime of congested, "bumper-to-bumper" travel observed on numerous urban freeways. An objective of freeway traffic control is to avoid such congested regimes by restricting entry to the freeway (e.g. by ramp metering), avoiding temporary bottlenecks, or other measures.
Traffic intersections also have the potential for major traffic congestion, particularly if queues of vehicles can spill over to affect adjacent intersections. Intersections typically have lower flow capacity than roadways since the passage space must be shared over time with different directions of travel. In the extreme case of congestion, "gridlock" may occur with queues from one intersection preventing dispersion from adjacent intersections. At low densities, issues of allocating green time to different directions or turning movements must be addressed for individual intersections. Also, co-ordination of green time to enable steady progression along a travel corridor arterial can be extremely beneficial.

The mechanisms available for roadway traffic control are typically limited to indirect levers such as traffic signals, lane closures and speed limits. Individual drivers make all departure time, travel, route choice and destination decisions for vehicles. The result is a "user equilibrium" of travel in which individuals attempt to minimize their own travel cost. In making such decisions, drivers ignore the "external" congestion costs their decisions impose on other users, so the resulting traffic pattern does not represent an efficient or "system optimal" equilibrium travel pattern. Both the user and system equilibrium travel pattern can be identified from a mathematical optimization problem [18] with nodal and roadway link continuity constraints. For the user equilibrium, the objective function consists of the integral of all travel time over an entire network of travel paths. In addition to the limited control mechanisms available, individual travel decision making and the resulting user equilibrium imply major difficulties for traffic control and transportation planning. Due to the non-convexity of the user equilibrium flow pattern, local improvements such as roadway capacity expansions, new roadway links or local improvements to traffic signal timing may result in a degradation of travel times overall. In the traffic engineering literature, this phenomenon is called Braess' paradox from the observation of a street improvement in Germany that resulted in increased travel times [5, 14].

\[
\begin{align*}
\text{Volumes: } & v_{AC} = 1,000 \text{ vehicles per period, } v_{BC} = 500 \text{ vehicles per period} \\
\text{Travel Time versus Volume Congestion Relationships:} & \\
& \begin{cases} 
\cdot t_{AC} = 25 + (0.01)v_{AC}, \text{ where } t_{AC} \text{ travel time on link AC in minutes} \\
\cdot t_{BC} = 5 + (0.04)v_{BC}, \text{ where } t_{AC} \text{ travel time on link AC in minutes} \\
\cdot t_{AR} = 5 + (0.01)v_{AB}, \text{ where } t_{AR} \text{ travel time on link AB after construction, in minutes} 
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{Case 1} & \\
& \begin{cases} 
\cdot t_{AC} = 25 + (0.01)(1000) = 35 \\
\cdot t_{BC} = 5 + (0.04)(500) = 25 \\
\cdot \text{Total System Travel Time} = (1000/35) + 500/25 = 47.500
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\text{Case 2 with New Link AB} & \\
& \begin{cases} 
\cdot \text{User Equilibrium and Continuity Conditions:} \\
\cdot t_{AC} = t_{AB} + t_{BC} \\
\cdot v_{AC} + v_{BC} = 1000 \\
\cdot v_{BC} = 500 + v_{AB} \\
\cdot \text{Link Solutions:} \\
\cdot \text{Equilibrium Volumes: } v_{AB} = 83, v_{AC} = 917, v_{BC} = 583 \\
\cdot \text{Travel Times: } t_{AB} = 6, t_{AC} = 34, t_{BC} = 28 \\
\cdot \text{Total Travel Time} = 48,300
\end{cases}
\end{align*}
\]

Fig. 6. Illustration of facility Improvement with Travel Time Deterioration.

In addition to these technical problems in roadway traffic control, there are also financial and organizational barriers to modernization and improvement. Improvements in traffic control will result in savings to the traveling public and society at large, but there are no direct revenues returned to responsible local public agencies reflecting such quality improvements. At the municipal level, traffic control expenditures often must compete with other obligations such as police or transit service subsidy. As a result, improvements that may have substantial
social net benefits may be foregone due to financial constraints or a lack of motivation at the local level.

III. A SOFTWARE MODEL FOR TRAFFIC ENGINEERING

Transportation control software systems can be broadly classified as "reactive systems" which interact with their environment via sensors and actuators. Real-time programming techniques are required in portions of a reactive program for operations such as Input/Output communication, precise timing, and strict access control for critical code segments. For example, when developing software for serial communications, it is essential to remove characters from the input port with sufficient speed such that subsequent incoming characters do not overwrite the previous character in the port before it is stored in an input buffer. Reactive systems are usually built using one of four basic computing models:

- **Ladder Logic.** Ladder logic is oriented toward constructing electrical interlocks using a graphical language that resembles a ladder with several rungs. This works very well for replacing relay logic that was previously constructed using large electrical relay panels. However, it is not well suited to data collection, processing of analog values or instrumentation.

- **Transaction Processing.** Transaction-oriented programming is oriented toward a state machine model of a controller where the controller changes states in response to particular inputs. A state machine refers to a model of a machine or process that can be described by a finite number of states (states) and conditions which cause transitions between states. This type of model is useful for digital sequencing and interlocking operations. For example, the phase sequencing at an intersection can be implemented using this type of controller model.

- **Transform Processing.** Transform-oriented programming is useful for control applications which can be effectively modeled by data streams and transforms. Transform processing can range from simple mathematical flow transforms to complex digital sequencing triggered by external data flows. This type of model can support transaction and ladder logic if the appropriate "transforms" are available.

- **Procedural Programming.** Procedural programming in assembly language is the state of the practice in constructing traffic control software. This approach provides a great deal of flexibility, but in practice this flexibility cannot be realized because much of the software development effort is directed at developing a basic infrastructure of modules for controlling serial ports and scheduling tasks. Most companies in the commercial sector are now abandoning this practice in favor of one of models described previously because real-time programming is still a bit of an art and the resulting code can be extremely difficult to maintain [1].

In selecting one of these programming models for constructing real-time roadway control software, several critical issues must be considered.

Portability: To insure portability, the control concepts must be completely decoupled from the physical instrumentation. Consequently, it is desirable to develop a standard roadway traffic control programming environment which could insure that decoupling. Using current C and assembly language development environments, it is not possible to prevent a developer from adding non-standard code that could be difficult to export or build upon. Examples include: using "just one peak" at an I/O port in a critical section of code or "just a few" global variables.

Maintenance: Several issues are involved in building "maintainable" software, including: software models, design documents, coding style, comment style, modular design, accurate "as built" documentation, and so on. Most of these issues can be adequately addressed by detailing standard software development practices. However, the concept of modular design is difficult to address by such a standard because different developers are likely to partition modules quite differently. In terms of maintenance, it is undesirable to maintain different software installations performing similar operations, but based upon different modules. To prevent this condition from occurring, it is necessary to restrict the developers tool box to a collection of modules which can be assembled to form a control and/or data acquisition strategy [17].

Configuration: A natural approach for an engineer or developer is to sketch a conceptual solution [2]. Sketches are semantically very rich and are useful for conveying the engineer's or developer's approach to the problem. However, to implement a sketch, it is necessary to translate this sketch into an executable form. Automating this translation from a sketch to execution could increase the reliability and development speed [2]. This translation is currently not possible in transportation control systems because there is no formal language upon which to develop these sketches. In contrast, there is a near one to one relationship between P&ID diagrams and the iconic function blocks used to implement those systems.

Software instrumentation: To understand the operational behavior of control software it is important to be able to effectively instrument it. This instrumentation typically involves inserting software probes that can collect, display, or chart relevant data. Traditional textual based programming language provide little support for this task other than a brute force approach.

All of the programming models (ladder logic, transaction processing, transform processing, and procedural programming) discussed above could be applied to transportation. In general, the ladder logic programming model is the most restrictive (in terms of what can and cannot be implemented) and the procedural programming has the greatest flexibility. Based upon the design criteria (portability, maintenance, easy configuration, and software instrumentation), a particular transform processing model called data flow programming appears to be the most desirable, at the expense of some flexibility. This approach is used successfully by several process control and factory automation software vendors [8]. Control algorithms are configured by connecting together various function blocks using icons in a CAD type configuration program on a personal computer. Once a control strategy has been completed, it can
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. Although this approach does not provide the greater flexibility afforded by a procedural language such as C, it provides sufficient flexibility to accommodate typical technical, environmental, and operational policy variations. Conceptually, this architecture provides a compromise between simply changing a few parameters in an existing signal control program, and developing a completely new program from scratch in assembly language. This approach holds considerable promise for the traffic control community.

The most fundamental issue in developing a function block programming technique for traffic control is defining the function block vocabulary and grammar for inter-connection of blocks to form control strategies. For our purposes, we developed and coded the set of function blocks summarized in Table I [7]. These blocks can be combined to perform a wide variety of traffic control tasks, even though they do not represent a comprehensive set of blocks. Before describing a particular application, we first formally define the function block language and implementation.

IV. DEFINITION OF THE FUNCTION BLOCK TRAFFIC CONTROL MODEL

A function block strategy is simply a collection of directed graphs where the function blocks are the nodes and the data flows are the arcs as illustrated in Fig. 7. A convenient description of a graph contained in a strategy is a topologically sorted list of blocks:

\[ A = \{a_1, \ldots, a_n\} \]

the set of activators

\[ F = \{f_1, \ldots, f_n\} \]

the set of data flows between the activators.

The data flow diagram corresponding to A and F is a universe of discourse (a network of interconnected function blocks) where A is used to represent function blocks and F is used to represent the connections between function blocks [9]. Connections are described using two predicates source and destin. Source is used to indicate which activation function \( a_j \) is at the source of a data flow \( f_i \), and destin is used to indicate the activation function \( a_k \) at the destination of a data flow \( f_i \). Precedences, we can define three unique connection modes:

- A data flow \( d \) is said to be a block connection if \( \exists \alpha [\text{source}(d, a)] \land \exists \beta [\text{destin}(d, a)] \). It is possible for this connection mode to have the same block as both the source and destination. This data flow can only be read by an external operator interface client.
- A data flow \( d \) is said to be an external input connection if \( \exists \alpha [\text{source}(d, a)] \land \exists \beta [\text{destin}(d, a)] \). This data flow

Table I: Function Block Summary

<table>
<thead>
<tr>
<th>Function Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-SWCH</td>
<td>Selects between two analog signals based on the state of a digital input</td>
</tr>
<tr>
<td>A-UI</td>
<td>Provides an operator with a mechanism to enter an analog value to a user interface</td>
</tr>
<tr>
<td>Filter</td>
<td>Provides a simple discrete approximation for a first order analog filter</td>
</tr>
<tr>
<td>Test</td>
<td>Compares an input against a set of absolute high and low bounds or relative to another signal. The result of these comparisons are digital points other blocks can connect to. It is useful for implementing conditional logic</td>
</tr>
<tr>
<td>Sel-H, Sel-L, Sel-M</td>
<td>High, low and middle selector blocks. The first two blocks have two inputs, the middle selector requires three inputs</td>
</tr>
<tr>
<td>RMSB</td>
<td>Provides supervisory rate selection for a ramp meter based upon an upstream volume sensor and up to six downstream occupancy values</td>
</tr>
<tr>
<td>LOOKUP</td>
<td>Provides a interpolated lookup table for defining non-linear transformations</td>
</tr>
<tr>
<td>D-Coll</td>
<td>A Coll occurs up to eight times (Analog or Digital) and records then state to a file. A background scheduler is set up so that file can reside on any disk file device. These devices include hard disks, floppy disks, RAM, and non volatile disks</td>
</tr>
<tr>
<td>RMDL, RMDO</td>
<td>Used to read digital inputs (DI) or write digital outputs (DO) on a 170 running ramp metering software</td>
</tr>
<tr>
<td>RMRI, RMRO</td>
<td>Used to read register inputs (RI) and write register outputs (RO) on a 170 running ramp metering software</td>
</tr>
<tr>
<td>VMS</td>
<td>Contains up to 8 prioritized messages that can be fired by triggering an external digital input</td>
</tr>
</tbody>
</table>

Fig. 7. Typical data flow program.

and a list of connections between those activators.

\[ F = \{f_1, \ldots, f_n\} \]

the set of data flows between the activators.
can be read or changed by an external operator interface client.

- A data flow \( d \) is said to be an external output connection
  if \( s_2, [\text{source}(d, a)] \land \neg s_3, [\text{dest}(d, a)] \). This data flow
  can only be read from an external operator interface client.

To provide a link between the data flow diagram and the
external world, a special class of device interface blocks is
required to interact with sensors and actuators. For example,
an activator that polls loop detectors, changes signal phasings,
communicates with a peer controller or archives data to a
journal file would be considered an external device and
described using the predicate \text{extern}. Similarly, blocks that
only interact with data flows can be described using the predicate \text{intern}.

For example, given the data flow diagram depicted in
Fig. 7, the set of activators would be \( A = \{a_1, \cdots, a_8\} \),
and the set of data flows would be \( F = \{f_1, \cdots, f_9\} \).
The set of block connections passing information between
blocks would be \( F_{\text{conn}} = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\} \);
an operator interface client could be used for reading these
values. The set of output only connections would be \( F_{\text{output}} = \{f_{10}, f_{11}\} \); an operator interface client would only be able
to read values associated with these connections. The set of
input only connections would be \( F_{\text{input}} = \{f_{12}, f_{13}\} \); an
operator interface client could be used for reading or writing
these values.

The set of external activators would be \( A_{\text{extern}} = \{a_1, a_2, a_3, a_7, a_8\} \) and the set of process activators would be
\( A_{\text{intern}} = \{a_4, a_5, a_6\} \). Each of the activators corresponds
to an activity or algorithm. The activators in \( A_{\text{proc}} = \{a_4, a_5, a_6\} \) deal only with data flows present in the software
model. In contrast the activators in \( A_{\text{ext}} \) correspond to
algorithms that know how to retrieve information from or
transmit information to physical devices external to the
software model. Since some activators interact with external
deVICES, which often impose timing constraints, it is important
to distinguish between extern and intern blocks.

Since data flows are not static, a reference to a particular
data flow must include a time reference. For example, \( F_t = \{f_{1, t}, \cdots, f_{m, t}\} \) is different from \( F_{t+T} = \{f_{1}, \cdots, f_{m}, t+T\} \)
if the activators have run between \( t \) and \( t+T \).

Each activator \( a_i \) is completely defined by block record \( B \),
containing four pieces of information:

- \( L_i \), local storage containing flags, parameters, calculated
  values, variables defining boundaries, and values explicitly saved from previous computations.
- \( I_i \), the input flow references, defining which inputs flows
  the activator must read to evaluate its algorithm. These
  flows can only be read, they can not be written to. This
  set of input flows can be formally defined as the set of
  flows \( F \), that have activator \( a_i \), for a destination:

\[
I_i = \{x \mid x \in F \land a_i = \text{dest}(x, a_i)\} \quad (1)
\]

- \( E_i \), the evaluation function used to update output flows,
  and local storage based upon input flows, previous output
  flows, and previous local storage.
- \( O_i \), the output flow references identifying the flows that
  will be updated by the evaluation function. These flows
  can be read from and written to. This set of output flows
  can be formally defined as the set of flows that have
  activator \( a_i \) for a source:

\[
I_i = \{y \mid y \in F \land a_i = \text{source}(x, a_k)\} \quad (2)
\]

New output data flows emitted from activator \( i \) at time \( t \)
are calculated by applying an evaluation function \( E \) to the
input sockets \( I_i, t \), the local variables calculated the previous
scan \( I_i, (t-T) \), and the output sockets calculated the previous
scan \( O_i, (t-T) \):

\[
O_{i, t} = E_i(I_{i, t}, I_{i, t-T}, O_{i, t-T})
\]

Rather than specifying the processing period \( T_i \) for each
activator \( a_i \), it is desirable to collect all the activators
that should be executed at a periodic rate \( T \), into a group.
This group is assigned a Period \( T \) that defines how often
the collection of blocks in a group should be executed.
Furthermore, to provide the capability for controlling the
execution of a group of blocks, a digital status connection is
provided. When the status control signal is active, the blocks
are executed at their periodic rate \( T \). When the status control
is low, the blocks are never executed. A group is completely
described by the following information:

\[
G = \{D, P, S\} \quad (3)
\]

- \( D \), a topologically sorted set of block records \( B \),
describing the input flows, local storage, output flows, and
evaluation function for each activator. Since this set of
blocks describes a directed graph, it is essential that this
list be topologically sorted such that all input flows are
updated before an evaluation function is run for a block.
Cycles are permitted, but a starting point must be specified
for each cycle and the blocks in a cycle are only executed
once per period.
- \( P \), defines the period of the group and specifies how
frequently the group of blocks shall be scheduled for
execution. This allows the developer to specify strategies
where important blocks are run frequently (such as for
a vehicle counter) and less important blocks, requiring
more computation, run less frequently.
- \( S \), is the status connection signaling the group whether
or not to execute the set of group records. This can be
useful for turning blocks on or off depending upon
characteristics.

A task \( \tau \) is defined as a collection of groups \( G \), such that all
groups have the same period. For example, if groups \( p, q \)
and \( r \) all have the same period \( T_i \), then:

\[
\tau_i = \{G_p, G_q, G_r\} \quad (4)
\]

The complete strategy is defined to be a set of all periodic
tasks:

\[
S = \{\tau_1, \cdots, \tau_t\} \quad (5)
\]

where the tasks are sorted such that the period of \( \tau_1 < \tau_2 < \cdots < \tau_t \).
Each task $T_i$ in the strategy $S$ must run all groups and sibling blocks composing it on some periodic basis as depicted in Fig. 8. Each group operates like a clocked digital circuit. Groups are scheduled using the rate monotonic scheduling algorithm [12].

V. EXAMPLE APPLICATION: FREEWAY RAMP METERING

A. Local Ramp Metering

Consider a freeway entrance that requires locally responsive ramp metering (Fig. 9). Although this is a relatively common traffic engineering problem, nearly every agency has developed through an evolutionary approach their own control model to address local geometric and policy constraints. Since all the control software is written in assembly language, it is very difficult to share control concepts or explain exactly how a particular technique was implemented. The following example was developed to show how the function block model for the OATC could be applied to this problem.

The control model selected for this example monitors the volume and occupancies from a set of loops adjacent to the ramp, computes a red interval based upon those volume and occupancies, and then selects the most restrictive of the two intervals. Since this is only a local control algorithm, it cannot provide an optimal metering rate for the system. However, when the cost of installing a completely networked system is prohibitively expensive, this technique is frequently used [12].

The actual implementation of this strategy is shown in Fig. 10. In this strategy, the dotted line around the blocks in the upper left corner is used to denote the blocks belonging to the "CONTROL" group, which is used to monitor the detectors and estimate traffic parameters. The "SENSOR" group is specified to run every 0.02 s (50 Hz). The blocks NB_PRS1 and NB_PRS2 monitor the presence of outputs from the adjacent loop detectors. The OCC1_SEL and OCC2_SEL blocks are switches that output either 0.0% or 100.0% if a vehicle is detected, or not detected, respectively. The AvgOcc block has a gain (multiplier) of 0.5 assigned to each input, so each scan it outputs the average occupancy across all lanes. This instantaneous occupancy is connected to the OCC_AVG to compute a moving average using a 25 point sample size.

The blocks NB_PUL1 and NB_PUL2 monitor the pulse outputs of the adjacent loop detectors. The VPS1 and VPS2 block compute the rate, in vehicles/s, of the incoming pulses. The AvgVPS block has a gain of 0.5 assigned to each input, so each scan it outputs the average vehicles/s per lane passing over the detectors.

The remainder of the strategy shown in Fig. 10 belongs to the group "RampCtl", which runs every 0.5 s (2 Hz). In this group, the average occupancy computed by the OCC_AVG block is filtered in the OCC_FILT block to remove "bumps" in the real time data. The filter used in this block is an approximate first order algorithm and the time constant can be set to a constant or changed dynamically via a connection into the block. The HzToVPH block converts the vehicles/s output by the AvgVPS block to vehicles per hour. That output of the HzToVPH is connected to the VOL_FILT block, which operates similar to the OCC_FILT.

The filtered occupancy is fed into the OCC_CTL, a lookup table that computes a recommended ramp meter red interval based upon the filtered occupancy. The operation of this block
is depicted in Fig. 11. The block is configured by specifying several x-y pairs defining a piecewise linear function. Every time the block runs, it computes an interpolated output using the lookup table and input value. The VOL_CTL block operates in a similar manner except instead of using a lookup table based upon occupancy-red interval pairs, the lookup table is defined by volume-red interval pairs. The outputs of the two lookup tables (OCC_CTL and VOL_CTL) are read by the SEL_RED block, which chooses the larger of two red intervals (most restrictive) for its output. The output of this block is the desired red interval for the ramp meter.

This red interval time is connected to two blocks, the signal drum sequencer called Meter and the test block called MinTst which shuts the meter down when the red interval is less then 7.2 s. The blocks named IntrLck and TurnOn connected to the drum sequencer provide the interlock that prevent the signal from shutting down during the red interval. The D_UI blocks named RED and GREEN are connected to the contacts controlling the lamps. The MESSG block names Status is used to display a message indicating if the ramp meter is on or off.

The strategy shown in Fig. 10 might need modifications in particular instances. For example, many cities and town demand that they be permitted to dump vehicles on the freeway at an accelerated rate when a ramp begins to spill back into local streets. A modified strategy providing this capability is shown in Fig. 12. In this figure, the D_UI block named Q_PRS monitors a loop at the ramp entrance (Fig. 9). To filter out transient pulses, this block is connected to a DLY_ON block named Debounce that delays turning on until the output of Q_PRS remains high for a specified duration (10s for this example). The output of the Debounce block is connected to the DLY_OFF block named QUEUE which remains on 60 s after its input, from the Debounce block, was last on. The output of this block is used by the strategy to determine if a queue exists. The reason for providing the added delay before turning the queue detector off is to prevent "chatter" during congested periods.

A safe red interval for dumping the vehicles is computed by the LOOKUP block named SAFE_RED. The lookup table used in this block differs from that used in VOL_CTL since it computes a minimum safe red interval instead of trying to maintain the optimal ramp meter strategy with queue override.

B. Example Supervisory Freeway Ramp Metering Control

Consider the problem of installing a supervisory computer to adjust metering rates for entrance ramps on a freeway. Although this type of control is a logical extension of the local ramp metering strategies discussed in the previous section, no standard controllers capable of providing supervisory control of freeway systems are available [3]. A few cities such as Minneapolis, MN and New York City have developed, either in house or with a contract to a system integrator, proprietary systems over the past 10 to 15 years. Since there are no standard controllers capable of meeting this need, this section introduces an example function block strategy used to construct the supervisory logic for adjusting the metering rate on a single ramp meter. The supervisory computer is assumed to interact with several controllers and local controllers are responsible for cycling the ramp signals according to metering rates received from the supervisory controller. This application was demonstrated in the fall of 1992 in Sacramento, CA Route 52.

The basic problem of supervising a single ramp meter is illustrated in Fig. 13. Detector stations are spaced at regular intervals along the freeway and provide volume and occupancy measurements every 30 s. A traffic signal on the entrance ramp permits vehicles to enter the freeway when the light is green. A supervisory algorithm that monitors relevant volume and occupancy measurements adjusts metering rates so as to maximize the flow of vehicles along the freeway, given...
same congestion risks (Fig. 5). The supervisory logic chosen for this example is based upon the bottleneck algorithm implemented in Minneapolis's custom supervisory controller. In this model (Fig. 14), an algorithm associated with each ramp meter monitors up to six down stream occupancy stations and one up stream volume station (Fig. 13). The down stream occupancy stations provide the feedback signals and the up stream volumes provide the feedforward (anticipatory) signals. The algorithm examines these parameters every 30 s. For each downstream occupancy station, the algorithm has a look up table defining appropriate metering rates for various occupancy values. Similarly, for the upstream volume detector, the algorithm has a look up table defining appropriate metering rates for various incoming volumes. The values computed from these seven lookup tables are each passed through a first order filter (with configurable time constants). This prevents sudden bumps in metering rates. Then, the most restrictive metering rate is selected. The most restrictive rate is added to a filtered manual bias. This bias is input by an operator observing freeway operations at critical freeway locations over closed circuit TV. This manual fine tuning of the controller is particularly important since traffic flow characteristics vary significantly and at this time cannot be predicted with sufficient accuracy [13, 11]. By using the bias input, the algorithm still provides coarse corrections, but the operator can slew the metering rate up or down as conditions warrant. It is this same functionality that we sought to encapsulate in a single Ramp Meter Supervisory Block (RMSB).

Fig. 15 illustrates the function block strategy that makes use of the RMSB block. The A/UI blocks names UpVol1, DwnOcc1, DwnOcc2, DwnOcc3, DwnOcc4, DwnOcc5, and DwnOcc6 provide the necessary real time data inputs (Fig. 15). The output of the RMSB block is transmitted to the local controller responsible for cycling the ramp meter signals. This group of blocks, named SuperV, is configured to execute once every thirty seconds. Of course, suitable look up tables defining occupancy-metering rate and volume-metering rate relations must be configured in the RMSB block.

The strategy depicted in Fig. 15 depicts a simplified strategy for supervising only one ramp. In a full implementation, the example group (SuperV) would be replicated several dozen times. Each group would be assigned a meaningful name (such as NB80E26a for entrance 26a on the northbound lane of interstate 80), the member blocks would be configured to retrieve the appropriate volume and occupancy values from field controllers, and the RMSB block would be configured with the appropriate lookup tables. In theory a strategy constructed in this manner should provide autonomous closed loop operation. However, cities that have implemented these systems have found that an operator must monitor the system and "tune" the controllers in response to non-recurrent events such as weather, ball games, and incidents. This is not surprising since complex industrial processes require similar tuning to respond to un-modeled events.

VI. Conclusion

Assuming the current developments in the area of transportation controller hardware and software continue it is reasonable to envision the emergence of a controller with the following characteristics:

1) A specification for an OATC that can be configured from a family of modular CPU's, I/O cards, network adapters, and serial ports depending upon the demands of a particular application.

2) A standard high level application development software model that can be used by traffic technicians and engineers to develop routine software such as signalized intersection control, ramp metering, data collection and so on. This software will be configured graphically on
Neither the software or hardware portion of this vision are complete at the current time. Although the hardware commercially available for such a controller, several issues remain including:

1. Developing a list of "approved" modules and vendors.
2. Defining standard mechanical and electrical connectors for approved modules.
3. Establishing OEM's to provide one-stop ATC controller shopping.
4. Commitment by state and local agency to such a platform to develop a sufficient market for multiple vendors to compete in.

The software aspect of this controller is both the most important, and challenging aspect of this controller vision. If we do not devote sufficient resources toward developing a "canned software model", software development issues will severely impede modernization efforts. This paper proposed a software model that is commonly used in the commercial sector and could provide the following benefits:

- Basic software infrastructure. All controllers will have a uniform look and feel regardless of the application they are running.
- This model provide a much safer route for incorporating changes since the "application" is customized, not the real time programming model.
- Function block diagrams provide a mechanism for exchanging control concepts.
- Economically it would not be feasible for states to each develop their own ATC software. To avoid getting locked into a single software vendor, it is critical that the block definitions and interface protocols be precisely specified. This will permit competitive bidding of independent modules such as the graphical configuration tool, the real time kernel, and real time user interface tools.
- Structured nature of blocks provides a formal framework for traffic engineering to define/request new or revised blocks. As this software model grow, it is conceivable to imagine a TRB or ITE committee overseeing this development.

ACKNOWLEDGMENT

The authors wish to acknowledge several suggestions from reviewers which clarified the presentation. Financial support from Caltrans, the California Department of Transportation, in the initial work leading to this paper is gratefully acknowledged. The views expressed by the authors do not necessarily reflect the views of the California Department of Transportation.

REFERENCES


Darcy Bullock received the B.S. degree from the University of Vermont in 1987, and the M.S. degree in 1988 and Ph.D. degree in philosophy in 1992 from Carnegie Mellon University.

He is an Assistant Professor in the Department of Civil and Environmental Engineering at Louisiana State University. His research and teaching interests are in the general area of IVHS. Over the past several years he has been actively involved in developing the hardware and software architecture for the next generation of traffic controllers.

Chris Hendrickson received the B.S. and M.S. degrees from Stanford University, and the Ph.D. degree in economics from Oxford University, and a PhD from the Massachusetts Institute of Technology. He is a Professor of Civil and Environmental Engineering and Associate Dean of the Carnegie Institute of Technology at Carnegie Mellon University. He is also Education Director of the Engineering Design Research Center, a National Science Foundation and industrially supported Engineering Research Center. His research, teaching and consulting are in the general area of engineering planning and management, including design for the environment, system performance, finance and computer applications. He has co-authored two textbooks, Project Management for Construction (Prentice-Hall, 1989) and Transportation Investment and Pricing Principles (John Wiley & Sons, 1984) and two monographs, Knowledge Based Process Planning for Construction and Manufacturing (Academic Press, 1989) and Concurrent Computer Integrated Building Design (Prentice-Hall, 1994).