CHAL - Control logic / Hardware Abstraction layer

Marco Zennaro and Raja Sengupta

WORKING PAPER
UCB-ITS-VWP-2007-6

October 2007

TTCS – Tools for the development of traffic control systems

CHAL - Control logic / Hardware Abstraction layer

Marco Zennaro and Raja Sengupta

UC Berkeley Center for Future Urban Transport
Civil Systems program, University of California, Berkeley
Email addresses: {zennaro, raja}@path.berkeley.edu

Abstract

Traffic control systems have reached a high level of sophistication: they are general purpose machines that can, in principle, run any traffic control software. The firmware they are running turns them into special purpose machines able to operate only according to some pre-defined rules. The firmware usually allows limited customizations through parameters, but it does not support the introduction of new control schemes. As a result, implementing a new traffic control scheme requires the re-implementation of the firmware, a complex task given the low-level programming required. The complexity of the task has created a rift between the academic community and the roadside equipment manufacturers' community. As a result, the traffic control firmware does not benefit from the work and novel ideas of the academic community and it is often sub-optimal when it comes to issue of interoperability, integration and cost-effectiveness. At the same time the efforts of the academic research are often wasted by an incorrect understanding of the underlying system: many sophisticated control schemes proposed by the researcher are never used because they are not implementable over the existing hardware. This paper introduces a software suite, the Tools for the development of Traffic Control Systems (TTCS), meant to bridge the gap between these two communities. The key concept is the introduction of the Control logic / Hardware Abstraction Layer (CHAL) between the control logic and the hardware, reducing the complexity of developing a new traffic control scheme. It enables the control designer to re-use the code developed in the design, testing and simulation phases directly on the traffic controller. Conforming to the CHAL architecture ensures that the design is implementable over the target hardware. Moreover CHAL decouples the control logic and the hardware (ensuring software portability), easing both hardware and software upgrades (avoiding the need to update the software when the hw changes and vice versa). A software library for the integration of Quadstone Paramics 5.4 into the CHAL architecture is presented.

Keywords: *Traffic control systems, Software portability, Software tools, Design Tools, Microscopic traffic simulator.*

I . Introduction

Traffic control systems (TCS) have evolved over time into very sophisticated systems. Modern TCS are quite complex when compared with the manually operated systems that appeared in London in 1868 (Wolkomir, 1986) as well as to the first automatic systems that appeared in Toronto in the early 50s.

They have grown in size, in the awareness of the surrounding environment and in the sophistication of their control schemes.

In 1917 a policeman could control up to 6 intersections with a single switch, whereas today, urban-wide traffic control systems control thousands of traffic lights (see for example the Los Angeles Adaptive Traffic Control System as described in Li and Zhang, 2005)

The first automated systems did not have sensors; nowadays modern systems often are connected to inductive loops placed underneath the pavement or radars placed on the side of the road (Lee et al., 2004).

While the automated system installed in 1950 in Toronto operated according to a simple pre-timed plan, modern systems adapt to changes of the local traffic pattern at the intersection in real time (Kell and Fullerton, 2004). Some sophisticated control systems, like SCOOT (Hunt et al., 1981) and ACTS are able to adapt to artery-wide or city-wide changes in traffic patterns.

In the 50s, traffic control systems were special purpose machines that offered very limited customization. They were able to operate only according to some pre-defined rules of operation. They were soon replaced in the 60s with microprocessor based traffic controllers. This simple step enabled a potential revolution in the field: the microprocessor-based controllers became general purpose computers. For the first time the control logic and the hardware were not, for the first time, coupled together. It became possible to upgrade the traffic control logic without replacing the traffic controller simply replacing the software.

This change can potentially result in substantial cost benefits. First of all it is now cheaper to upgrade the system since only software changes are needed (and the hardware can be re-used). At the same time traffic agencies are free to buy controllers from the cheapest vendor, since the same control scheme can be implemented over different controllers.

Unfortunately there were not enough commonalities between controllers manufactured by different vendors, so that the firmware produced by one vendor could not be used on another. A particular control scheme implemented by a vendor could only be run by a particular controller (Bullock and Urbanick, 2001).

On top of that, the hardware diversity and the low-level programming necessary to write the firmware was such to make quite difficult to implement new traffic control schemes. Usually if a transportation agency wanted a not standard control scheme it would have to go to the vendor for help. Unfortunately quite often the returned system would not behave as expected (Gitelson, 1970), making the development of these systems quite rare.

It was at this time that, as observed by Bullock and Urbanick (2001), a rift started to emerge between the “roadside equipment research area community” (governmental agencies and controller manufacturer), and the “analytical-type operation research area community”, (mainly universities). As a result, the traffic control firmware does not benefit from the results and novel ideas of the academic research and it is often sub-optimal when it comes to issue of interoperability, integration and cost-effectiveness. At the same time the efforts of the academic community are often wasted by an incorrect understanding of the underlying system: many sophisticated control schemes proposed by the researcher are never used because they are not implementable over the existing hardware. The rift widened over the years with the complexity of the TCSs.

The coupling between the control scheme (the software) and the hardware increases the costs associated with all the phases of the system life cycle. It increases the platform cost, bounding it to some specific hardware that is unlikely to be the cheapest available. It increases the maintenance cost, especially when one of the needed hardware components became obsolete. An even higher cost is associated at the upgrade stage, where a simple change of the control scheme may require significant hardware changes.

The controllers’ diversity was such that it created frustration amongst maintenance personnel. They were required to learn how to operate and wire a controller from scratch every time a new model was released (Bullock, 2001). At the end of the 80s the National Electrical Manufacturers Associations answered to this frustration creating the TS-1 standard (NEMA, 1989). The TS-1 standard simplified the work of the maintenance personnel, specifying the basic set of features and connectors that must be provided.

Unfortunately this did not address the software portability problem: the firmware written for a particular controller still could not be run on a different one.

Similarly the TS-2 standard (NEMA, 1992) simply expanded the features and connectors requirements to support coordination, but it did not address the software portability problem.

As the systems grew in size, sophistication and awareness, so did the associated design, deployment and maintenance complexity. Therefore, in order to simplify the design and deployment task, the TCS are developed to run only on a particular hardware. As a result, for many of the currently available TCS, the control scheme (implemented by the software) and the hardware that enable, execute and actuate it are bound together in the same package. As a result, even if the NEMA hardware is general purpose, i.e. not restricted to execute some fixed code, the software available for the vendor restricts its usage to run some particular code. In other words, even if the level of sophistication and customizations dramatically increased from the one provided by the systems available in the 50s, these systems still operated according to a pre-defined fixed set of rules of operation.

At the same time, as the controllers started to be coordinated across wide areas and entire cities, it was more and more costly to be bound to a particular vendor. Upgrading the traffic control scheme may require replacing all the controllers in the system, making the upgrade unfeasible. Even a simple system expansion is destined to be costly as long as it is not possible to choose the cheapest available vendor because of software compatibility issues.

Consider for example the Adaptive Traffic Control System (ATCS), developed by the Los Angeles Department of Transportation (LA DOT), (Li and Zhang, 2005). It is a quite sophisticated system, offering multiple modes of operation, reacting in real-time to changes of traffic pattern at intersections and city-wide, optimizing timing plans along multiple intersections along multiple arteries. Since its introduction in the 90s, ATCS has been proven to reduce the congestion and to increase the traffic network capacity utilization. As it is often the case, the system requires a particular computing, sensor and communication infrastructure. The computing platform, for example, is specified down to the firmware. A transportation agency that is contemplating to upgrade the system to run ACTS may have to change all the traffic controllers, even if they are of the correct family (i.e. 2070s). In fact, quoting directly from Li and Zhang (2005), "*it is usually more cost-effective to replace the firmware with the right model than to modify existing ones. It may take more than a year and as much as $200,000 to modify foreign firmware to work with ATCS software*". As a result, even if the ACTS traffic control scheme could potentially be implemented to run on any modern traffic controller, a transit agency will still have to replace all its infrastructure once (incremental upgrades are not often an option) if it wants to use the ATCS system.

A similar problem was experienced by the LA DOT itself when the ATCS was upgraded to deal with traffic priorities. The existing expensive communication infrastructure could not be modified to accommodate the new needs. As a result the Traffic Priority System (TPS) had to rely on a separate communication network.

These examples show that the tight coupling between traffic control scheme and hardware lead to high costs.

The first attempt to address the software portability problem was initiated by the California department of Transportation (Caltrans) and the New York Department of transportation (NY DOT). The effort lead to the definition of the 170 standard, that specified not only the electronic connectors, but also the memory mapping and the processor (the Motorola 6800), so that the code written to run on a particular 170 controller could be run on any 170 controller. Only a very limited amount of vendors decided to support the effort.

Unfortunately, even if the problem that was addressed had enormous cost benefit potential, the approach had some significant shortcomings. The first problem arises in the microprocessor specifications: the speed by which new processors are being introduced and discontinued is such that after a couple of years from the development of the 170 standard, the processor would be already outdated. The second issue is that the standard does not address the "rift" between the roadside equipment and the analytical-type operations communities: the low-level programming necessary to program the 170 is so complex to be understood only by the vendor engineers. Soon, as the systems grew in size, the low level programming became too difficult even for the equipment engineers. Quoting Li and Zhang (2005), "*the LA agency recognized the difficulty of continuing to maintain and enhance the [170-based] system because of the low-level programming language it required*".

The first issue, the processor requirement, was recognized and (at least partially) addressed in 1989 by the Advanced Transportation Controller initiative that culminated in the Model 2070 traffic signal controller specifications. Instead of requiring the controller to have a particular microprocessor, it was open to continuous processor updates, as long as they were able to run a particular operating system (OS-9). Unfortunately the second issue, i.e. the programming complexity, was only partially addressed (while the OS-9 abstract machine is indeed easier to program than a machine with no operating system, the task is still quite challenging).

In the 90s the National Transportation Communication for ITS Protocol (NTCIP) emerged, as the TS 3.5 standard (NEMA, 1996). It was a successful attempt to make modern traffic controller inter-operable. The TS 3.5 standardize the data format and the communication protocols between controllers. As a result any TS 3.5 ready controller can be interchanged with another.

Unfortunately the protocols work as long as the traffic controllers are used according to a predefined set of rules of operations supported by the standard. In addition it is not possible to implement novel ideas on top of it. As a result, the rift between the two communities (roadside equipment and analyst-type operations) is still not bridged.

This paper introduces a software tools chain for the development of traffic control systems (TTCS) aimed at bridging the divide between the two communities while decoupling the traffic control scheme from the underlying hardware. The key idea is the introduction of a Control logic / Hardware Decoupling Layer (CHAL), between the software and the hardware platform (see figure 1). The underlying hardware is transparent to the software, hence hardware upgrades and changes do not require software modification. At the same time, a software upgrade would not require unnecessary hardware changes.

CHAL abstracts all complex details from the control logic developer enabling university researchers to directly program the control scheme without having to rely on the support of the vendors. At the same time CHAL ensures software portability between any CHAL enabled controller and it does not bind the software to any particular processor, hardware or operative system (as both the 170 and 2070 standards do).

CHAL abstracts the complexity of the hardware to the control software and the complexity of the control software to the hardware.

The tools chain has been designed to simplify and support the development of TCS. It supports the design stage, integrating high level tools able to understand traffic control logics in the language they are described in the literature. It also includes traffic microsimulators, where to test and debug the design, verification tools, to formally verify the system design, and multiplatform compilers, able to compile the design to many different target architectures.

As a result the development of a new control scheme could be carried without any complex knowledge of the underlining hardware systems. At the same time any scheme implemented using TTCS is implementable as long as it conforms to the CHAL interface that is quite simple and easy to understand.

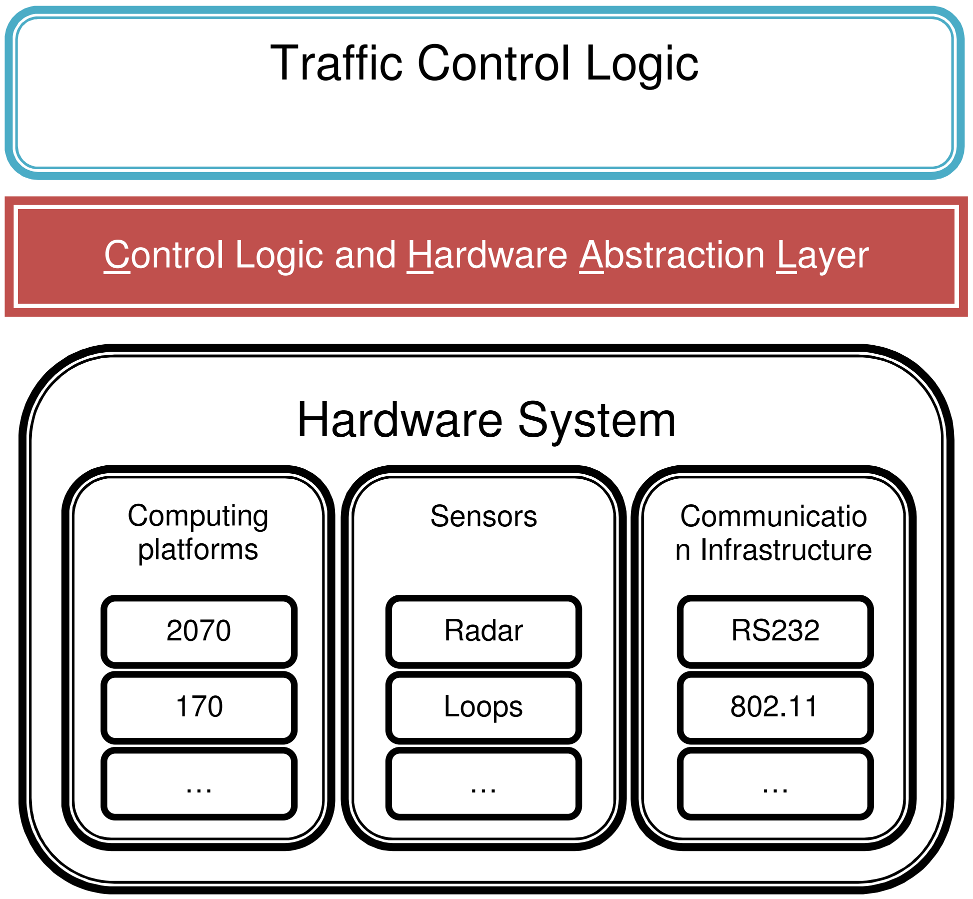


Figure 1: the Control logic Hardware Abstraction Layer (CHAL) hides the complexity of the hardware from the control logic

The rest of the report is organized as follows: section II describes CHAL, its philosophy, its architecture and the various protocols and messages formats. Section III describes the CHAL enabling library developed for Paramics.

II. Control logic / Hardware Abstraction Layer (CHAL)

This section introduces the Control logic / Hardware Abstraction Layer (CHAL). Section II.a describes the philosophy behind the development of CHAL, while Section II.b describes its architecture. Section II.c and II.d describe the protocols and the message formats used in CHAL respectively.

II.a. Design philosophy

This section introduces the Control logic / Hardware Decoupling Layer (CHAL), and its architecture.

The CHAL philosophy can be summarized as follows:

- CHAL should not be tied to any specific hardware and it must ease the introduction of new hardware technologies.
- CHAL should not be tied to any specific operating system.
- CHAL should not support a limited set of rules of operation. It should support introduction of new control schemes while ensuring safety;
- Ease and support the development of traffic control systems through all the phases of the product lifecycle;
- Bridge the gap between the "roadside equipment" and the "analytical-type operation" research communities
- Rely on over the shelf (OTS) open technology;
- Keep the system architecture simple and scalable;

The 170 standard had limited success because of the fact that it was tied up to a processor that was quickly rendered obsolete. The CHAL architecture is easily extensible and expandable, so that new sensors, computing platforms and control schemes can be added easily and seamlessly to the system. CHAL does not restrict the type of data gathered by sensors. We want to avoid situations like the one described in

Bullock and Urbanick (2001), where the data gathered by a sensor is forced to adapt to a contact open/ contact close loop detector standard, throwing away a big part of the gathered information. This is obtained by fixing the communication protocols between sensors, actuators and control logic while supporting an extensible message format.

CHAL has been designed to simplify and support the development of TCS, from the design stage to the deployment and upgrade stage.

It is not possible to test the control logic directly on the target hardware system: the traffic control grid is in use 24/7 and only thoroughly debugged controls should be allowed to run. In order to tune and test the design, simulation is often used (Lee, Nevers and Robinson, 2004). Unfortunately after implementing the scheme to work in the simulator environment is then necessary to re-implemented it to be run on the target hardware platform. This results in an unnecessary duplication of efforts. Moreover quite often after the academic researchers develop and tune their algorithms in simulation they soon discover that they cannot implement them on the roadside cabinets. They can rely on the traffic controller manufacturer to do the implementation, but often the control scheme is not understood nor is it implementable. As a result the implementation often does not conform to the design specifications (Gitelson, 1970).

CHAL addresses this problem offering an identical interface for the simulation environment and the real equipment. As a result, the control code developed and tested in the simulation environment is ready as is for the real hardware systems (see figure 2). Conforming to CHAL enables researchers to implement their novel ideas while avoiding the complexity of developing new firmware. CHAL conformance ensures that the algorithms presented to the roadside equipment vendors are implementable. Recently the first release of a CHAL library for Quadstone Paramics 5.4 has been developed by the authors and it is already available for download on the Internet (TTCS, 2006). A CHAL server which enables the remote control of NTCIP-enabled controllers is currently under development.

CHAL simplifies the design and the development stage making transparent the migration from simulation to the deployment hardware platform. It further eases upgrades because new CHAL-compliant hardware can be seamlessly substituted with another. Similarly, new control software can easily be added into the system without having to modify the existing hardware.

This approach has the advantage of simplifying the development and maintenance of traffic control systems. As a result the overall cost is kept low. The cost is further reduced avoiding the use of any proprietary technology and relying instead on OTS open technologies. This can be a source of significant savings both in the initial capital investment as well as the operating and upgrade cost, as proved recently by the experience in Queensland, Australia as reported in (ITS international, March 2006).

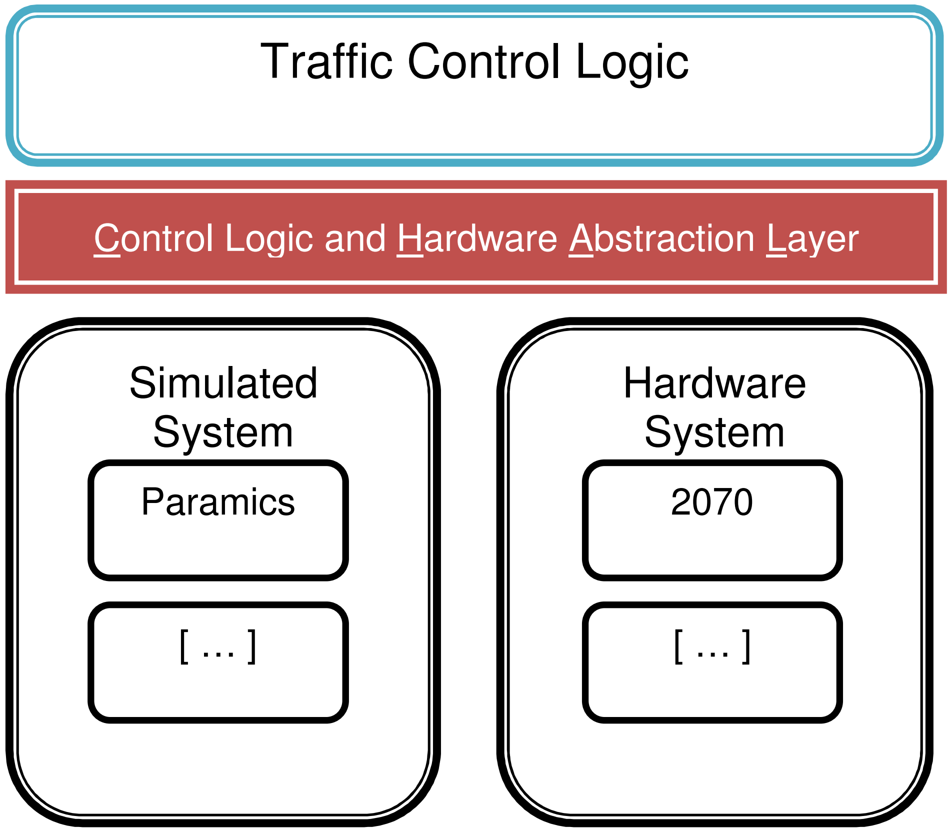


Figure 2: Switching between the simulation environment and the real hardware environment is completely transparent to the control logic because of the Control logic Hardware Abstraction Layer (CHAL)

II.b. CHAL architecture

A CHAL traffic control system consists of 3 types of components: sensors, control logic algorithms and actuators (see figure 3). Sensors are devices that gather traffic information, such as presence, speed and flow. Examples of sensors are: under-pavement induction loops, radars, roadside cameras or in-vehicle sensors broadcasting over DHRC frequencies. Traffic actuators are devices that can affect the traffic flow. Examples of actuators are traffic lights or Changeable Message Signs, like the ones currently in use in the San Francisco Bay Area (CCIT, 2006). Control logic algorithms are software components, possibly distributed, which control the traffic actuators by using the information collected from the sensors.

The interfaces between these three components are fixed in the CHAL standard. CHAL defines how the components communicate and what the minimum set of information to be exchanged is. The protocol supports different communication patterns to accommodate different situations and the message format is easily expandable to support new type of sensors, actuators and control strategies.

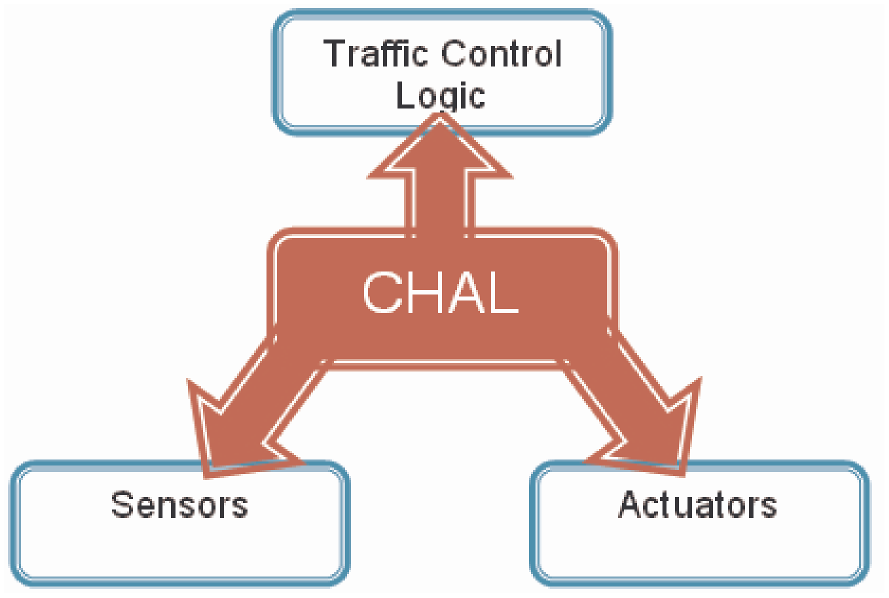


Figure 3: CHAL architecture

Each CHAL component has a unique CHAL identifier. Similarly, each intersection and each road segment between intersections has a unique CHAL identifier. This identifier is an alphanumerical string. Distinct CHAL entities have distinct identifiers.

This paper does not address how to create and ensure unique identifiers, given that the issue has been extensively studied and the literature on the topic is quite rich. The NTCIP standard relies on ISO standards for the definition of Object unique Identifier (OID) (NEMA 2002). Other simple solutions are to generate components IDs to use the MAC or IPv6 address of the device or to follow an algorithm like the one used in Jini to create unique service IDs (SUN, 2004). For roads and intersections may be sufficient to use the GPS coordinates of the center of the intersection or of the road leg.

All the CHAL communications are assumed to be delivered over a communication protocol offering two services: a non reliable broadcast and a reliable unicast. CHAL should not be tied up to any technology and need to be able to work over any communication protocol with the required semantic. A possible candidate for such communication service is the standard TCP/IPv4 protocol. CHAL can easily be implemented on top of it, hence in the following we would adopt the language used by this protocol, but CHAL does not requires TCP/IPv4 to be the communication protocol to be used.

II.c CHAL protocols

In this section we describe the communication protocols used in CHAL components. Currently the CHAL protocols described here have been implemented in libraries available for the traffic simulator Quadstone Paramics 5.4, and the programming tool for control software Mathwork Simulink (TTCS, 2006). The authors are currently working on a CHAL server for the control of NTCIP-compliant controllers.

The goal of the protocol design was to be general enough to support a wide variety of applications while at the same time not introducing unnecessary overhead over the communication network. Traditionally the communication protocol can be grouped in two categories: the reliable ones, suck as TCP and the low-latency ones, like UDP. While the protocols belonging to the first category guaranty the message delivery, the ones in the second do not (but they may offer a high probability of delivery, placing a higher burden on the communication network). The protocols in the first category introduce overhead and increase latency, while the ones in the second class are lightweight and keep latency to the minimum. Monitoring applications rely on the first type of protocols because information needs to be gathered periodically and reliably with some tolerance to delay on data delivery. Safety applications require data to be sent aperiodically and as fast as possible. In this case, reliability is too costly because the data has value only within a certain time period and any delay introduced to increase reliability increases risk. CHAL supports both classes of applications. All the CHAL communications are assumed to be delivered over two communication protocol primitives: a non reliable (generally broadcast) protocol (lightweight, low latency) and a reliable unicast protocol (overhead, higher latency). They are combined together in two high level protocols that are described in the following sections. The INT_EXT_CONT described in section II.c.1 is used by a software application to take control over a particular actuator. The SENS_PS described in section II.c.2 is used to connect and gather data from sensors.

II.c.1. INT_EXT_CONT

In CHAL, in order for a traffic control software component to take over control of an actuator (e.g. an intersection) it must follow the protocol shown in figure 4 (the INT_EXT_CONT protocol). This application can tolerate some latency but it cannot tolerate message loss. Because of that, it relies mainly on the unicast reliable protocol mentioned in the previous section.

The protocol can be described as follows: first, the software control component broadcasts on a well-known port (7605) a request for external control (REG_EXT_CON). This request contains the unique identifier of the actuator as well as some information about how frequently the control message will be sent. The actuator then will unicast back a message (REQ_ACC) informing the software that its request has been granted or denied. In order to produce a reliable and robust system, a standard time-out retransmission protocol is implemented on the external controller side (since we assume that broadcasted messages are not reliably transmitted by the communication protocol).

When the REQ_ACC packet is received the external control software can start unicasting the phase data (PH_SET) to the actuator, at the specified rate. It is important that the external controller produces the phase data at the rate it declared in the registration phase since the actuator will process the data at that rate. If data is not received at the specified rate, possibly because the network is unable to support such a data rate, the actuator server will notify the sender and roll back in the default safe mode (e.g. flashing red or pre-timed plan for traffic light).

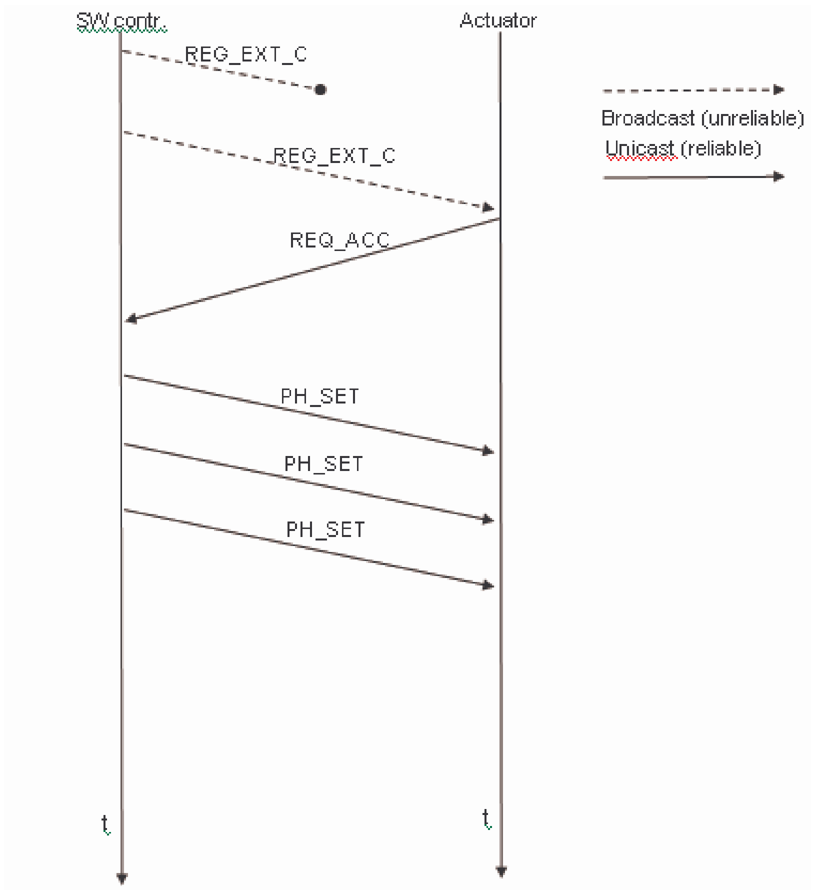


Figure 4: the INT_EXT_CONT protocol

II.c.2. SENS_PS

In order for the external software to have access to the sensor data it must follow the protocol shown in figure 5 (the SENS_PS protocol). Different applications can tolerate different amounts of overhead and latency. Because of it the protocol offers both a low-latency unreliable mode and a reliable mode with a higher latency. The first one will be used by safety applications, where the delay due to retransmission can have undesirable consequences, while the second will be used by less time sensitive applications, like monitoring applications.

The protocol can be described as follows: first, the external control software component broadcasts over a well-known port (7606) a request to query data from a particular sensor or sensor set (SUB_SENS). The request specifies the mode of operation through the service rate specification. If the rate is zero, the request is time-sensitive and should be fulfilled as soon as possible without delay and overhead. Otherwise, a reliable fixed rate service is requested.

The sensor can be identified by its unique ID. The sensor may deny access to that set of data (for example if the sensor requested does not exist or if it is malfunctioning) or it may decide there is not enough communication bandwidth to support the request, or the requested rate may not be supported by the sensor.

If the request is granted then the sensors will unicast at the requested frequency the sensor data to the external subscriber (SENS_DATA).

An alternative protocol, currently implemented in PCIL v1.0, has the entire sensor data broadcasted (unreliably) over the network. This approach avoids unnecessary duplications and has the advantage of cluster all the sensor data in a single packet, but has the disadvantage of potentially losing some data on the way (due to the unreliability of broadcast).

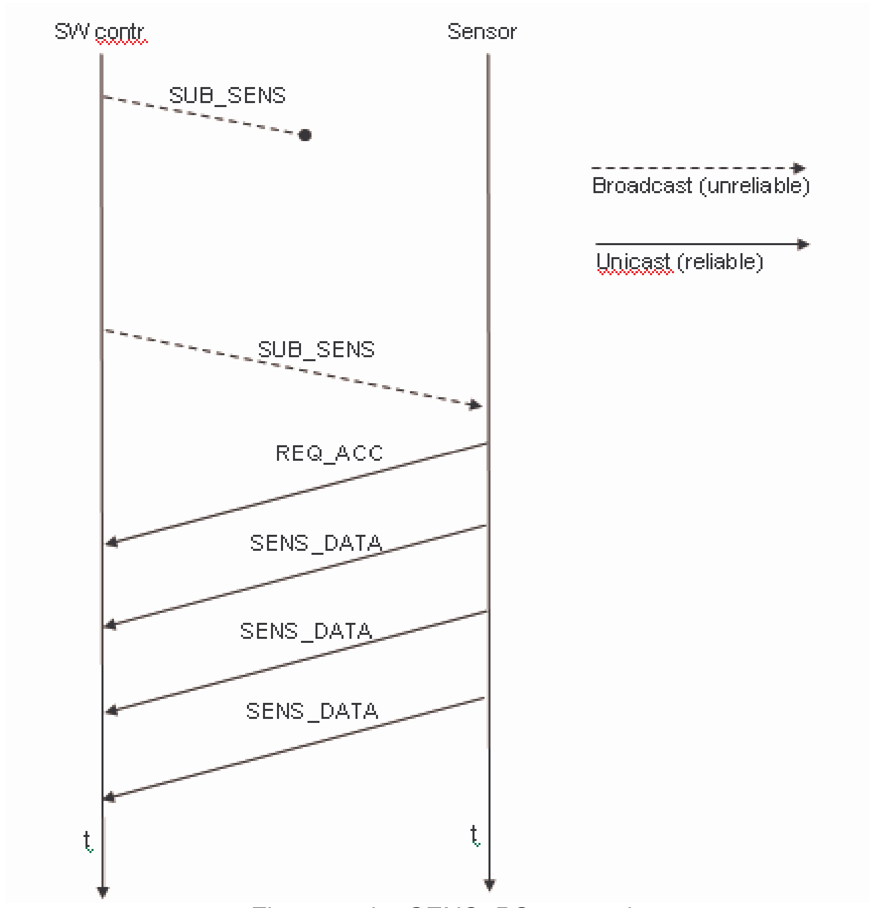


Figure 5: the SENS_PS protocol

II.d CHAL message format

This section describes the format of the messages used by the two protocols introduced in the previous section. They are exchanged between the components of the CHAL traffic control system.

II.d.1. Register External Control Message Format

This packet (REG_EXT_CON) is used by a software component to try to get permission to control an actuator. The format of the packet is given in table 1.

The first three fields are common to all the packets received and sent by CHAL, they identify the packet type and the sequence number (to avoid duplications and re-orderings) and as the length of the packet (that is not constant).

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to REGISTER_INTERSECTION (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes)
Actuator_type	4	Specifies the type of the actuator
Time resolution	4	Time resolution (in seconds)
Actuator ID	4	Unique CHAL actuator ID
DATA	[…]	Data specific to the controller type

Table 1: register external control message format

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to REGISTER_INTERSECTION (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes) to be 40 bytes.
Actuator_type	4	Specifies the type of the actuator, in this case GENERIC_TRAFFIC_CONTROLLER
Time resolution	4	Time resolution (in seconds)
Actuator ID	4	Unique intersection controller identifier
Main 1	4	Unique road segment identifier (major direction)
Main 2	4	Unique road segment identifier (major direction)
Minor 1	4	Unique road segment identifier (minor direction)
Minor 2	4	Unique road segment identifier (minor direction)

Table 2: register external control message format for traffic light controllers

The following three fields specify the type of actuator that we want to connect with, the actuator we want to control (through its unique CHAL ID) and the time resolution. Examples of actuator types are: Changeable Message Sign, Traffic Lights, etc. The Actuator type is not required to belong to a fixed set so that a new type of actuator can be easily added to CHAL.

The time resolution field is used to change the resolution in which the system control works (i.e. the rate at which control updates are generated).

Table 2 shows a register external control message format to be used for traffic lights controllers, in the case of a standard four leg intersection. The data part in this case is made by 4 fields that describe uniquely the intersection that will be externally controlled as well what is the major intersecting road and which one is the main direction on the two roads. If the intersection to be controlled has more or less legs there would be other fields at the end (one for each leg).

2.3.2. Unregister External Control Message Format

This packet is used by a software control component to inform an actuator that it will no longer be remotely controlled through CHAL. The format of the packet is given in table 3.

The actuator ID is used to identify which actuator the control is unregistering for.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to UNREGISTER_INTERSECTION (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes)
Actuator ID	4	Unique actuator ID

Table 3: unregister external control message format

2.3.3. Actuator State Change Message Format

This packet is used by the software control to inform an actuator that one of the externally controlled actuators needs to change state (e.g. a traffic light need to change phase). The format of the packet is given in table 4.

The actuator ID is used to identify the actuator which we want to change status of.

The last field is the one that describes the new state the actuator need to change to. For a changeable message sign this may be the text or image to be displayed on the screen. For a traffic light controller this is the phase that the controller should assume: the value can be a value between 1 and 24, each one identifying a phase of a standard NEMA 24 phases ring (Lee, Nevers and Robinson, 2004).

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to SET_STATE (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes)
Actuator ID	4	Unique CHAL actuator ID
DATA		Data specific to the actuator type

Table 4: actuator state change message format

2.3.4. Subscribe to sensor updates Message Format

This packet (SUB_SENS) is used by a software component to try to get permission to receive data for a particular sensor (or sensor set). The format of the packet is given in table 5a.

Field Name	Field size (bytes)	Field description
Packet number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet type	4	Set to SUB_SENS (to distinguish this packet from other packet types)
Packet size	4	Specifies the size of the DATA part of the packet (in bytes)
DATA	[…]	Three-tuple as the following:

Table 5a: subscribe to external sensor set

Sensor type	4	Specifies the type of the sensor
Time resolution	4	Time resolution (in seconds)
Sensor ID	4	Unique CHAL sensor ID
[…]	[…]	[…]

Table 5b: example of content of data part

The following DATA field contains at least one three-tuple (one for sensor). The three fields specify the type of sensor that we want to connect with, the sensor we want to get data from (through its unique CHAL ID) and the time resolution. Examples of sensor types are: Inductive loop, radar, etc. The sensor type is not required to belong to a fixed set so that a new type of sensor can be easily added to CHAL. The time resolution field was introduced to control the resolution in which the system should work (sensor updates).

2.3.2. Unsubscribe to sensor updates Message Format

This packet is used by a software control component to inform an actuator that it will no longer be remotely controlled through CHAL. The format of the packet is given in table 3.

The data part contains at least one sensor ID is used to identify which actuator the control is unsubscribing for.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to UNSUB_SENS (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes)
DATA	[…]	Unique sensor ID lists

Table 6: unregister external control message format

2.3.4. Sensor Data Message Format

This packet is used to communicate to the CHAL infrastructure the intersection sensor data. The packet summarizes the data gathered from ALL the sensors around an intersection. The format of the packet is given in table 7.

The CHAL protocol does not limit the type of data to belong to a fixed set. The data provided by a sensor is dependent on the sensor type. The data gathered from the sensor around an intersection, can be, for example: presence (1 if the detector has been triggered, 0 otherwise), count, gap (in seconds), speed (in km/hr), and flow (in vhc/hr). Some of the measurements may not be supported by a particular sensor. In that case the value is set to NOT_AVAILABLE.

The timestamp field is used to evaluate how fresh the data gathered from the sensors is.

The last field is used to communicate the status of a sensor.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to GEN_INT_SENSOR (to distinguish this packet from other packet types)
Packet_size	4	Specifies the size of the DATA part of the packet (in bytes)
DATA	[…]	

Table 7a: sensor data message format

Timestamp	4	In millisecond since initialization
Sensor ID	4	Unique CHAL sensor id
Sensor Type	4	Sensor type
[…]	[…]	Dependant on the sensor type

Table 7b: example of content of the data part

2.3.5. Stop Message Format

This packet is used only in the simulation environment. It is used by the Traffic simulator to communicate to the CHAL infrastructure that the simulation has been stopped. The control logic needs to be stopped. The format of the packet is given in table 8.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to STOP_PACKET (to distinguish this packet from other packet types)

Table 8: stop message format

2.3.6. Pause Message Format

This packet is used only in the simulation environment. It is used by the Traffic simulator to communicate to the CHAL infrastructure that the simulation has been paused. The control logic needs to be stopped. The format of the packet is given in table 9.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to PAUSE_PACKET (to distinguish this packet from other packet types)

Table 9: pause message format

2.3.7. Resume Message Format

This packet is used only in the simulation environment. It is used by the Traffic simulator to communicate to the CHAL infrastructure that the simulation has been restarted. The control logic needs to be restarted. The format of the packet is given in table 10.

Field Name	Field size (bytes)	Field description
Packet_number	4	This is a packet counter used to detect duplications and packet re-ordering.
Packet_type	4	Set to RESUME_PACKET (to distinguish this packet from other packet types)

Table 7: pause message format

III. CHAL library for Quadstone Paramics

In order to address the need of a Simulation environment for the development of traffic control systems, we have integrated CHAL with the microscopic traffic simulator Quadstone Paramics (Aitken, 2006). Paramics is a suite of microscopic simulation modules providing a powerful, integrated platform for modeling a complete range of real world traffic and transportation problems. It is fully scalable and designed to handle scenarios as wide-ranging as a single intersection, through to a congested freeway or the modeling of an entire city's traffic system. It is currently being used in over 40 countries world-wide and has in excess of 500 users including commercial consultants, cutting edge transportation researchers and state-funded Government agencies. Paramics has been deployed successfully in hundreds of networks worldwide and has been particularly useful in modeling large-scale networks in California, New York City and Sydney amongst others. Additionally, Paramics has been utilized on numerous ITS projects in Europe and Australia. It is currently used by the Partner for Advanced Transportation and Highways (PATH) in the Arterial Traffic Lab where it has been integrated with external 170s and 2070s controllers through the McCain Traffic Supply Controller Interface Device (CID) for Hardware In the Loop (HIL) simulation, as described in (Skabardonis et Al., 2007).

Given its wide-spread use Paramics has been chosen to be the first Simulation tool to be integrated in the TTCS tool suite. Other simulators can be integrated easily to the suite simply implementing the standard CHAL interface.

The current library, named Paramics CHAL Integration Library (PCIL), has been developed for the version 5.2 of Paramics. None of the features that has been added to the 5.* version have been used. As a result the library should be usable with previous version of Paramics. If you are experiencing problem using the integration library with old version of Paramics please contact the authors.

The PCIL installation instructions are given in section III.a

III.a Installation instructions

In order to use Paramics in the CHAL environment it is necessary to use the Paramics CHAL Integration Library (PCIL). It consists of a single small dynamic link library (dll) that can be downloaded from the Tools for the development of Traffic Control Systems website (TTCS, 2006).

First, create the Paramics model as usual (refer to [Aitken, 2005]) and save it. In Paramics a model is saved as a folder with multiple files inside. Assume for now that the folder is the following:

```
E:\My Code\My Paramics\CHAL_project\
```

First of all the PCIL library must be copied inside the model folder.

When this is done, navigate inside the folder and open with a text editor (e.g. notepad) the file named “programming.modeller”. The file may be either empty or list a series of libraries that are loaded before opening the project. Add PCIL to the list, complete with the path. In our case we would add the following line:

```
E:\My Code\My Paramics\CHAL_project\Paramics2CHAL.dll
```

This is all what is required to integrate a Paramics project with CHAL.

Now load the project with the Paramics Modeller. The modeler screen should look like figure 3. Read carefully the text appearing in the reporter section of the modeler screen. The first two lines should look like these:

```
Attempting to load plug-in
'E:\My Code\My Paramics\CHAL_project\Paramics2CHAL.dll'... OK

Loading: Paramics 2 CHAL Interface Library (by Marco Zennaro,
zennaro@path.berkeley.edu)
```

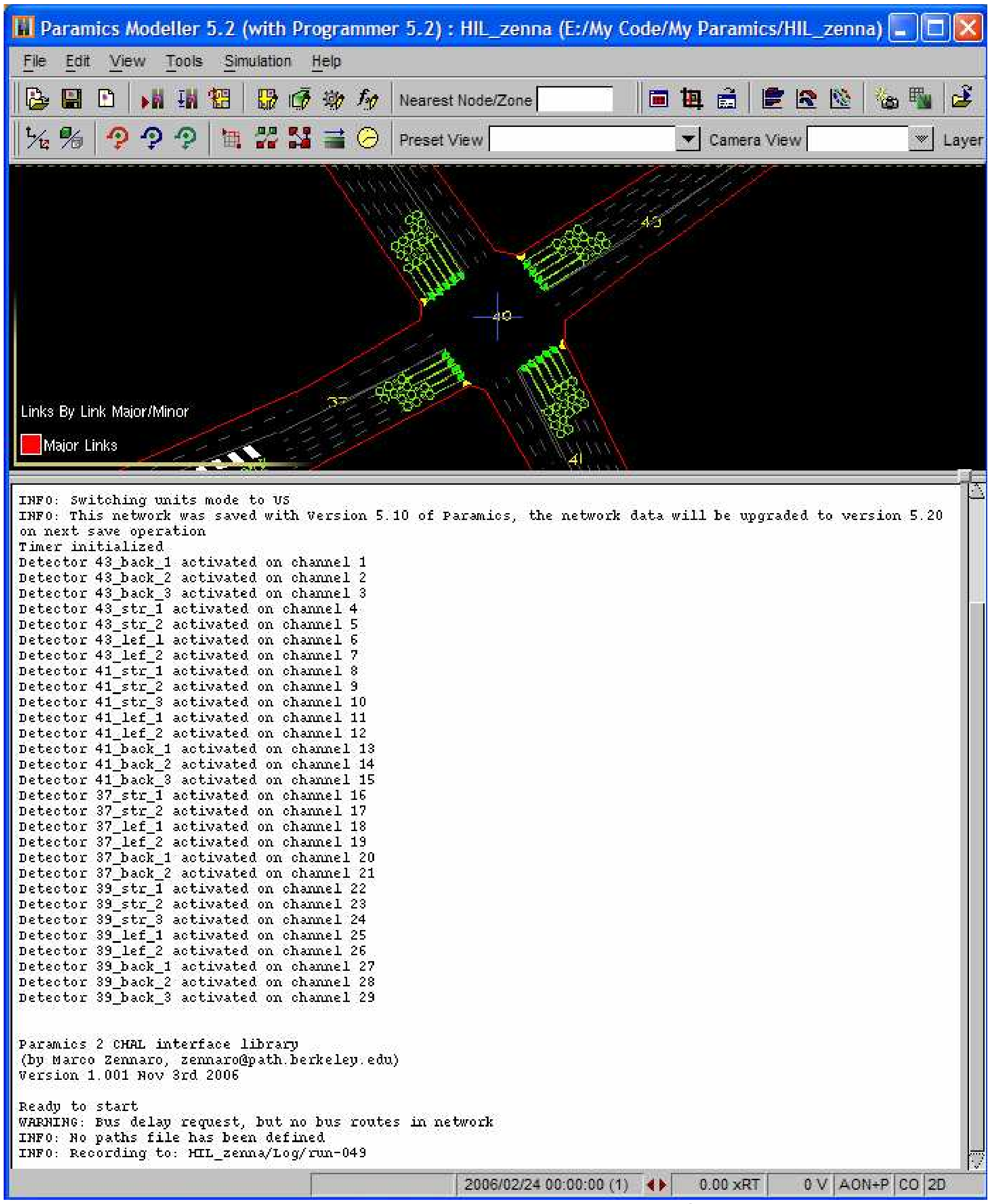
The first line shows that the CHAL integration library has been found. The second shows that it has been loaded.

Then what follows shows that the communication network is properly working and it is ready to connect with the control logic software. The network will be used by the library to communicate with the rest of the CHAL system.

```
Initializing WSA: success
Socket definition: success
Socket binding: success
Waiting for Control input
```

Then some internal timer is initialized and the Paramics detectors are connected to CHAL:

```
Timer initialized
Detector 43_back_1 activated on channel 1
[…]
```


Paramics Modeller 5.2 (with Programmer 5.2) : HIL_zenna (E:/My Code/My Paramics/HIL_zenna)
File Edit View Tools Simulation Help
Nearest Node/Zone
Preset View
Camera View
Layer
Links By Link Major/Minor
Major Links
2006/02/24 00:00:00 (1)
0.00 xRT
0 V
AON+P
CO
2D

```
INFO: Switching units mode to US
INFO: This network was saved with Version 5.10 of Paramics, the network data will be upgraded to version 5.20
on next save operation
Timer initialized
Detector 43_back_1 activated on channel 1
Detector 43_back_2 activated on channel 2
Detector 43_back_3 activated on channel 3
Detector 43_str_1 activated on channel 4
Detector 43_str_2 activated on channel 5
Detector 43_lef_1 activated on channel 6
Detector 43_lef_2 activated on channel 7
Detector 41_str_1 activated on channel 8
Detector 41_str_2 activated on channel 9
Detector 41_str_3 activated on channel 10
Detector 41_lef_1 activated on channel 11
Detector 41_lef_2 activated on channel 12
Detector 41_back_1 activated on channel 13
Detector 41_back_2 activated on channel 14
Detector 41_back_3 activated on channel 15
Detector 37_str_1 activated on channel 16
Detector 37_str_2 activated on channel 17
Detector 37_lef_1 activated on channel 18
Detector 37_lef_2 activated on channel 19
Detector 37_back_1 activated on channel 20
Detector 37_back_2 activated on channel 21
Detector 39_str_1 activated on channel 22
Detector 39_str_2 activated on channel 23
Detector 39_str_3 activated on channel 24
Detector 39_lef_1 activated on channel 25
Detector 39_lef_2 activated on channel 26
Detector 39_back_1 activated on channel 27
Detector 39_back_2 activated on channel 28
Detector 39_back_3 activated on channel 29


Paramics 2 CHAL interface library
(by Marco Zennaro, zennaro@path.berkeley.edu)
Version 1.001 Nov 3rd 2006

Ready to start
WARNING: Bus delay request, but no bus routes in network
INFO: No paths file has been defined
INFO: Recording to: HIL_zenna/Log/run-049
```

Figure 3: Paramics Modeler V5.2 after loading a CHAL project

At this point, if all the previous operations completed successfully, the library displays a message to inform that it is ready to proceed:

```
Paramics 2 CHAL interface library
(By Marco Zennaro, zennaro@path.berkeley.edu)
Version 1.001 Nov 3rd 2006
Ready to start
```

At this point, if the remote control is running, you can start the simulation. It is necessary to start at least a traffic controller logic software somewhere in the CHAL network before starting the Paramics simulation.

References

AITKEN, S., 2006. *Quadstone Paramics V5.2, Modeller User Guide.* [e-book]. Edimburgh. Available from: http://www.paramics-online.com

ANON. 2006. Flow Charted. *ITS International* [on-line] March/April issue. Available from: http://www.itsinternational.com

BULLOCK, D., URBANIK, T.. 2001. *Traffic Signal Systems: Addressing Diverse Technologies and Complex User Needs.* Millennium paper for A3A18, Committee on Traffic Signal Systems. National Academy of Science, Washington D.C.

GITELSON, J.. 1970. *Traffic Signal Computers*. San Francisco. California division of Highways.

HUNT, P.B., ROBERTSON, D.I., BRETHERTON, R.D., WINTON, R.I. 1981. *SCOOT, a traffic responsive method of co-ordinating signals*, TRRL Laboratory Report 1014.

KELL, J.H., FULLERTON, I.J.,1991. *Manual of Traffic Signal Design for Transportation Engineers*. Englewood Cliffs, NJ, Prentice Hall

LEE, A., NEVERS, B. ROBINSON, B., August 2004. *Signalized Intersections: Informational Guide FHWA-HRT-04-091*. [e-book]. Washington D.C.
Available from: http://www.tfhrc.gov/

LI, Y., ZHANG, W. 2006. *Summary of requirements: Los Angeles Transit Priority System*, UCBerkeley PATH report

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1989. *Standards Publication No. TS 1*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1992. *Standards Publication No. TS 2*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1996. *National Transportation Communications for ITS Protocol, overview. Standards Publication No. TS 3.1*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1996. *National Transportation Communications for ITS Protocol, simple transportation management framework. Standards Publication No. TS 3.2*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1996. *National Transportation Communications for ITS Protocol, class B profile. Standards Publication No. TS 3.3*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1996. *National Transportation Communications for ITS Protocol, global objects definitions. Standards Publication No. TS 3.4*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 1996. *National Transportation Communications for ITS Protocol, object definitions for actuated traffic signal controller units. Standards Publication No. TS 3.5*, Washington D.C.

NATIONAL ELECTRICAL MANUFACTURERS ASSOCIATIONS. 2002. *National Transportation Communications for ITS Protocol, the NTCIP guide, updated version 3.02b.* Washington D.C.

SKABARDONIS A., et Al., 2007. *Final Report: Development of Hardware-in-the-Loop (HiL) Simulation and Paramics / VS-PLUS Integration*, Caltrans Project Task Order 5311, California Partner for Advanced Transit and Highways Report.

SUN Microsystems Inc. 2001. *Jini technology Core Platform Specification*, version 1.2. Palo Alto, CA.

WEBSTER, F.V., 1958. *Traffic Signal Settings Road Research Paper Number 39*, HMSO, London, Scientific and industrial research

WOLKOMIR, R.. *A High-tech attack on traffic jams helps motorists go with the flow*. Sminsonian, Vol 17, No. 1, April 1986, pp42-51.

Tools for the Development of Traffic Control Systems. 2006. Paramics2CHAL integration library. [on-line]. (Updated 10 Nov 2006). Available from: http://ttcs.zennaro.net

Travel Times on Changeable Message Signs. 2006. [on-line] (Accessed 10 Nov 2006). Available from: http://www.calccit.org/projects/traveltime.html