CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

CARTESIUS and CTNET - Integration and Field Operational Test

Michael G. McNally, Craig R. Rindt

California PATH Research Report
UCB-ITS-PRR-2010-28

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation, and the United States Department of Transportation, Federal Highway Administration.

Final Report for Task Order 6324

May 2010

ISSN 1055-1425

CARTESIUS and CTNET - Integration and Field Operational Test

Final Report for PATH Task Order 6324

Michael G. McNally (mmcnally@uci.edu)
Craig R. Rindt (crindt@uci.edu)
Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697-3600

STATE OF CALIFORNIA DEPARTMENT OF TRANSPORTATION

TECHNICAL REPORT DOCUMENTATION PAGE

TR0003 (REV. 10/98)

1. REPORT NUMBER CA10-6324	2. GOVERNMENT ASSOCIATION NUMBER	3. RECIPIENT'S CATALOG NUMBER
4. TITLE AND SUBTITLE CARTESIUS and CTNET - Integration and Field Operational Test		5. REPORT DATE March 2010 6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) Michael G. McNally (mmcnally@uci.edu) Craig R. Rindt (crindt@uci.edu)		8. PERFORMING ORGANIZATION REPORT NO. UCB-ITS-PRR-2010-28
9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute of Transportation Studies University of California, Irvine Irvine, CA 92697-3600		10. WORK UNIT NUMBER 193 11. CONTRACT OR GRANT NUMBER 65A0208
12. SPONSORING AGENCY AND ADDRESS California Department of Transportation Division of Research and Innovation, MS-83 1227 O Street; Sacramento CA 95814		13. TYPE OF REPORT AND PERIOD COVERED Final Report June 2005- September 2009 14. SPONSORING AGENCY CODE
15. SUPPLEMENTAL NOTES None		

16. ABSTRACT

This report describes the conclusion of PATH Task Order 6324: *CARTESIUS and CTNET---Field Operational Test.* We describe the results of the multi-year project focused on integrating Caltrans primary signal management system, CTNET, with a major product from the Caltrans ATMS Testbed: the Coordinated Adaptive Real-Time Expert System for Incident management in Urban Systems, or more simply, *CARTESIUS*. The major products of this research include numerous software products for integrating CTNET with field devices, simulation software, with other traffic management systems in general, and with a streamlined re-implementation of the CARTESIUS incident management system. The report details the development of the various software components necessary for external systems with CTNET using both the AB3418e protocol and Tent's own custom socket-based communications protocol for communications between CTNET clients and the CTNET Commerce. The use of these software components to link CTNET to various systems is described, including a non-standard field infrastructure, the Paramics microsimulation, and the CARTESIUS incident management system. The resulting system is used to evaluate a more deployable re-implementation of CARTESIUS connected to the simulation via CTNET. The results of the evaluation demonstrate that the reimplementation produces performance similar to the original system for a restricted evaluation subset. Further work is necessary to lead to complete deployment, particularly defining requirements that are compatible with existing TMC processes. Nonetheless, the work described in this project represents a step toward a deployable next generation architecture for multi-jurisdictional incident management using existing Caltrans assets.

17. KEY WORDS Cartesius, CTNET, traffic control, incident management	18. DISTRIBUTION STATEMENT No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161	
19. SECURITY CLASSIFICATION (of this report) Unclassified	20. NUMBER OF PAGES 90	21. PRICE N/A

Reproduction of completed page authorized

Abstract

This report describes the conclusion of PATH Task Order 6324: *CARTESIUS and CTNET---Field Operational Test.* We describe the results of the multi-year project focused on integrating Caltrans primary signal management system, CTNET, with a major product from the Caltrans ATMS Testbed: the Coordinated Adaptive Real-Time Expert System for Incident management in Urban Systems, or more simply, *CARTESIUS* . The major products of this research include numerous software products for integrating CTNET with field devices, simulation software, with other traffic management systems in general, and with a streamlined re-implementation of the CARTESIUS incident management system. The report details the development of the various software components necessary for external systems with CTNET using both the AB3418e protocol and CTNET's own custom socket-based communications protocol for communications between CTNET clients and the CTNET CommServer. The use of these software components to link CTNET to various systems is described, including a non-standard field infrastructure, the Paramics microsimulation, and the CARTESIUS incident management system. The resulting system is used to evaluate a more deployable re-implementation of CARTESIUS connected to the simulation via CTNET. The results of the evaluation demonstrate that the reimplementation produces performance similar to the original system for a restricted evaluation subset. Further work is necessary to lead to complete deployment, particularly defining requirements that are compatible with existing TMC processes. Nonetheless, the work described in this project represents a step toward a deployable next generation architecture for multi-jurisdictional incident management using existing Caltrans assets.

Keywords: CARTESIUS, CTNET, traffic control, incident management

Executive Summary

In 2005, Caltrans released an RFP seeking to integrate two of its assets---one functional and in wide deployment, and one moving through its research program as a possible solution for corridor management. In CTNET, Caltrans has a functional traffic signal management subsystem that can connect engineers' desktop computers to operational traffic signal controllers using an interface that mimics that offered at the cabinet. This capability for remote control of intersections simplifies the engineer's tasks and offers an excellent means for tracking the status of Caltrans traffic signal controllers. In funding the development of CARTESIUS via the ATMS Testbed and PATH programs, Caltrans has previously developed advanced concepts for dealing with the complex issues of multi-jurisdictional incident management using a set of software agents that use distributed problem solving approaches for sharing local information and action plans across jurisdictional boundaries to produce globally superior response plans. The RFP proposed integrating the two systems and running a field-operational test of the combined system to establish its performance and determine its feasibility for broader deployment.

Task Order 6324 is the third of three research projects focused on this effort (the prior two being TO-5313 and TO-5324). The results of the series of projects have been mixed. On the positive side we established a clear understanding of the relative roles of CTNET and CARTESIUS. CTNET is a robust traffic signal management system with a well-defined functional role that fits into the broader National ITS Architecture. It is a standalone system with well defined interfaces that use (and indeed pioneer) accepted standards for center to field element communications. Its user interface, while somewhat dated, still offers useful functionality to engineers charged with managing remote traffic signals. CARTESIUS' functional role is more varied, but its primary contributions can be distilled into (a) a unique approach for representing transportation system disruptions abstractly that (b) can be shared across jurisdictional boundaries and (c) used into local optimization algorithms that (d) produce collections of control actions whose impacts can (e) also be shared across jurisdictions and (f) used to find local responses that are compatible with global constraints.

The project has led to the successful development of two custom libraries (Jab3418e and JCTNET) that allow external programs to connect to CTNET and act as authenticated clients in order to obtain system state information and to send new timing plans to signals in the field. These libraries were used to develop a simulation platform that can simulate CTNET Compatible field controllers interacting with the Paramics microsimulation. More generally, the libraries open CTNET for further integration in any traffic management system that needs read/write control to traffic signals. These libraries were used to connect CTNET and a simplified re-implementation of the CARTESIUS system that focuses on generating control response recommendations that can eventually be integrated into Caltrans TMC processes. The CARTESIUS re-implementation satisfies a subset of the requirements defined in the final report for PATH TO 5324. In particular, the multi-agent features were deferred for later development due to challenges related to expanding the original algorithm beyond two agents. Additionally, the development of a graphical user interface was deferred because it became clear in this and in related research that interfaces for TMC operations must be highly specialized to fit into prevailing TMC processes. Instead, the system was designed to provide analysis and recommendations using an Enterprise Service Bus platform to implement a Service Oriented Architecture that could be integrated into existing TMC interfaces as they evolve. A simulation-based evaluation of the combined system shows that with proper pre-specification and tuning, the CARTESIUS system can identify response strategies that improve performance versus the status-quo.

The research also identified a number of areas that require further research or changes in strategy. The project fell short of its original goal to carry out a complete field operational test of the combined CARTESIUS/CTNET system. The reasons for this were both technical and institutional. The report includes recommendations for avoiding similar problems in the future that center on the use of better Systems Engineering practices. We also found that some of the components necessary to deploy a fully functional CARTESIUS are still missing. One notable missing feature is the unavailability of an effective near short-range demand estimator that can feed into CARTESIUS' response algorithms. There are promising tools still under development by Caltrans that may meet this need. Another critical issue is the development of a general purpose approach to translating potential response actions into estimated impacts on the system. Finally, it has been clear in this project and on other projects related to the Testbed, that Caltrans TMC processes have evolved over time to achieve certain levels of efficiency with available human and technical resources. Altering these processes with new software and analytical tools is not a trivial task. Development of clear processes for achieving such transformations could help speed the integration of new technologies into Caltrans operations.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This document reports on the research and development carried out under PATH Task Order 6324: CARTESIUS and CTNET—Integration and Field Operational Test. The goal of this multiyear project has been to move the research-based CARTESIUS incident management system closer to deployment by integrating it with Caltrans' CTNET arterial traffic management system. This project is the third of three task orders funded for this purpose.

1.1 A Brief History of CARTESIUS and CTNET

CARTESIUS—the Coordinated Adaptive Real-Time Expert System for Incident management in Urban Systems—arose from the California Advanced Traffic Management System (ATMS) Testbed program at the University of California, Irvine (UCI) during the 1990s that developed a unique proving ground for traffic management technology. The Testbed links a real-world, multi-jurisdictional transportation system with research laboratories at UCI to facilitate the testing and evaluation of cutting edge Intelligent Transportation Systems (ITS) technologies.

CARTESIUS is arguably one of the most important products of the first generation of the Testbed. It was developed with the realization that traffic management in urban corridors is complicated by jurisdictional as well as operational problems. Traffic in a freeway/arterial corridor requires coordinated management that optimizes corridor traffic while preserving the various levels of authority, data control, and decision-making power inherent in a multi-jurisdictional environment. CARTESIUS approaches this problem by employing advanced cooperation and conflict resolution methodologies for coordinated traffic management operations among multiple jurisdictions using a multi-agent architecture to represent the diverse interests and needs of each agency. On the whole, CARTESIUS refers to both a *philosophy* regarding how to manage traffic under the constraints inherent to multi-jurisdictional transportation systems using a Distributed

Problem Solving (DPS) approach and an *implementation* of that philosophy in a working two-agent prototype that demonstrates the efficacy of the philosophy.

CTNET is a distributed software system for integrated management of traffic signals that allows operators to remotely manage, view, and log real-time traffic signal field data. The system uses a client/server architecture in which a CTNET CommServer acts as a proxy server for a traffic master in the field and its associated traffic signals. CTNET clients connect to the CommServer to gain access to monitoring and management functions provided by the CommServer. The clients provide a convenient user interface to the user using TIGER-line maps to display geographically accurate maps of the managed traffic signals.

The natural synergy between these two systems led to the funding of this research and development effort, which was carried out under three PATH task orders: 5313, 5324, and 6324. We briefly discuss each in the sections below.

1.1.1 TO-5313

The original purpose of this TO 5313 was to test CARTESIUS, the implementation, as a precursor to developing a deployable version of CARTESIUS for general use in California. The research plan originally identified three broad issues for this test:

- Architectural issues related to moving the software from the relatively homogeneous laboratory setting at UCI to a heterogeneous field environment and usability issues associated with the use of CARTESIUS by Transportation Management Center (TMC) operators in practical, mission-critical settings.
- Verification and validation of embedded knowledge and agency policy regarding control and interagency-coordination strategies.
- Protocols for testing developing technology in the field to resolve legal and institutional barriers to field testing a comprehensive traffic management system

To address these issues, the research plan called for a test of CARTESIUS in two evaluation modes. In the first mode, the system was to process real-time data coming from sensors in the field and was to provide advisory management strategies and control actions for the consideration of Caltrans and local traffic management center personnel. This mode of evaluation was to provide on-line, risk-free verification and limited validation of CARTESIUS.

The second evaluation mode was to involve usability and stress testing of the CARTESIUS system with the CARTESIUS agents remotely connected to the Paramics traffic simulator at the UCI laboratories. In this mode, TMC operators/personnel were to be asked to respond to the scenarios using CARTESIUS to implement control strategies in Paramics.

During the course of TO-5313, numerous problems arose related to the deployment of CARTESIUS even in the limited on-line roles envisioned for the tests. As a result, the scope of the project was revised several times to accommodate these problems. At the same time, work on PATH Task Order 5324/6324—intended to explore integrating CARTESIUS with Caltrans CTNET traffic signal management system—provided additional incentive to revise the functional role of CARTESIUS to be more compatible with existing technology and standards. Coupled with the identified difficulties with the existing prototype, the management team of both this research effort and the CTNET effort collaborated to develop a revised plan for TO 5313 with a focus on enhancing the deployability of CARTESIUS with a first effort being the CARTESIUS/CTNET integration effort in TO 5324/6324.

The results of TO-5313 was that the follow on task orders 5324/6324 should carry out the following steps for the continued development of CARTESIUS to meet the needs of Caltrans and other stakeholders.

- Development of a set of functional requirements for the new CARTESIUS software implementation that meets the needs of all stakeholders and, in particular, seeks to put the CARTESIUS project on a path toward eventual deployment in California’s TMCs.
- Development of a software design that efficiently satisfies these requirements
- Implementation of that software design for the Testbed sub-network involving Caltrans–District 12 and the City of Irvine.

1.1.2 TO-5324

Following the recommendations of TO-5313, the scope of task order 5324 was revised to focus on the development of a complete set of requirements for the implementation of a new version of

CARTESIUS to overcome problems of the original prototype and to improve its compatibility for integration with CTNET. Similarly, task order 6324 was restructured to implement a new version of CARTESIUS according to these requirements, integrate it with CTNET, and evaluate the integrated system's operation.

1.2 TO-6324: Revised Project Scope

As a result of the above modifications, the revised work plan for TO-6324 differed significantly from the original work plan—its approach to evaluating CARTESIUS/CTNET in particular. This evaluation focuses primarily on the use of simulated data from the 405 corridor (405 freeway and parallel Alton Parkway arterial) to evaluate the performance of the coupled system. The evaluation framework, however, is structured to use CTNET's standard communications interfaces for communicating with simulated field devices. The evaluation procedures used are similar to those described in the original work plan, with the main modifications being the data sources used for the evaluation and the degree to which operating agencies will be involved in the evaluation.

Also, to help ensure that the product of the research would be useful to the customers, the research and development has been carried out with the intent of eventually integrating the adaptive signal control module being developed in PATH TO-6323 as a control action provider for CARTESIUS. This approach would ultimately enable the convergence of work being done on PATH TO-6324 and that on PATH TO-6323. The approach would also allow the integration of the adaptive control model to be incorporated as a module in the CARTESIUS implementation in CTNET.

1.2.1 CARTESIUS Response Formation and Strategy Translation

CARTESIUS response formation is the product of a heuristic search through the problem space defined by estimated traffic conditions and estimated incident impacts in terms of demand and capacity disruptions. The search is driven by an agency-specific set of response strategies and related control actions that can be used to implement those strategies. These agency-specific configurations comprise a knowledge base that constrains the possible solution space to solutions that are consistent with local traffic control policies. Any evaluation of the CARTESIUS/CTNET system will be governed by this knowledge base and the particular set of possible strategies and control actions available in the knowledge base. For the remaining research, two types of control actions will be added to the knowledge base. The first will use predetermined timing patterns accessible from the controllers using CTNET to provide coordinated signalization strategies to the search algorithm. This will permit CARTESIUS to interoperate effectively with typical installations currently in use. We will also consider adding a control action to the knowledge base that uses the adaptive control system being developed in PATH TO-6323. This control subsystem solves the corridor ramp-metering and traffic signal control problem using a multi-objective formulation that produces solutions for optimal control that specify the set of non-dominated corridor control options. CARTESIUS can explore this set using its search algorithm to find the solution that is acceptable to all participating agencies.

1.2.2 Path to Deployment

The integration of CARTESIUS and CTNET offers a significant step toward deploying true corridor control of non-recurrent congestion. The use of existing timing patterns provided by CTNET improves the deployability of CARTESIUS by using control actions that have already been vetted by traffic engineers rather than trying to dynamically determine new strategies. Consequently, a completion of the final year of this project will move CARTESIUS significantly closer to potential real-world deployment.

Toward this ultimate end, the CARTESIUS/CTNET evaluation is being conducted on the I-405 corridor network in the City of Irvine (COI). This location has been chosen because we have at least limited authority to conduct tests involving closed-loop control. On the arterial, we have installed a system of Type 2070 controllers at all signalized intersections that operate independently from the local COI system. Research already completed as part of this project has connected these controllers to CTNET; a secondary system based on state-of-the-art Siemens ACTRA Central Traffic Control System with custom-designed Input Acquisition Software is in place as a backup, should the CTNET configuration prove problematic. Software has been developed, and laboratory tested, that permits real-time adaptive control of Caltrans District 12 ramp meters in the study area (a feature not currently possible under Caltrans District 12 ATMS). We have established real-time communication with these control devices and also receive real-time raw data streams from loop detectors within the study area. We have memoranda of understanding with both the COI and Caltrans District 12 that permit the research team to conduct closed-loop control experiments in the study area.

These existing efforts will ultimately allow us to connect the CARTESIUS/CTNET system to real-world data streams in order to evaluate the strategies recommended by CARTESIUS. Toward this end, the CARTESIUS/CTNET system will be configured to interoperate with the I-405 corridor network, including all available traffic signals, ramp meters, and information systems. We will evaluate this deployment configuration using our Paramics simulation of the corridor using custom plugins already written as part of this research to allow the simulation to interoperate with CTNET, and ultimately CARTESIUS. The system will also be connected in a read-only manner to the real-world data feeds described above to evaluate its recommendations as compared with actual operations. In the final analysis, we expect to assess the degree to which constraining the control actions available to CARTESIUS (e.g, by using only pre-vetted signal timing patterns stored in CTNET) impacts the quality of the system's response to incidents.

1.3 CARTESIUS and the modern TMC

During the course of this research, and the research in related projects on the Testbed, the research team has had the opportunity to evaluate the core processes of the modern Caltrans TMC at District 12. The operations in the TMC are dominated by engineers and staff whose primary roles are to

- identify disruptions in the system,
- communicate to relevant parties about that disruption
- dispatch traffic management and maintenance teams to the site,
- disseminate incident details to the traveling public
- monitor and react to changing conditions on the ground as appropriate

To carry out these functions, the operators rely on well defined processes centering around particular software tools:

- The ATMS system for monitoring traffic conditions and controlling changeable message signs (CMS)

- The CHP CAD system for monitoring emergency response.
- The district's radio communications system for dispatching teams.
- The TMC activity logging system for recording all actions taken by the TMC and qualitative assessments of the performance of the system.

Additional tools are available, such as an event management system whose role is to provide specific information to the regional 511 information system as well as various tools for controlling CCTV, highway advisory radio (HAR), and a multitude of other systems. At the same time, Caltrans statewide continues to develop the ATMS system, new activity logging tools (TMCAL), and its surface street operator interface (CTNET) in parallel to the efforts in local districts.

Put together, this collection of software tools and interfaces make defining a realistic use-case for a new incident management system that fits into existing processes challenging at best, and ultimately beyond the realistic scope of this project. As such, efforts in this research focused less on creating new interfaces and shifted instead toward processing available data into high-level system assessments and response suggestions that can more feasibly be integrated into TMC processes by eventually incorporating them into existing and new interfaces already under development. Not only does this approach bring the promise of CARTESIUS closer to actual use by Caltrans, it also is consistent with the new philosophy of the Caltrans ATMS Testbed of which CARTESIUS has always been a central component.

1.4 Overview of this Document

Given the above backdrop, we continue in this report by discussing the features of both CTNET (Chapter 2) and CARTESIUS (Chapter 3). We then discuss the problems and solutions to integrating these components into a functioning system in Chapter 4. In Chapter 5, we turn to the design and re-implementation of CARTESIUS as a background traffic management service instead of a top-level traffic management application that would compete for space in the ever-congested set of TMC processes. Chapter 6 describes the development of the required communications libraries to allow CARTESIUS and other Java-based programs to act as clients to the CTNET system, thus obtaining read/write capabilities to any traffic signal subsystem under

CTNET management. In Chapter 7, we describe the evaluation strategy for the combined system. This includes a discussion of how the strategy was gradually downgraded from a full-blown closed-loop field operational test to a more limited evaluation using simulation. We also describe the development of customized simulation platform that was used as well as its features and limitations. Chapter 8 discusses the evaluation results for integrated system on the Testbed study network. Finally, Chapter 9 offers some conclusions and recommendations for CARTESIUS, CTNET, the ATMS Testbed, and future corridor field operational tests.

Chapter 2
CTNET

2.1 Overview

CTNET is a distributed software system for integrated management of traffic signals that is widely deployed by Caltrans and some municipalities across the state. CTNET allows operators to remotely manage, view, and log real-time traffic signal field data. The system uses a modular client/server architecture in which a CTNET CommServer acts as a proxy server for one or more Traffic Responsive Field Masters (TRFMs) in the field (see Figure 2.1). The masters, in turn, control a subset of traffic signals, which can be managed by either Caltrans C8, version 4 software (type 170 controllers) or TSCP version 1.02 software (type 2070 controllers). The TRFMs support both synchronized traffic responsive and time of day traffic signal coordination schemes.

CTNET clients connect to the CommServer to gain access to monitoring and management functions provided by the CommServer. Access privileges are controlled on a per-user basis, allowing the system to limit specific functions to authorized users only. The existing CTNET client provides a user interface which uses TIGER-line data to display geographically accurate maps of the managed traffic signals. An open (non-proprietary) communications protocol is used between CTNET clients and the CTNET server so that third party clients can be developed to work with the system.

By itself, CTNET does not provide automated, globally coordinated incident response. The existing CTNET client software allows TMC operators to manually alter timing plans to implement control responses determined outside the CTNET architecture. These responses may come from any source, but are most likely to be derived on the fly by TMC operators with expert knowledge of the system. Thus, CTNET could benefit from the integration of the global incident management capabilities that are fundamental to CARTESIUS.

2.2 Functional Role

CTNET is a solution to the Surface Street Control market package (ATMS03) as defined by the National ITS Architecture (Iteris, Inc, 2002). This market package provides the central control and monitoring equipment, communication links, and the signal control equipment that support local surface street control and/or arterial traffic management. A range of traffic signal control systems are represented by this market package ranging from fixed-schedule control systems to fully traffic responsive systems that dynamically adjust control plans and strategies based on current traffic conditions and priority requests. This market package is generally an intra-jurisdictional package that does not rely on real-time communications between separate control systems to achieve area-wide traffic signal coordination. Systems that achieve coordination across jurisdictions by using a common time base or other strategies that do not require real time coordination would be represented by this package. This market package is consistent with typical urban traffic signal control systems.

This market package consists of 6 equipment packages

- Roadway Basic Surveillance: This equipment package connects the TMC to fixed traffic sensors. CTNET satisfies the functional requirements for this equipment package for loop detectors that are part of the managed arterials.
- Roadway Equipment Coordination: This package provides for direct communications between field controllers to facilitate coordination without requiring direct intervention by the center. CTNET provides this functionality through its Field Master Program.
- Roadway Signal Controls: This package provides the fundamental equipment for signal control. By itself, CTNET only provides a subset of the functionality defined for this equipment package: connectivity between field controllers and the TMC. It is, however, a critical part of a standard Caltrans signal deployment that does satisfy the functional requirements for this equipment package.
- Collect Traffic Surveillance: This package provides the TMCs with remote monitoring and control capabilities for traffic sensors, making data accessible to TMC processes, and to other TMCs. This equipment package describes CTNET’s core functionality. It provides these capabilities, including inter-TMC access, using its client/server

architecture. Furthermore, CTNET provides a dynamic map interface for monitoring traffic performance and device status as well as for updating signal control parameters.

- TMC Signal Control: This package provides the TMCs with remote management capabilities of signalized intersections. As above, CTNET’s client/server architecture provides these capabilities.
- Traffic Maintenance: This package monitors the status of field devices and notifies operators of faults. CTNET provides these capabilities as an integrated part of the client interface.

2.3 Use Cases

CTNET is a traffic signal management system that allows operators to remotely control arterial traffic detection and control devices as if they were accessing them directly at cabinet in the field.

The system is designed to greatly simplify the maintenance and operation of a large set of traffic signals by allowing one or more operators to remotely monitor signal status and operation, log traffic detection and control data, and modify traffic signal timing plans. A set of typical use cases are described below for an operator working with a CTNET client. These cases all assume that a CTNET data file has already been created for a managed arterial, and that a CTNET CommServer processes is deployed and operational.

Basic flow- system monitoring: An operator in the TMC is tasked with ensuring the system is operating normally. She opens the datafile for the arterial network of interest using the CTNET client application. She clicks on the connect button to login to the CTNET CommServer managing the arterial network. She then selects the pre-saved view of the network that shows the status of all the managed intersections on a map, including current phasing, vehicle calls, pedestrian calls, and cabinet alarms. The operator uses the mouse to move over system detector icons to view flow, occupancy, and speed data for particular sections that she knows to be congested during this time of day. She confirms them to be within acceptable ranges according to her experience.

The operator then clicks the display alarms toolbar button to view the preemption and alarm logs. She notes several emergency vehicle preemptions have occurred, but that their durations were within normal operating parameters.

Having finished her monitoring task, she adds a note to the operations center log that everything appears normal and continues with her other tasks until her next scheduled check.

Alternative flow- device failures: After opening the saved view of the system, the operator notes a number of problems. First, the loss of signal (LOS) icon is displayed for the cabinet associated with one of the intersections. She makes a note, knowing that this indicates a communications failure between the master controller and the intersection’s local controller.

Second, after musing over what is usually a particularly busy intersection, she notices that the reported volume is zero for the system detector on the busiest leg. Surmising that this is likely due to a detector fault, she checks a previously configured detector report within CTNET to see if the detector is bad. She makes a second note for maintenance after confirming it is.

Finally, when viewing the alarm logs, she notices a cabinet flash alarm on a minor intersection has occurred in the past hour. Checking the maintenance and police logs, she finds no mention of an incident at the intersection and adds a last note to explore the reason for the intersection being going to flash mode.

Having finished her preliminary analysis, the operator phones the maintenance department to dispatch a technician to troubleshoot the connectivity problem and system detector problems she noted. She then contacts other operations personnel to identify the cause of the flash alarm, which she finds was due to a maintenance operation that had not been properly logged in the ATMS system. She adds the log and goes about her duties.

Alternative flow- uncharacteristic flow: While reviewing system detector flows, the operator notes unusually high occupancy and low volume at a major intersection in the network. She uses the Tic’s CCTV system to view what is happening at the intersection and discovers that a collision has occurred in the middle of the intersection, causing a severe congestion on all approaches. She consults incident logs and, finding no mention of the accident, calls emergency operations to report the accident and initiate an appropriate emergency response.

Because the TMC is a participant in such emergency responses, she begins to follow her agency's standard operating procedures for managing traffic in this situation.

Alternative flow- updating timing plans: Engineers at the operations center have produced updated planning model using a new regional model and recent counts obtained using the CTNET reporting functions. The new model includes significant changes in daily traffic demands along two major arterials managed by CTNET. After performing network-wide re-timing and coordination using standard agency engineering practice, a TMC operator is tasked with updating the timing plans in the field.

He begins, as above, by opening data file for the network of interest using the CTNET client and clicking on the connect button to login to the associated CTNET CommServer. Referring the updated timing plans produced by the engineers, he right clicks on the cabinet of the first intersection he needs to update to get the cabinet menu and selects the "get timing" item, which is active because the CommServer has determined that the operator has permission to view and update timing data. This opens up the time data sheet with the timing data uploaded from the field master.

The operator proceeds to enter in the first new timing plan. Most parameters are the same, but the maximum extensions have changed to reflect the changed demands in the system. Clicking the coordination tab, he also updates coordination parameters that have changed due to shifting prevailing flows predicted for the system. After finishing this task, he clicks the upload button, sending the new timing data to the controller.

He performs these tasks for each intersection with an updated timing plan, then disconnects his CTNET client from the CommServer and closes the application.

2.4 CTNET Communications

As we'll discuss in Chapter 6, the project team made a design decision in the CTNET/CARTESIUS integration to not alter CTNET and instead to have CARTESIUS act as a CTNET client to interact with the traffic signal subsystem. Understanding the underlying communications protocols used by CTNET was therefore a central task. We summarize those protocols here.

The client/server architecture of CTNET is based upon two distinct sets of communications links as shown in Figure 2.1. The first set (CI-1) connects the CTNET CommServer to the field masters, which in turn forward messages to and from local controllers as appropriate. Communications at this level use an extended version of California's AB3418 protocol (Caltrans, 1995) that was specifically developed to allow central-to-master communication. The AB3418 extended (AB3418E) protocol adds messages for obtaining current 8 phase operation, presence, system detector measurements, as well as the ability to get, set, and manage controller time data. The AB3418 specification is defined for the complete network stack, detailing the physical data-link, network, and application layer requirements. The lower layers (physical, data-link, and network) are generally structured to support serial line communications (RS-232) between the CommServer and the field master---typical of connections provided over telephone lines or similar. However, it is notable that CTNET supports addressing using the TCP/IP stack that is the backbone of the broader Internet. It should be noted that the TCP/IP connections break some of the assumptions in the AB3418 protocol about the ability of the stack to convey real-time Application Layer messages. As such, use of the TCP/IP link for real-world operations may need further vetting. Nonetheless, the undocumented TCP/IP support made it possible to link CTNET to Paramics without any modifications to the former. Since this is only used for simulated analysis, the real-time implications are unimportant (this is discussed in greater detail in section 7.2).

The Application layer specification is the most relevant aspect of the protocol for the purposes of this research as it defines the structure of the message packets that are used to transmit data between the controllers and the CommServer. Very simply, the specification defines how application message frames must be structured to pass data between the server and the field controllers. The first byte of the message defines the message type using standards from the National Transportation Communications for ITS Protocol. The following bytes contain the appropriate payload for the defined message type. AB3418 protocol defined only a few basic message types for interacting with field controllers. AB3418e expanded this to support the more advanced get/set features need to meet the needs of CTNET, which in turn, uses AB3418e as its core protocol for communicating the status of the system to CTNET clients.

The second set of communications links (CI-2, CI-3, …, CI-n) connect CTNET desktop clients to the CTNET server and, by proxy, to the field controllers. Two protocols are used in this infrastructure. First, the CTNET client/server application is built atop a set of messages passed between the Client and the CommServer using simple socket-based communications over TCP/IP. These messages are used to for user authentication, CommServer configuration, and to send commands to the CommServer to request and set field element parameters from the server. (Note that once a client is connected to the CommServer, it can also send messages directly to field elements using the AB3418e protocol.) The CTNET application messages are passed as fixed-length packets in which the first byte identifies the message type and the remaining bytes store to a tab-delimited string of parameters for the message. In addition to the CTNET application messages, CTNET clients also receive AB3418e packets via UDP from the CommServer once they have connected. These packets are generally status messages forwarded by the CommServer from the field elements so that the Client can display the system status on the user interface.

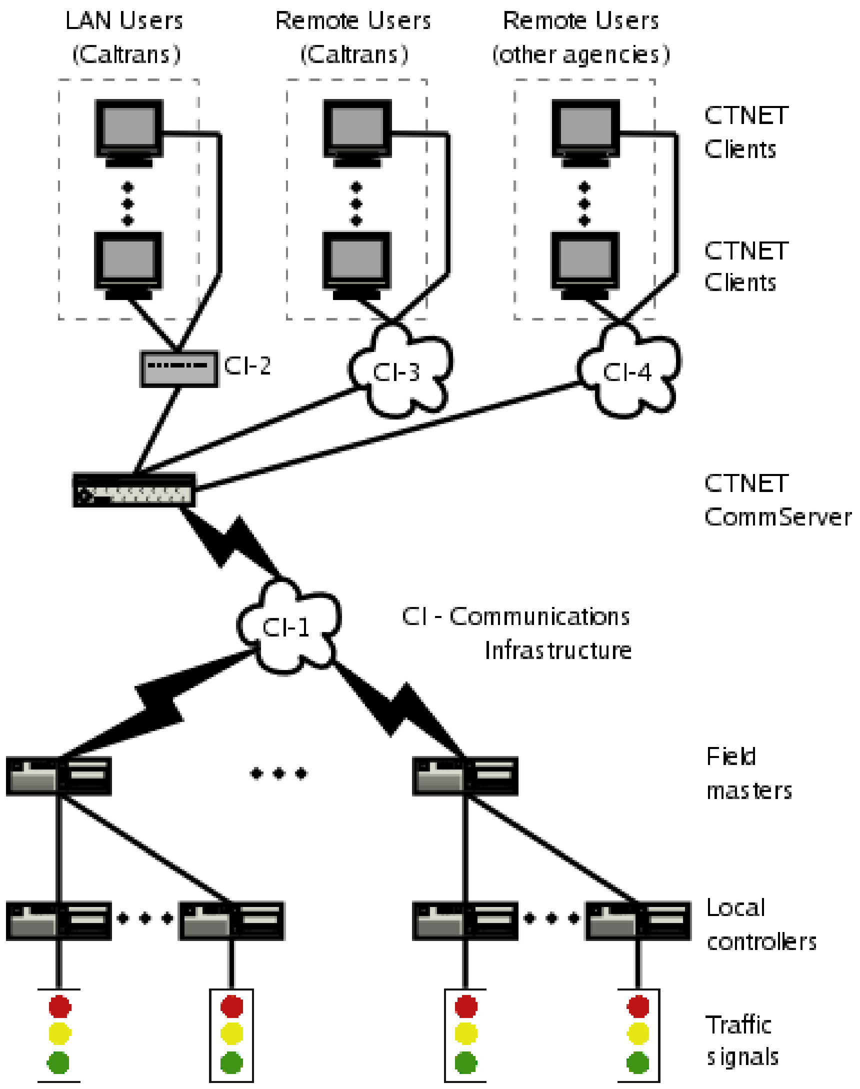

Figure 2.1 The CTNET communications infrastructure.

Chapter 3

CARTESIUS

3.1 CARTESIUS as a theoretical system

The core goal of the original CARTESIUS research was to develop a new Distributed Problem Solving (DPS) approach for the provision of real-time decision-support to TMC personnel in multi-jurisdictional, network-wide management of non-recurring congestion. The approach is based on the Functionally Accurate/Cooperative (FA/C) approach to DPS (Lesser and Corkill, 1981).

3.2 Traffic Management through Distributed Problem Solving

The core features of the CARTESIUS philosophy permit conforming distributed systems to:

- reflect the jurisdictional distribution of traffic problem-solving expertise,
- provide local management of data sources, and
- reduce synchronization delay and communication overhead.

The main goals of the architecture are to allow:

> "a set of agents to interact in order to produce an efficient and conflict-free global solution in an environment that is inherently distributed. Their interaction is constrained by the lack of complete information that results from the data and control knowledge being distributed according to geographical and administrative criteria. Input data available to the TMC operator, such as detector data describing traffic conditions, visual data coming from CCTV cameras installed at key locations, and incident reports or confirmations, are normally partitioned according to the institutional organization of the management agencies, which must administer the information in their possession as

efficiently as possible, avoiding the transmission of irrelevant or ill-timed data. Another constraining factor is the need of each agency to preserve dedicated control over its jurisdiction, maintaining its decisional power, and applying local criteria and guidelines. At the same time, the agencies, as interacting components of a regional, integrated management organization, must attempt to converge towards a common goal, that of ameliorating network-wide traffic conditions, and can do so by effective collaboration and resolution of potential conflicts (Logi, 1999)."

To meet these goals, CARTESIUS uses the FA/C DPS framework. This paradigm organizes effective cooperation among DPS agents in order to produce an acceptable solution in reasonable time by assuming that individual agents need not always have all the information about the global problem to completely and accurately solve their sub-problems. In particular, in some applications

> "there exist inter-agent constraints among subproblems that can be exploited to resolve the inconsistencies that may arise due to the lack of accurate, complete and up-to-date information. Agents can produce tentative, partial results based on local information and then exchange them with other agents to resolve local uncertainties and global inconsistencies (Logi, 1999)."

The CARTESIUS formulation maps these concepts to the multi-jurisdictional traffic management domain by describing a problem solving agent for each jurisdiction that is solely responsible for controlling its subnetwork. Each jurisdictional agent can first focus on its local problems, then exchange high-level information about its local candidate solutions with other agents. This exchanged information allows each agent to identify and remove local solutions that are at odds with the collection of management actions proposed by remote agents.

A key theoretical contribution provided by CARTESIUS is its definition of the features of the traffic management problem domain in a manner that is consistent with the FA/C DPS formulation. Toward this end, CARTESIUS leveraged an existing problem solving approach for traffic management, the Traffic Congestion Manager (TCM) (Logi, 1995), which used methodologies similar to those employed in existing FA/C systems.

Figure 3.1 shows the event processing that occurs in a three-agent CARTESIUS system when an incident is reported somewhere in the managed network. The event notification causes each

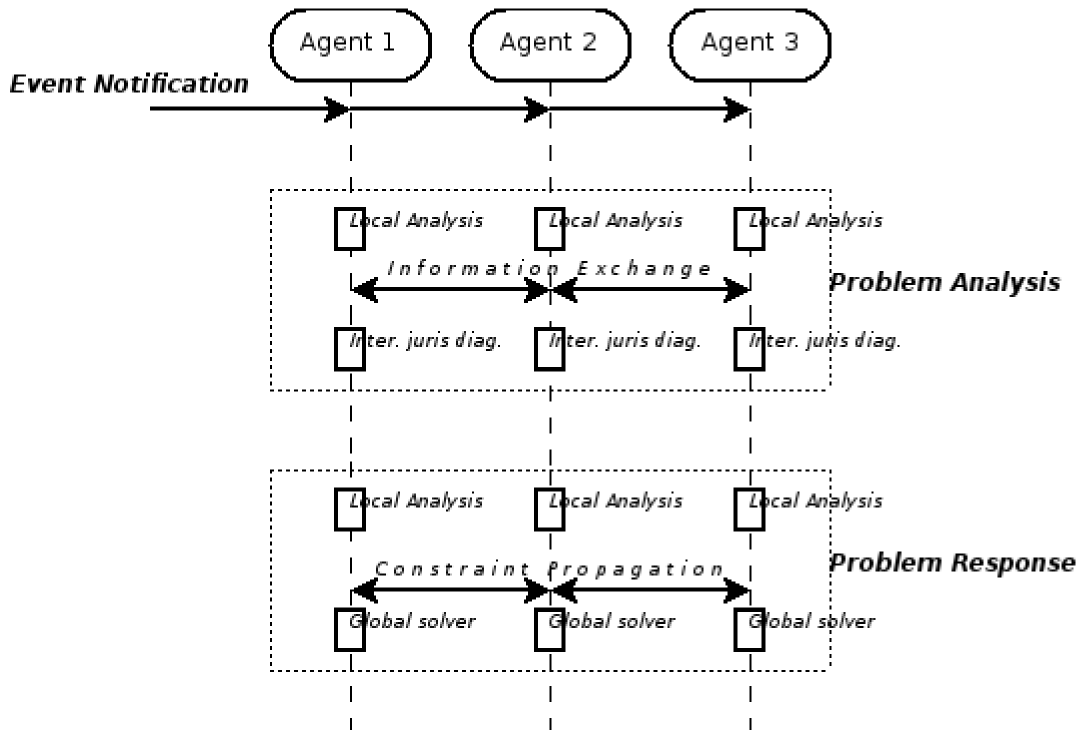

Figure 3.1 Message passing in a three agent CARTESIUS system.

agent to begin a two-stage analysis. During the *Problem Analysis* stage each agent characterizes all problems in its jurisdiction and attempts to identify their causes.

Information is shared between agents to allow causation to be determined across jurisdictional boundaries. During the *Problem Response* stage, each agent searches for strategies and the local control actions that implement them. The assumptions underlying these local control actions are propagated to other agents as constraints that those agents are requested to consider in a global solver step. At this point, each agent has its own copy of all feasible local solutions that conform to globally propagated constraints. A given agent's copy of this solution set contains detailed information about local actions to be taken for each solution and high-level information about the remote plans that are consistent with these local actions.

3.2.1 Local Problem Solving

The CARTESIUS approach is based upon the concept that each agent solves any problems local to its jurisdiction subject to limited information shared between the agents, which may include new potential problems caused by actions taken in neighboring jurisdictions. Thus, a problem solving agent can simply focus on mitigating a given set of problems using a local algorithm. The core features of the local problem solving algorithm that makes up the CARTESIUS philosophy are discussed below.

3.2.1.1 Knowledge Representation

CARTESIUS characterizes problems in the system using problem-specific frame-driven recognition techniques (Charniak and McDermott, 1985). At their core, these techniques depend on a "hierarchical database of knowledge packets," or *frames* (Minsky, 1974; Winston, 1975), that represent expert knowledge regarding the causal origins of problems in the transportation system as *problem frames* defined in terms of network topology and qualitative characterizations of sensor data. The database of problem frames are systematically compared to individual *network objects*—discrete partitions of the traffic network with particular operational characteristics (e.g., freeway on-ramp, freeway off-ramp, intersection approach, etc.). Each jurisdictional agent must therefore create and maintain this database of problem frames and network objects representing the network it manages. Specific knowledge about neighboring jurisdictions networks, control systems, particular problem types, and real-time data is not required. This partitioning satisfies the need for the system to preserve local autonomy and greatly simplifies data management.

3.2.1.2 Problem Characterization

The identified problems output from the local frame matching procedures, using real-time measurements from the system, collectively form a *problem state*. Each problem in the problem state identifies a *critical section* in the network for which the estimated demand exceeds the estimated capacity and, depending on the problem frame that matched to the current conditions, provides an assessment of the cause and general characteristics of the disruption based upon local expert knowledge embedded in the system's problem frame database.

The characterization process also recognizes the fact that a problem at a particular location in the network can lead to secondary congestion at other locations in the network (e.g., on-ramp spillback onto the arterial network). The local problem characterization works to identify *derives from* relationships between problems. These relationships are used in the logic that determines responses recommended by the CARTESIUS system.

3.2.1.3 Problem Response

To mitigate the identified problem state, the CARTESIUS is structured around a *planning* approach, in the Artificial Intelligence (AI) sense, to generate a sequence of local control actions that can change the estimated state of the transportation system to a more favorable outcome. The approach uses a heuristic Breadth First Search (BFS) algorithm that starts from the identified problem state, identifies subgoals that aim to move the system to a more desired state (e.g., a state in which demand and capacity are balanced), and finds concrete control actions that can achieve these subgoals. This general formulation is implemented for the traffic management problem by defining subgoals (with corresponding strategies) as follows:

- *increase capacity* at a congested location by **signalization**,
- *decrease flow* at a congested location by **traffic diversion**, and
- *decrease flow* at a congested location by upstream **metering**.

Defining concrete actions to implement these strategies, and algorithms to estimate those effects is a location and implementation-specific task because the available strategies and associated control actions will vary with each deployment—every jurisdiction is likely to employ different control subsystems that have distinct characteristics that are difficult to generalize.

3.2.1.4 Problem Monitoring

The type of high-level traffic management tackled by CARTESIUS inherently must deal with large degrees of uncertainty regarding knowledge of the state of the system and the effectiveness of implemented control strategies. Toward this end, CARTESIUS includes problem monitoring functions whose role is to track the performance of given control solution—in an absolute sense as well as whether the evolution of the system under that solution is consistent with expectation. The secondary function of the problem monitor is to filter out redundant information (such as the re-reporting of existing problems) to reduce the need for unnecessary analysis of problems that have already been considered.

3.2.2 Information Sharing for Distributed Problem Solving

The jurisdictional partitioning of traffic management described above can effectively isolate the management of problems to a single jurisdiction in the system as long the effects of those problems are contained in one jurisdiction. The existence of spillback effects across jurisdictional boundaries, and the fact that some control strategies (such as diversion) can place operational burdens on portions of the network outside the jurisdiction of the diverting agency, requires a means for sharing information between jurisdictions regarding such effects.

It is here that CARTESIUS DPS philosophy provides benefits by defining the specific types of information that need to be shared across jurisdictions in order to ensure that the collective of local responses to a given problem state produce a global solution that is acceptable to all jurisdictions.

The CARTESIUS adaptation of FA/C methods to this problem defines three distinct stages during problem solving when distributed agents should exchange information, and also what type of high-level information should be exchanged at those points. In each case, the general approach is to locally perform detailed analysis, then to create and post an abstraction of that analysis that can be used by other agents that then leverage the abstraction to refine their analysis. The process can be iterative and each instance must be carefully analyzed to ensure tractability and the avoidance of race conditions in the distributed system.

3.2.2.1 Problem Analysis: Spillbacks and derived problems

The characterization of traffic problems is complicated in a distributed system because of the potential relationship between problems as discussed above. Since such propagation can cross jurisdictional boundaries, the local problem characterization process described above must be augmented to handle information regarding the possible relationship between problems across jurisdictional boundaries.

CARTESIUS handles this problem by identifying all spillback effects and posting their characteristics to other agents for them to identify possible links between them. This information sharing allows agents whose jurisdictions are involved to compute derives from relationships between problems that cross these boundaries and to share those determinations with the agent collective.

3.2.2.2 Problem Response

The nature of partially decoupled DPS for complex, large-scale systems invariably leads to uncertainty and imperfect knowledge that makes the use of heuristic algorithms the only tractable approach. Any exact or heuristic optimization algorithm requires specification of an objective that can be evaluated in reasonable time (as dictated by the constraints of the algorithm and available computing power). Furthermore, the algorithm needs a means for determining how to reach the objective (i.e., a search direction).

CARTESIUS uses a problem solving heuristic that seeks to both avoid the spread of congestion that may lead to oversaturation and secondarily, to achieve a balanced ratio between network capacity and traffic demand. The CARTESIUS traffic management agent uses problem solving algorithm that incrementally searches a space of problem mitigation strategies, each of which can be *translated* by a corresponding control action or set of control actions.

Thus, a strategy to *reduce critical section demand through diversion* might be realized by setting a collection of Changeable Message Sign (CMS) to reroute traffic around the problem.

The expansion of the solution search tree through strategy and control actions generates new nodes representing an estimate of the state of the system if the control actions on the path from the root node to the target node are applied.

The selection of given strategy or the specific control actions that translate that strategy may require the satisfaction of particular conditions that define dynamic constraints that must be met for those strategies or actions to achieve their anticipated effects. These constraints may only apply to the local jurisdiction, but often, as in the case of diversion strategies requiring sufficient network capacity, they may propagate to remote jurisdictions.

CARTESIUS propagates such constraints as conditions that the remote agent must meet. The remote agent translates these conditions into strategies that have the goal of meeting the condition. These strategies are used to expand the search tree by branching from existing nodes to generate new portions of the search tree that will satisfy the conditions broadcast by other agents.

The process continues until no more conditional strategies are being broadcast by any agents. This indicates that the global search has been exhausted and the existing candidate solutions in the tree represent global collective of candidate solutions under consideration, including the extent to which each solution meets any broadcast constraints. The task of selecting a single global solution from among the set of candidates is one of removing solutions that are inconsistent with local or remote constraints. Since each agent has the same list of the available feasible solutions, it is now possible for the agent collective to jointly select a single global solution consisting of a combination of locally acceptable control actions.

Chapter 4

Integration Strategy

In Task Order 5313, we laid out recommendations for moving the CARTESIUS system closer to deployment (Rindt and McNally, 2006). These were based on findings from a detailed analysis of CARTESIUS, its theoretical capabilities, its practical capabilities, and our assessment on feasible directions for the system and where it fit into the future of traffic management in California. This chapter summarizes these findings in the context of integrating the system to work with CTNET.

4.1 Alter CARTESIUS

4.1.1 Legacy Software

4.1.1.1 Barriers to Deploying the Prototype

The original CARTESIUS prototype proved difficult to deploy in even very limited capacities involving in laboratory testing. The reasons for this difficulty were numerous and cross cutting, but collectively they severely limited its potential for real-world deployment in the near to medium term. This led to a number of key recommendations that shaped the course of this project:

- Re-implementing the CARTESIUS traffic management prototype using appropriate general purpose languages and communications protocols to free it from costly vendor lock-in that ultimately restricts its development.
- Exploring modern, service-oriented messaging architectures that allow individual components to be loosely coupled using event-driven methodologies to achieve coordination as necessary.

- Reworking of the CARTESIUS traffic management agent design that separates analytical functionality from interface functionality.
- Adoption of an out-of-the box service-oriented messaging architecture for information sharing between CARTESIUS agents and between those agents and external data sources and control subsystems.
- Redefinition of the CARTESIUS data model that focuses on leveraging data standards and software tools and libraries that operate on those standards. Furthermore, to facilitate broader use of CARTESIUS-related data, we recommended the use of a general-purpose relational database that can expose core data used by CARTESIUS to multiple sources.
- Focusing CARTESIUS on the problem of estimating the implications of anticipated control actions given currently deployed control algorithms. The problem of actively adjusting control (or advising local controllers) to mitigate non-recurrent congestion using customized control algorithms can be investigated where necessary, but preference should be given to existing research focusing on those problems independently.

4.1.1.2 Aging Priorities: Functional Redundancy in CARTESIUS

The shortcomings discussed in 4.1.1.1 can be considered as a failure to properly bound the functional role of the CARTESIUS management agent. In a practical sense, however, they are primarily the result of shifting the CARTESIUS project’s priorities. The history of the original CARTESIUS prototype led to its implementation as a “Swiss Army knife” of traffic management that ultimately sought to optimize all components of the transportation system on the basis of estimation models contained within CARTESIUS itself. This development was largely driven by the need to create a functional prototype that could demonstrate the efficacy of the core CARTESIUS algorithms for finding solutions to non-recurrent congestion for a given set of potential control actions. Unfortunately, this development path also resulted in CARTESIUS assuming responsibility for functions outside its mandate. Thus, the CARTESIUS prototype became a system for optimizing traffic signals, ramp meters, and variable message signs using its own internal algorithms.

Assuming responsibility for all of these functions makes maintenance of the system challenging and makes a particular CARTESIUS traffic management agent’s implementation susceptible to obsolescence since it is difficult to incorporate newly available technologies and algorithms into a monolithic agent. Furthermore, such feature concentration limits the scalability and fault-tolerance of the system. Finally, developing standards in the industry (Iteris, Inc, 2002) put the CARTESIUS prototype at odds with current and likely future of traffic management systems.

In an age when the deployment of research projects is a primary goal of sponsors, researchers would be well served to plan accordingly from the early stages of research. This is particularly true for research that ultimately targets one or more functional roles that exist in mature, complex operating environments such as ITS.

Based on these findings, we made the following recommendation:

- Continuing CARTESIUS research be guided by a well defined set of functional requirements derived from the known capabilities of the CARTESIUS approach and from the realities of the real-world technical and institutional landscape.

4.1.1.3 Finding Synergy

As has been suggested in earlier sections, the existing CARTESIUS prototype suffers from a lack of focus that can be alleviated by clearly defining CARTESIUS's role in traffic management. A side effect of proper tasking of CARTESIUS is that it will make it easier for CARTESIUS to find synergy with existing traffic management technologies, and more importantly, with developing research that has the potential to broadly impact the quality of traffic management systems.

4.1.2 The Way Forward

4.1.2.1 Simplification and Openness

The core CARTESIUS philosophy is a powerful conceptualization of the multi-jurisdictional traffic management problem. However, the findings of chapter 3 argue that the current prototype is a dated realization of this philosophy that must be revised. This begs the question of whether the revision should entail a modification to the existing prototype, or whether a complete reimplementation is warranted.

In our judgment these points support reimplementation primarily because it is clear that the existing CARTESIUS prototype attempts to do too much in incident management given its capabilities. This makes the benefits of reimplementation greatly outweigh the benefits of revision—particularly since the latter most revolve around the value of an existing code base that solves problems outside its appropriate functional role.

As a result, we recommend focusing on CARTESIUS core strength as a system for managing conflicts between the potential actions of different jurisdictions and negotiating alternative strategies to mitigate those conflicts. This does not preclude using a CARTESIUS management agent to perform optimization, but the most feasible course of action for deployment is to emphasize CARTESIUS monitoring and general advisory role regarding multi-jurisdictional conflicts.

4.1.2.2 CARTESIUS as an Organizing Principle

That the CARTESIUS approach to multi-jurisdictional traffic management via DPS is a promising management strategy should not be in question; the results of the original research demonstrate the power and potential of the underlying concepts. The challenge lies in transforming these concepts into incrementally deployable components that are compatible with existing systems.

The real-world landscape is one of existing traffic management subsystems, each using a variety of variably standards-based approaches to managing traffic. For most jurisdictions, the emphasis is on *strategic optimization of control settings* (on roughly annual timescales) to match prevailing demands. Top-down management of non-recurrent congestion is generally limited to rapid deployment of emergency and operations services (to clear disruptions) and information provision (to notify travelers of disruptions). Locally adaptive control schemes that obviate the need for centralized re-optimization are in limited deployment but are likely to increase in the future.

Whether more aggressive top-down management is warranted in operational contexts remains an open question. CARTESIUS research has certainly shown promise for centralized management through active re-optimization of field controllers and diversion-oriented information provision under varying assumptions and a variety of system failures. But no comparisons have been made to truly adaptive systems that rapidly respond to changing demand patterns.

Assuming such systems continue in development and eventually see broader deployment, what is the future role of top-down management approaches such as that used by the current CARTESIUS agent prototype? We believe that the answer lies in the strength of viewing multi-jurisdictional traffic management as a DPS process. Regardless of the local algorithms employed, multi-jurisdictional traffic management is inherently an instance of DPS. Each controller, or control subsystem takes action on the basis of local knowledge to achieve some local goal (e.g., minimizing delay for the managed subsystem). The extent to which these local goals assist or hinder system-wide goals is ignored for the most part.

For instance, virtually all adaptive traffic signal algorithms are based on dynamic programming approaches to estimating prevailing demands and adapting the control parameters of local controllers to best service those demands. These demand estimation procedures, however, will have difficulty responding rapidly to discontinuous changes is demand caused, for instance, by massive re-routing of freeway traffic to the arterial system. Foreknowledge of such rapid demand shifts could reasonably be added to local adaptive control algorithms to rapidly respond to demand shifts and prevent the oversaturation and queuing that can cripple arterial control systems.

The CARTESIUS philosophy offers an organizing principle for traffic management systems that lends itself to unobtrusive information sharing that can be leveraged by control subsystems, when feasible and/or desired. It also provides a mechanism for distributed diagnosis of interconnected problems in the system. In short, CARTESIUS becomes an organizing principle about how to conceptualize control resources in the transportation system.

The degree to which a given jurisdiction participates in the CARTESIUS system could vary from the most abstract level of information sharing down to detailed participation of each controller in the global problem solution. This flexibility makes CARTESIUS simpler to deploy incrementally by allowing managers to minimize the risk they take in participating with other CARTESIUS components.

4.1.2.3 CARTESIUS as an Information Sharing Protocol

Adopting the CARTESIUS DPS philosophy requires a protocol for sharing high-level information related to traffic management between jurisdictions. We propose transforming the information sharing techniques used in the original CARTESIUS two-agent implementation into a general-purpose collection of messages that can be leveraged by any CARTESIUS aware component as it sees fit. We propose that this can be a model for general purpose TMC to TMC traffic management communication that may have relevance to developing standards in the National Intelligent Transportation Systems Architecture (NITSA).

As discussed in section 2.1.2, CARTESIUS was originally designed using a hierarchical frame-based representation of possible problems in the system that relied upon representing the transportation network as a collection of *network objects* whose dynamic state can be compared with the problem frame database. This is a somewhat unconventional approach to network representation that may or may not be convenient to a particular jurisdiction.

This highlights the broader problem of defining data representations for sharing information across jurisdictions. Any collection of jurisdictions that agree to participate in a collaborative management system must ultimately agree on how common knowledge across jurisdictions is represented. If CARTESIUS is implemented as a monolithic agent that acts to control all resources, this problem is mitigated to the extent that a custom, CARTESIUS-compatible network representation can be built for each implementation. This creates a higher cost for participation, however, that may not be palatable to potential participants.

Ideally, a deployable CARTESIUS system would play nicely with a broad range of network representations that may be in use by existing management systems. One approach is to simply perform data transformations on-the-fly as information is exchanged between CARTESIUS components and external management systems.

With the above caveats in mind, the knowledge partitioning approach endorsed by the CARTESIUS philosophy is a good one that minimizes the need for a given jurisdiction to maintain datasets representing the assets of neighboring jurisdictions. This, in turn, supports the goal of allowing jurisdictions to control access to possibly sensitive data about their systems.

The level of common knowledge needed is ultimately dictated by the types of information sharing required for CARTESIUS to perform its function. For CARTESIUS, the critical information falls into three areas:

- Messages related to the problem state,
- Messages related to strategy-based problem solving, and
- Messages related to problem monitoring.

4.1.2.4 CARTESIUS as a TMC Operator's Tool

From the perspective of the TMC, a CARTESIUS agent should be viewed as a high-level advisory system that listens to CARTESIUS-related messages and offers operators insights as to

- the state of the transportation system as a whole as it relates to the local jurisdiction,
- the role of a particular agency's jurisdiction in the performance of that system, and
- the collection of possible mitigation actions, given available resources, that are consistent with (do not conflict with) neighboring jurisdictions actions

The advisory role reduces the core operational responsibility of the system, taking it out of the direct control loop. The CARTESIUS agent's role as a direct controller of the system would be a jurisdiction-dependent decision and would require CARTESIUS to become an authenticated client to existing control subsystems.

In this perspective, the role of the CARTESIUS agent is to offer view of the system from a DPS perspective. Its role would be to highlight potential conflicts given that perspective and, where mechanisms have been installed, broker solutions to those conflicts. What this means in a practical sense is that the entire system need not be re-architected to adopt the DPS view of traffic management for CARTESIUS to be useful. Furthermore, it would remove significant burdens from the CARTESIUS prototype to take active management of the system and instead would offer monitoring, advisory, and post analysis capabilities that leverage the features of the DPS viewpoint to offer a high-level view of conflicts in the global management of disruptive events in the transportation system.

This is a reworking of the original CARTESIUS agent's role, which sought to actively manage non-recurrent congestion in a jurisdiction from the top down. In place of the former "heavy-controller" approach adopted for the original agent, the reimplementation would be a lighter-weight tool that could more easily interoperate with existing or future systems. The CARTESIUS approach to monitoring problems could also make the agent useful for evaluating the performance of various TMC strategies and different levels of inter-jurisdictional coordination used during incident management.

An additional finding that has arisen in related research into TMC Performance Evaluation conducted under the Testbed (Rindt and Recker, 2007) is that the processes used in TMC operations are highly tuned for efficiency and speed. Adding additional software tools to the TMC processes is challenging because it must be tightly integrated into an already busy process. Software that fails in this integration is often ignored and therefore fails in its purpose. The broader lesson of this finding is that attempts to build new interfaces for the TMC are probably doomed to fail unless they are part of a broader systems engineering effort that involves all stakeholders from the beginning. Given this finding, the research team decided to de-emphasize the development of the CARTESIUS GUI in favor of focusing on generating advisory information that can be integrated into TMC process through proper systems engineering on the time scale dictated by TMC managers rather than the research process. In short, CARTESIUS was recast as more of a data-centric product than an interface centric product. We believe this increases the odds that the system will find its way into deployment in Caltrans operations.

4.2 Leveraging Existing Software: CTNET

Within the context of the prior discussion, the importance of the CARTESIUS/CTNET integration becomes clearer. The desire to focus CARTESIUS on tactical management means that CARTESIUS must be increasingly dependent upon other components of the transportation management system to provide direct operational control of the traffic system. In the domain of arterial traffic management systems, Caltrans CTNET software fills this niche very well—providing direct, two-way access to traffic controllers, including the ability to collect live system detector data and to alter the timing plans or active timing patterns associated with field masters and the local controllers they manage.

In this relationship, it’s clear that CARTESIUS must function as a client within the CTNET client/server architecture. Otherwise, the functional roles of the components become muddied, with either CARTESIUS moving into the realm of operational control or CTNET moving up into tactical and strategy control policies involving portions of the transportation system beyond the scope of an arterial traffic management subsystem.

4.3 Link Related Modules

Because of its global emphasis, CARTESIUS must interact with traffic management subsystems beyond CTNET. Because of the capabilities available in the Testbed, the original prototype was more closely integrated with the information and control systems of the Caltrans District 12 TMC. The Testbed's subsystems provide direct access to the D12 ATMS, which provides detector data as well as current control settings for changeable message signs and ramp metering subsystems. This connectivity continued to evolve through during the project, spurred by parallel efforts to foster better information sharing between Caltrans districts and between Caltrans and external entities.

Beyond direct measurement of traffic states, CARTESIUS is also dependent upon predictions of future traffic states. The original prototype used embedded models to obtain these predictions, but it was always assumed that the targeted research would produce better predictive models that CARTESIUS could tap as external modules. Indeed, the development of several path-flow estimators in the Testbed vindicated this assumption, and the new implementation of CARTESIUS was developed with eventual connectivity to these systems in mind, event though they were not ready for integration during the course of this project.

Chapter 5

CARTESIUS Development

5.1 Overview

This chapter details the design, implementation, and component verification of the CARTESIUS re-implementation carried out during this project. The design was tailored to satisfy the requirements for the next-generation of CARTESIUS detailed in the final report of PATH Task Order 5324 which are summarized in the Appendix and referenced by their labels. FR- prefixes refer to functional requirements, IR- prefixes refer to user interface requirements, ER- prefixes refer to external interface requirements, and DR- prefixes refer to database requirements. Unless otherwise noted, all references to the original version of CARTESIUS refer to the original thesis describing the work (Logi, 1999). After describing some of the broad design choices made in the re-implementation, we detail the design of the various software components that make up the re-implemented CARTESIUS and discuss how we verified these components meet or do not meet the specific requirements from TO-5324.

5.2 Initial Design Choices

The re-implementation process began with the original CARTESIUS source code, which was exported from the G2 expert system shell (Gensym, 1995) in which it was originally deployed. The code was analyzed and mapped into a collection of rough flow charts indicating how the original CARTESIUS processed input data through its internal algorithms and arrived at a response strategy. Based upon these charts, we set out to achieve a number of modifications to the code in order to meet the above collections of requirements.

5.2.1 Development platform

Among the early design choices made involved the platform for development. As we noted in our earlier analyses (Rindt and McNally, 2008) the original implementation's use of the G2 shell tightly integrated user interface objects and underlying code. This commingling of interface code and core logic hampered maintainability of the system. Furthermore, the per-seat cost of G2 and related software licenses create a potential barrier to uptake of the system by potential users. Based upon these factors, we chose to explore development using a general-purpose programming language that offered platform portability, substantial availability of libraries to meet the detailed requirements for the system, and general ease of development and maintainability.

At the time of development, a wide range of potential programming languages, toolkits, and frameworks were available ranging from low-level general-purpose programming languages like C and its descendents to high-level scripting languages such as perl. To simplify the choice, the team used some broad judgments regarding the suitability of various solutions. Generally, these judgments were made by identifying whether systems similar to CARTESIUS had been developed using a particular programming language. This reduced the choice set to a C/C++-based solution (similar to CTNET), a C# solution using Microsoft's .NET libraries, a Java solution using the wide range of available Java libraries, or a Python scripting solution that would interface with core C++ libraries.

Given these alternatives, we developed a set of needs for the development platform and used these to score available platform alternatives. First, we considered the portability of the alternative---particularly whether the system could easily be moved between Windows-based and UNIX/Linux-based systems. The increasing prevalence of Linux-based servers and their relatively low cost of maintenance increased the importance of portability---as did the consideration that the system may one-day be deployed in embedded settings.

Next we considered the relative performance characteristics of the alternatives. CARTESIUS is an analytical platform that performs heuristic searches of complex problem spaces making performance an important issue. That being said, the performance criterion was less important as the existing implementation was able to solve relatively complex search problems using older (ca

Table 5.1 Scores of the CARTESIUS development platform alternatives

Criterion (weight)	C/C++	C#	Java	Python
Portability (2)	2	0	2	1
Performance (0.5)	3	2	2	2
Ease of use (1)	1	2	2	1
Libraries (2)	2	2	3	1
Composite Score	**1.9**	**1.3**	**2.4**	**1.1**

1998) hardware and the initial screening described above removed any alternatives that we deemed too slow.

We also considered how easy the alternative would be to use for development. There are many factors to consider in this case, including the features of the language, the developer's familiarity with it and its associated libraries, and the availability of integrated development environments (IDEs) that support the language. Ultimately this is a subjective measure, but is important for determining potential success of the project.

Finally, we considered the availability of libraries for modeling, communications, database access, and other features. Given the dependence of CARTESIUS on external data, analysis tools, and its need to interact with other systems, this feature was critically important.

We then scored each alternative for each criterion and developed an average score for each using weighting specified by the developers as shown in Table 5.1. The resulting choice was a Java-based solution using available analytical, communications, and database libraries to speed development. Java's maven build system also simplifies portability and deployment to diverse platforms. This combination can be developed using a variety of interchangeable IDEs to maximize developer flexibility.

5.2.2 CARTESIUS as a bundle of services.

The original CARTESIUS was developed in the mid-1990s, during the early stages of the Internet and modern middleware technologies. The system relied on a combination of proprietary inter-process communication (IPC) and error-prone, low-level socket-based communication to share data between CARTESIUS agents and external data sources. In the intervening years, a number of middleware technologies have matured that fit the communications needs of CARTESIUS. These include general-purpose solutions such as the Common Object Request Broker Architecture (CORBA) (Pope, 1997) to various web-based technologies such as the Simple Object Access Protocol (SOAP) (Gudgin et. al., 2007) and XML-RPC (St. Laurant et. al, 2001).

In analyzing the requirements for CARTESIUS specified by PATH TO 5324 (A complete listing of these requirements are reproduced in the Appendix), the design team recognized that CARTESIUS could be viewed as a loosely coupled bundle of services that each perform specific tasks to independently operate on data to produce new information that can be used by other services. This view is consistent with Service-Oriented Architecture (SOA) design principle. CARTESIUS fits this model in the sense that the events that drive the need for incident management can be injected into a chain of processing constructed from a collection of core logic that comprises CARTESIUS functions. The output from this processing is response recommendations that can be integrated into incident management processes at the TMC---either by providing a new operator interface supplying the guidance, or by integrating the guidance into existing interfaces.

Additionally, the flow-oriented structure of processing imagined for CARTESIUS suggested that using an Enterprise Service Bus (ESB) architecture (Chappel, 2004) might be the best approach for realizing the system as a SOA. An ESB is a collection of middleware for realizing SOA by providing message routing and transformation services on top of a general-purpose communications bus to which multiple data sources (coming via multiple communications protocols) and service components can send and/or receive messages. Very simply, an ESB simplifies the stitching together of SOA components by providing a means for specifying how information should flow between them. Designing this type of system focuses on clearly defining the data to be exchanged, making the processing logic atomic, and specifying the information flows using the tools of the particular ESB software employed.

Table 5.2 Relative merits of ESB platforms for CARTESIUS deployment

	Service Mix	JBOSS ESB	Mule ESB
Features	2	2	3
Ease-of-use	1	2	3
Stability	1	2	2
Support	2	3	3
Composite	1.5	2.25	2.75

In analyzing ESB software alternatives, the team considered a number of alternatives with a bias toward Java-based solutions given our earlier analysis favoring that language. The most accessible, free software solutions were Apache Service Mix, JBOSS ESB, and Mule Soft's Mule ESB. We scored the relative merits of these systems on features, ease of use, stability, and the activity of the support community using an equal weighting. The results, shown in Table 5.2 favored the Mule ESB, which provides an excellent and easily configurable feature set along with good documentation and a vibrant user base offering support.

The Mule ESB offers additional advantages, most notably its use of the Spring Framework which offers configuration advantages such as Inversion of Control---in which configuration settings are used to inject dependencies into the application at runtime making it simple, for instance, to switch between database implementations.

The choice to structure CARTESIUS as a SOA using an ESB platform structured the design of the remainder of the system by providing clear guidelines on how to decompose the system into smaller reusable components that are easier to develop, test, and deploy. The design of the various components of CARTESIUS is described in detail in Section 5.3.

5.2.3 Database requirements and abstractions

Since Java is a well-known object-oriented programming language, it naturally leads to the use of object-oriented data abstractions. The development team considered a number of data abstraction models but settled on the use of the Data Access Object (DAO) design pattern, which is a foundation of Java development (Sun Microsystems, 2007). Among the advantages of this approach are that the core business logic of the application can be insulated from the details of the data source. Instead of worrying about low-level access details, the logic is coded to an interface that specifies the type of data that will be made available from some DAO that is instantiated at run-time and subsequently used to obtain the necessary data during execution. This not only simplifies development of the core business logic for ESB components, but allows the system to easily use alternative data sources as they become available by simply creating implementations of the DAO interface for the new sources.

The domain model for CARTESIUS is, in many ways, very specific to CARTESIUS. The core objects in the system include:

- A traditional network graph representing the traffic network as a collection of nodes and links with various attributes detailing their performance characteristics (number of lanes, free flow speed, etc.)
- The network as a collection of NetworkObjects for the purposes of problem diagnosis (discussed later). NetworkObjects are classified into 5 basic types:
    - RoadPiece (a uniform section of roadway)
    - Capacity Reduction (such as a section of roadway with a lane drop)
    - Merge (where two road join into a single facility)
    - Diverge (where two roads split off from a single facility)
    - Intersection approach

  The network graph is partitioned into a finite set of NetworkObjects where are used in the event diagnosis process (discussed later).
- The demand in the system is specified as hourly flows along an enumerated list of paths, each specified as a list of network links.
- A collection of sensors mapped onto a subset of the network links, which provide occupancy and speed information for the purposes of estimating the current state of the system

- A collection of traffic control and information devices including traffic signals, ramp meters, and changeable message signs as well as their various control plans.

These basic objects are used by CARTESIUS to model the transportation system for the purposes of incident analysis and response. The details of these processes and their implementation follow in the next section.

5.3 Implementation, and Component Verification

The software requirements for CARTESIUS that were specified in the PATH TO-5324 final report (Rindt and McNally, 2008) were organized into a set of atomic software components designed to operate atomically. Each of these was then designed, implemented, and verified versus the relevant components as detailed in the next section.

5.3.1 Core functional requirements

The core functional requirements describe how the basic functions of CARTESIUS as an incident management system should be organized. These requirements broken into 4 major areas whose relationships are shown in Figure 5.1:

1. Core functional requirements, which spelled out what the core models should do in terms of traffic analysis and response computations;
2. User interface requirements, which described expectations for how users should interact with the CARTESIUS system;
3. Database requirements, which described how data related to CARTESIUS operation and analysis should be stored and maintained; and
4. External interface requirements, which focused on how CARTESIUS should interact with data sources, other models, and with other CARTESIUS agents.

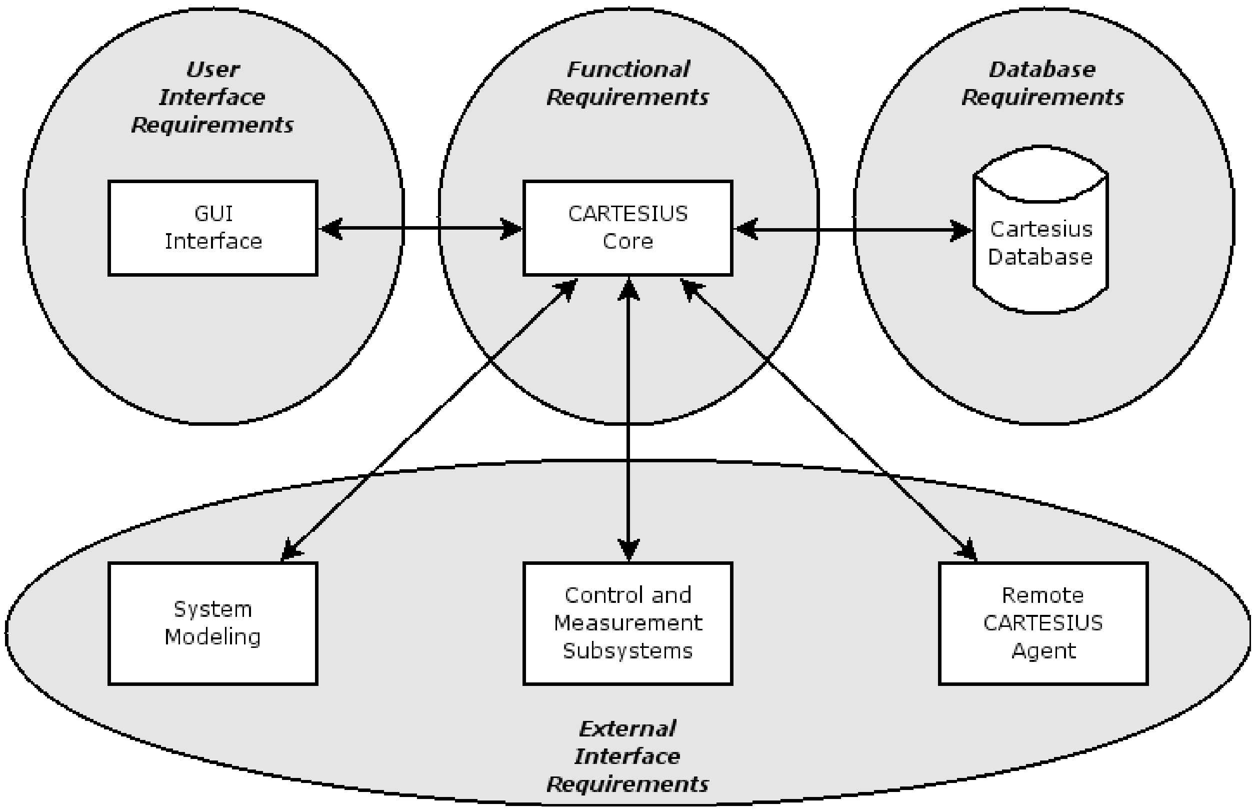

Figure 5.1 Relationships between the different classes of requirements for CARTESIUS

The core functional requirements for the system are further divided into four primary functions as shown in Figure 5.2. The *System Monitoring* function is responsible for tracking the state of the system to identify when disruptions occur that CARTESIUS should attempt to manage. The *Event Analysis* function must produce a problem characterization that analytically describes the disruptions and serves as CARTESIUS's representation of all the problems in the system. When the active problem characterization is modified the *Problem Response* function finds a problem. The design of the system created distinct software packages for each of these clusters of requirements. We discuss the designs of each in the following sections.

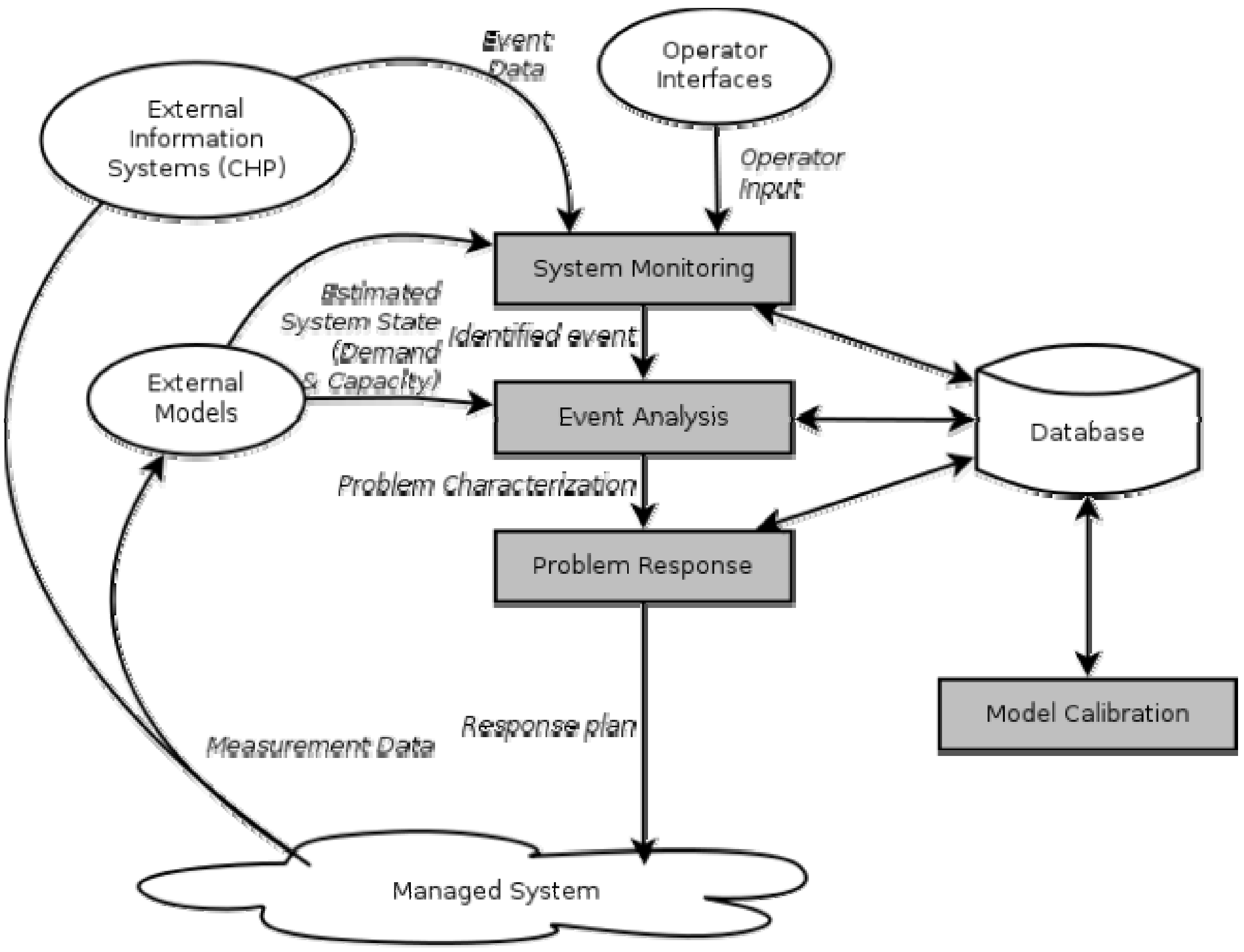

Figure 5.2 The four types of core requirements (shaded) used in the redesign of CARTESIUS.

5.3.1.1 System Monitoring

The system monitoring package was developed to perform two sub-functions: state measurement and state forecasting. The state measurement function provides CARTESIUS with access to state measurements obtained from field devices or other third party sources (such as the TMC), while the state forecasting function provides CARTESIUS with access to estimates of the future state of the system based upon current and historical data. Since both these measurements and estimates come from external sources, this requirement was met by defining a set of DAOs specifying the interface for accessing the necessary system monitoring data. In particular, DAO interfaces were specified for the following data types.

- VDS, VDSStateData: These classes and their associated DAOs provide access to VDS data collected from sensors in the roadway. In the Testbed, these data are provided by Caltrans sensors and made available via PeMS and/or the ATMS FEP data feed. The VDS class specifies a VDS station and it's associated metadata (id, location, lane geometry, etc) while the VDSStateData class represents volume and occupancy data collected over some time period (30-seconds, 5-minute, 1-hour). The VDSStateData class can be used to represent raw or processed VDS data as needed. Thus, if the implementation supports it, the VDSStateData can also provide speed and capacity estimates for the VDS section.
- IntersectionController, IntersectionControllerState: These classes and their associated DAOs provide abstract representations of intersection controllers and their associated states---including system detector measurements. The ability to adjust timing plans is currently limited to adjustment of green minimums, maximums, and extensions. This limitation is driven by the Capabilities of the CARTESIUS logic and can be relaxed as needed. Note that the CARTESIUS connection to CTNET is provided as an implementation of the DAO using custom Java libraries described in Chapter 6.

Collectively, the above data types and their DAO implementations satisfy functional requirements 1 and 2 (FR 1 and 2) as specified by TO-5324. State measurements and estimates can be obtained by accessing available live data streams or databases (e.g., Testbed databases). Furthermore, forecasts about future states are available by accessing aggregated historical data and applying those estimates to determine near-term demands. This satisfies FR 3.

Verification that FRs 1-3 were met at the component level was achieved by testing using VDSStateData implementations to Paramics (via a DAO implementation using Testbed plug-in described in section 7.2). Similarly, we implemented the IntersectionController DAO using the JCTNET library and verified its operation using simulation (see section 6.3.3).

The above inputs provide the necessary data for the monitoring functions, which are carried out by the Monitor class. The Monitor class provides a checkStatus method to evaluate the current state of the system. The state is assumed to be constant unless an event triggers the monitor to run its checkStatus method. The purpose of this method is to determine whether any of the existing events in the system are severe enough to warrant mitigation efforts (this task is specified by FR 4). In this implementation, the system uses an Event class to such occurrences:

- Event: An externally specified event that might impact the performance of the transportation system. Typically this would be mapped to CHP and TMC logs of incident activity.

Events are passed to the monitor externally via the ESB, which calls the monitor's checkStatus method to initiate event processing. A change may be the addition or deletion of an event from the active event set, or it may be a change in the characteristics of a particular event in that set—for instance a modification in the expected duration of a lane closure that is input by the operator. Whenever this happens, the checkStatus method considers all Events by accessing the EventDAO in addition to the Event passed by the ESB. For each such event, the monitor determines whether the demand exceeds the estimated capacity of the section following the original CARTESIUS implementation. If so, the Event is translated into a CARTESIUSEvent using a transformation class that implements a doTransform method taking an Event object and returning a CARTESIUSEvent object. The implementation is designed to allow any number of transformations to be applied to produce the CARTESIUSEvent object. The current implementation, however, is limited to a single transformer that computes the estimated reduction in capacity on the basis of the number of lanes closed by the event. No transformer is currently available for demand-related impacts (such as special events).

- CARTESIUSEvent: The CARTESIUS representation of a system disruption (see the Event data type above) that results in demand/capacity imbalance or "significant" deviation from normal operations in either the demand or the capacity of the system with respect to historical demand and/or capacity. A CARTESIUSEvent differs from an Event by including a list of DemandEffects and/or CapacityEffects associated with the event. CARTESIUSEvents are the target of CARTESIUS responses, which seek to reestablish the balance between demand and capacity by selecting set of control actions.

- CapacityEffect: Effects of this type represent estimated capacity reductions caused by an event in question (e.g., due to a lane being closed). They are specified as a network location (e.g., a freeway section) and a capacity reduction (or increase).

- DemandEffect: Effects of this type represent estimated demand increases caused by an event in question (e.g., due to a special event). They are specified as deviations from prevailing demands on given paths through the network.

It is possible that an Event has already been observed and processed by the system and that therefore a CARTESIUSEvent already exists. If so, this instance is modified instead of a new CARTESIUSEvent instance being created. Note that the translation of the Event into a CARTESIUSEvent satisfies FR 4, requiring CARTESIUS to estimate the impact of a particular event on traffic operations, and FR 5, requiring CARTESIUS to implement a function that can identify the onset of an event in the system---in this case using the threshold model.

The resulting collection of CARTESIUSEvents is returned from the checkStatus function and dispatched by the ESB for event analysis. Here, the ESB is used to implement FR 6, that requires identified events be sent to any components that have requested event notifications.

5.3.1.2 Event Analysis

The event analysis package was developed to update the CARTESIUS problem characterization based upon analysis of prevailing events. This function is controlled by a top-level Analyzer class that exposes an analyzeEvents method to the ESB. When the ESB passes an updated set of events to the analyzer, it carries out event diagnosis using a distinct style of event characterization. The purpose of the event diagnosis step is to identify the location of event and what type of event it is, which is later used to select appropriate response strategies. The location is specified as a *critical section*, which is basically the facility and a post mile or cross street where demand exceeds capacity. Following the original implementation, the *type* of the event is assumed to fall into one of three classes: an accident, insufficient capacity at a signalized intersection, or a bottleneck at a freeway on-ramp.

The original CARTESIUS event diagnosis implementation used a frame matching technique to classify events. While we had some questions about the general portability of the method used, we determined that modifying the system to use an alternative would require a greater level of redesign than was warranted for this project. The general procedure in the method is to loop over all network objects possibly affected by an event (as specified in the CARTESIUSEvent instance) and compare them to a given database of general ProblemFrames representing conditions that would be consistent with particular problem types. The conditions are generally specified as thresholds on sensor data for particular groups of sensors on adjacent facilities. These groupings are specified *a priori* as a set of NetworkObjects that represent the system (see section 5.2.3). The purpose of the Frame Matching method is to transform the specific traffic data observed with a particular problem into one particular *qualitative representation* from a finite set of such representations. This allows the response algorithm logic to distill the infinite state space into a searchable subspace.

We implemented the original frame matching algorithm using Java reflection techniques using the original implementation as a reference. In this case, a Frame is defined as a collection of Slots, each of which refers either to attributes with which candidate objects attributes are compared. Java's reflection capabilities are used to implement a general algorithm whereby the attribute names in the ProblemFrame definitions are used to obtain the necessary methods to access the related data in the candidate object.

The re-implementation used a direct dump of the original CARTESIUS ProblemFrames to generate the Java source code to recreate them in a compatible format. Details of the knowledge base can be found in the original CARTESIUS reference. The library of problem frames are structured such that all events will be matched with a ProblemFrame by including default ProblemFrames representing cases with unknown causes. In this manner, all events are diagnosed with a cause. For each event matched with a ProblemFrame representing an imbalance in the system with a known cause, the algorithm generates a Problem object that includes the type of problem identified, its cause, and the NetworkObject and critical section affected. The Problem object also provides problem-specific methods for computing supply-demand changes at the critical section, which follow the logic of the original CARTESIUS implementation. Events that match to ProblemFrames without causes are logged for later analysis so that the problem frame database can be improved. These features satisfy FR 7-11 from the requirements. They were verified by using a test dataset designed to match with all ProblemFrames in the library, which confirmed that the library classified all cases it was presented with, including outliers. The problem diagnosis algorithm does not currently consider spillback effects from neighboring jurisdictions and therefore does NOT satisfy FR 12.

Because CARTESIUS assumes that Problems in the system may be related, a given set of Problems are collectively represented by a ProblemDescription object that represents the "global" problem that CARTESIUS must seek to mitigate through response options. If the state of any of the sub-problems changes, the corresponding global ProblemDescription must change accordingly.

5.3.1.3 Event Response

The event response function is designed to be triggered by the ESB when a new ProblemDescription is produced. The event response is focused on formulating local response sets from available control actions and collaborating with other agents to develop a consistent global strategy with non-conflicting actions across jurisdictions.

5.3.2 User Interface Requirements

During the course of development, a decision was made to drop the development GUI interface in favor of designing CARTESIUS to run as a service that offers recommendations based upon a set of pre-defined configurations dictated default responses that would normally have been required by the operator. This approach is intended to focus on the delivery of information rather than to initiate control actions, and is consistent with the philosophical shift discussed in section 4.1.2.4 of making CARTESIUS a data-centric product rather than an interface centric one. As a result all interface requirements (IR 2-17) were deferred for later development except for IR 1, which required that core analytical functions be separated from the GUI. This requirement was met in the transition to a data-centric system. This will ease the later integration of CARTESIUS products into existing and new TMC processes.

5.3.3 External Interface Requirements

5.3.3.1 Distributed Problem Solving Interfaces

The challenges in re-implementing CARTESIUS as a deployable system were most evident in the implementation of the distributed problem solving interface. The promise of the original implementation and its theoretical underpinnings remain. However, the details of practical implementation proved beyond the scope of the current project. The reasons for this are varied but center on translating the original implementation, which was hardcoded to interface two CARTESIUS agents into a prototype, into a general purpose implementation that can share data across N agents. The constraints of this project made pursuing this general purpose goal unrealistic.

As a result, the re-implementation was designed as a single agent system, with hooks to allow for the later incorporation of a multi-agent information. Thus, requirements ER 1-4 were not met by this re-implementation. It is notable that this development mimics that of the original CARTESIUS implementation, which began as a single agent system and was later extended for multi-agent purposes.

5.3.3.2 External Data Interfaces

The specific data required by CARTESIUS from these subsystems is described by the core functional requirements. They include data from:

- Event/incident notification systems
- Measurement subsystems
    - Current flow, occupancy, and possibly speeds
- Control subsystems
    - Traffic signals
    - Ramp meters
    - CMSs

The ability to receive events regarding disruptions to the system from external sources was implemented by creating customized transformers to translate event data from the CHP CAD feed and/or the District 12 Activity Logs into Event objects that are placed on the ESB to trigger action by the CARTESIUS monitor described in section 5.3.1.1. This functionality satisfies requirements ER 5-7 and was verified using data from both the live CHP CAD XML feed and D12 activity log data pushed from a postgresql database to the ESB. Both sources and their associated transformers were able to trigger event notifications that activated the monitor and generated CARTESIUSEvents objects that drove further processing.

The requirements specify that CARTESIUS be able to obtain current demand estimates from an external model. The project team had hoped that the Testbed's path flow estimator would be available for their purpose, but this software was not available on time. Instead, the system was structured to use DAO to obtain demand estimates from static sources pushed to the database. This approach effectively deferred the integration of the demand model to a later date. The default implementation is to use the results of a regional planning model or the demand inputs to a site-specific traffic microsimulation model to obtain demand estimates (this is the approach taken in the evaluation described in Chapter 8.) The result is that the system has a limited implementation of requirements ER 13-15. Since there is no live demand model, the ability to request a new demand estimate was not included so ER 16 was not satisfied.

Like the demand requirements, CARTESIUS needs capacity estimates to drive its core models. The original implementation used commonly accepted engineering factors to estimate capacities on the basis of facility type, number of lanes, and prevailing conditions. The requirements for the re-implementation suggested that more data-driven capacity estimates might be obtained from external models but the re-implementation current relies upon the same internal estimates of the original. As a result, ER 17 was not satisfied.

Similarly, the idea that intersection capacities might be highly depended upon external control systems led to a suggested requirement that the CARTESIUS be structured to query those systems for capacity estimates. At the time of development, however, no systems capable of providing such estimates existed so the re-implementation used the original CARTESIUS approach of estimating intersection capacities using Akcelic's intersection delay model. Thus, ER 18 was not satisfied.

Finally, the CARTESIUS is general enough that is can access historical data about prevailing system performance, demand, and events as easily as it can access current data. The system is not currently making use of this capability, but the capacity exists and thus ER-19 is partially satisfied.

5.3.3.3 Logical Database Requirements

As noted earlier, the core CARTESIUS algorithms were designed to the DAO design pattern to shield the implementation from later changes in the back-end data sources. Development and testing of the system was performed using a Postgresql database backend (Postgresql Global Development Group, 2008). Postgresql is one of the leading open source relational database management systems. Postgresql is known for its speed with complex data structures and also boasts a spatial extension that implements advanced geographical information system (GIS) functionality within the database. The developers experience with Postgresql and its broad acceptance in the industry was used to justify its use for CARTESIUS.

Software access to the database from CARTESIUS and other components of the ESB were coded using the Hibernate Relational Persistence system for Java (King, 2004). Hibernate offers a well-support system for object-relational mapping whereby storage of software objects to various backend database systems is configured automatically from plain Java source code. The Hibernate system greatly simplifies object persistence by handling the bulk of transactional issues that typically would have to be coded by hand. Hibernate also natural integrates with Spring and the Mule ESB and is generally the *de facto* standard for object persistence when using these systems.

The combined use of the off-the-shelf database and persistence tools coupled with the data models specified in the above sections allow for complete persistence of all data and activity carried out by CARTESIUS. As such, all database requirements from TO-5324 (DR 1-8) have been met in the reimplementation

5.3.4 Security, Compatibility, and Reliability

For security, CARTESIUS relies on the Spring security features that come bundled with the Mule ESB. Spring security provides the ability to require:

- user authentication for any access to live system data.,
- layered security using a role-based system that limits access to sensitive data or control functions to sub classes of users, and
- full logging of all actions taken to support security auditing.

The re-implementation, however, is not backward compatible with the original CARTESIUS version. Providing such backward compatibility would have significantly limited the re-design to legacy technologies. Given that CARTESIUS has no current deployments, maintaining such compatibility did not seem necessary.

5.4 Verification and Validation of Integrated System

We integrated the complete CARTESIUS system, consisting of the components described above, using the Mule ESB for message transformation and routing. The resulting system is driven by event and sensor data passed from any sources for which connectors to the ESB have been created. The results of processing are sent to any destinations that have connected to the ESB and requested the results. The current implementation of the integrated system has only been validated using simulation connected to the ESB and CTNET. The details of this validation are the subject of Chapter 8. Prior to considering those issues, however, we first discuss how CTNET was connected to the CARTESIUS system and then how the simulation-based evaluation environment was developed.

Chapter 6
Accessing CTNET

6.1 Background

The integration strategy outlined in Chapter 4 requires that CARTESIUS be capable of acting as a client to CTNET. Because CTNET development proceeds independently from that of CARTESIUS, the project team made a design decision to not alter CTNET and instead to have CARTESIUS act as a CTNET client to interact with the traffic signal subsystem. To achieve this, we needed a new communications library for CARTESIUS that could communicate directly with CTNET using its own communications APIs. Because CARTESIUS was developed in Java, this meant we needed Java libraries that implemented the core CTNET protocols. These libraries were, in turn, used to develop specific Data Access Object (DAO) implementations in CARTESIUS to allow it and its upstream data providers to access traffic signal data from CTNET. The following sections discuss the design of these libraries.

6.2 Jab3418e

As discussed in section 2.4, CTNET uses an extended version of the AB3418 standard protocol for communication with field elements, which are forwarded to CTNET clients when they are connected. We developed the Java AB3418e library (Jab3418e) to allow Java applications in general and CARTESIUS in particular to send and receive these messages. The two use-cases for the library were (1) to allow CARTESIUS to interact with the CTNET CommServer to receive AB3418e messages and (2) to allow simulation developers to implement the AB3418e protocol to create simulated field devices with which CTNET can communicate. Note that these use cases did not require the ability to send and receive messages over serial connections, as is required in the original specification. This is because the library will only be used over TCP/IP connections. Modifying the library to support such connections is feasible, but was beyond the scope and needs of this project.

6.2.1 Design

The design of the Jab3418e library parallels that of the original c++ CTNET communications software provided by Caltrans (Caltrans, 2003). This design uses an abstract base class (NetMsg) to implement a common generic interface to all possible message types in the AB3418e protocol. Classes are then derived from NetMsg for each message type, overriding the methods that the particular message type implements and returning false if they are unimplemented. Using this design, programmers simply use the abstract NetMsg class to query received messages for particular data. If the data is not provided by the message, it is ignored. This is a classic approach to message polymorphism in c++.

Unlike c++, Java is a dynamic (interpreted) language that is also introspective (the program can ask the interpreter what class a particular object is). These features allow for a different approach to message polymorphism that allows the client to discover the particular message type on the fly and therefore operate with stricter typing on the client side. This means that the client-side programmer can filter out unwanted message types and operate using only message relevant to its purpose. From the design perspective, this means that unlike the c++ implementation, the top-level NetMsg class does not implement methods for the entire AB3418e message set. Instead, we created sub-class hierarchy for organizing messages into groups that have common data subsets. This approach reduces the size of the code and improves maintainability without the loss of flexibility provided by the original C++ library.

6.2.2 Implementation

Jab3418e was implemented using Java version 1.5 due to make use Java's generics features added in 1.5. The following software classes were created for the implementation.

- **Ab3418eData**: The top-level abstract base class for representing data to be passed in Ab3418e messages. This abstract class contains helper methods common to the Java implementation of Ab3418e messages. In particular, it provides a set of (static) methods for packing and unpacking Ab3418e data frames with the data stored in derived classes. These methods seek to ease rapid implementation of message types by using reflection to automatically pack and unpack derived class data fields as long as they conform to the naming standard. Derived data classes not conforming to the naming standard, or that can't be packed simply as a set of ordered bytes containing the data of the member fields have to override the packAb3418eFrame and unpackAb3418eFrame methods.

- **AddressInfo**: This abstract class extends Ab3418eData to add addressing information.
- **NetMsg**: NetMsg adds a message type to the AddressInfo data, which makes this data a unique message.
- **GetShortStatus, GetSetErrorMessage, GetStatus8, GetLongStatus8,** and **GetControllerTimingData**: are all interfaces containing Request, Error, and Response member classes implementing the appropriate accessor methods for the relevant data each sub-message type. Thus, a client would create a GetStatus8.Request object to send a GetStatus8Request message to a field element. It would expect to receive a GetStatus8.Response or GetStatu8.Error message as a reply from the field element.

The above class hierarchy represents the application-layer messages that are passed under the AB3418e protocol. Actually sending and receiving such messages over AB3418e requires an additional helper class that can encode and decode those messages into data frames for transmission over the communications links. We created the Ab3418Message<T> class for this purpose:

- **Ab3418Message<T extends NetMsg>**:
  This class is a Java implementation of the Ab3418eMessage type that encapsulates all messages used for communication between FieldElements and TMC applications. The class is implemented as a wrapper that can generate and decode AB3418E data frames for transmitting any data that extends the NetMsg data class.

```
GetLongStatus8.Response msg = new GetLongStatus8.Response( masterId, localId, 1 );
msg.getData().setStatus( 0 );
msg.getData().setPattern( 0 );
msg.getData().setActivePhase( 1 );
updateData( msg, pds );  // a function to update the system detector data
int frame[] = Ab3418eMessage.allocateFrame();

int length = msg.packAb3418eFrame( frame );

if ( ( length = Ab3418eMessage.stuffFrameFCS( frame, length ) ) ==
                      Ab3418eMessage.ERROR_CODE )
{
       throw new RuntimeException( "Failed stuffing frame" );
}
String bstr = "";
for ( int i = 0; i < length; ++i )
{
       if ( frame[ i ] != 0 )
       bstr += " " + Integer.toHexString( frame[ i ] & 0xff );
}
logger.info( "Sending data[" + bstr + "] to CommServer" );
try
{
       DataOutputStream ods = new DataOutputStream(clientSocket.getOutputStream());
       // DataOutputStream can only write bytes, copy over
       for ( int i = 0; i < length; ++i )
       {
              ods.write( frame[ i ] & 0xff );
       }
}
```

The above listing shows example code for using the library to generate and send an AB3418e message over a TCP/IP connection---in this case a simulated response to a GetLongStatus8 request that contains the active phase at a signal and the most recent system detector data.

6.2.3 Verification

To verify the Jab3418e’s implementation, we performed extensive testing to confirm that the library could both successfully send and receive messages over the AB3418e protocol. To test the library, we used the CTNET CommServer as a source for confirming that generated messages provide the approach AB3418 message framing. The library was used to construct a simulated field element (master) that could be addressed using TCP/IP (an internet hostname and socket). The simulated element was designed to send a predictable pattern of data for the GetStatus8 heartbeat messages expected by the CTNET CommServer.

Testing proceeded as follows:

1. The CommServer was configured to connect to this test element.
2. A CTNET client logged into the CommServer and was configured to connect to the simulated field element
3. The CTNET client interface was used to check that the sequence known to be sent by the simulated field element arrived.

To confirm that the client-side processing provided by the Jab3418e library correctly receives and decodes incoming AB3418e messages, we also connected a test client that uses the JCTNET library (provided below) to connect as a client to the CTNET CommServer to receive the messages from the simulated field element. Since the pattern of data sent by the simulated field element is known, the client side program is able to verify (or disprove) whether the messages received are being properly transmitted through the system. Using this procedure, we were able to confirm that the Jab3418e library successfully implements the GetStatus8, GetLongStatus8, GetControllerTimingData, and SetControllerTimingData message sets, which together satisfy the basic needs for connecting CARTESIUS and CTNET.

6.3 JCTNET

6.3.1 Design

As described in section 2.4, direct communications between a CTNET client and the CTNET CommServer are performed by sending specially formatted tab-delimited strings over a TCP/IP socket connection. Additionally, the client will receive messages via UDP from the CommServer once a connection is established that contain the GetStatus8 messages of field elements specified in the client's configuration. In analyzing the CTNET source code, we determined that CTNET follows the model-view-controller paradigm (Burbeck, 1992). In this design pattern, the server maintains a *model*---a collection of data that represents the state of the system (in this case, the field elements it is connected to). The server exposes that data to clients by means of a *controller*---essentially a set of specific commands that can query or modify that model. The client presents a view of that data to the user by using the controller to obtain the state of the system. The view also offers the user the ability to send commands to the controller to operate on the data. Thus, the CTNET CommServer implements the model and controller features, and the CTNET Client provides the view.

In this conceptualization, the purpose of JCTNET is to create a low-level replacement of the view that can be used in Java applications in general, and the CARTESIUS Java implementation in particular. In our analysis, we also noted that most of the direct interactions between the CTNET client and server can be grouped as atomic, logical interactions. Generally, the client sends a message to get or set data over TCP/IP using the controller's API, the server process that message, queries or alters its underlying model, and returns a response or an error. As such, we designed a system based upon threaded socket workers that are spawned on the client side to complete atomic actions. For instance, the login process requires the client to send a LoginCommand message (with username, password, CTNET version, and county name parameters) to the CommServer and to wait for a response message (which means success) or an error message (which includes a reason for failure, such as a bad password). We termed each such atomic process a functional component.

To manage the series of independent functional components, we designed a system for processing incoming messages from the CommServer using a ChainOfResponsibility design pattern (Gamma et. al., 1995). In this design, the programmer specifies a list of message processors that should be called (in order) when a message is received. The list is traversed, with each processor getting the opportunity to process the message, until a processor "consumes" the message (by returning a particular response), at which point the processing of that message is terminated. This design differs somewhat from the original CTNET design that passes messages to all components, which individually process them if they are relevant. The original design is more of a decentralized design that offers flexibility. The JCTNET design, on the other hand, places more emphasis on centralized control (and configuration) of where messages are sent, which could potentially reduce processing overhead as the system grows, while also offering more fine-grained control of potentially sensitive data.

6.3.2 Implementation

The JCTNET library was implemented in Java (requiring version 1.5 or later because of the use of generics). The following details the major components of this library.

- **MessageProcessor**: A MessageProcessor can participate interpretation of CtnetClientCommandMessage objects. Generally, The CTNET Client/Server architecture is built around simple message packets consisting of a command type and a tab separated list of parameters. The CTNET Client, in this case, acts the controller, sending requests to the server, which interprets them, accesses the CTNET Server logic to obtain a result, and returns that result in one or more replies that are encoded in a manner consistent with the CtnetClientCommandMessage class. The CTNET client must read, decode, and act upon these messages to provide core services to the user. The reading step is handled by the CtnetTcpClientWorker. The decoding step is handled by the CtnetClientCommandMessage class hierarchy. The acting step is handled using a hybrid of the publish/subscribe pattern and the chain of command/control pattern. Essentially, CTNET client functions are encapsulated in a set of classes implementing the MessageProcessor interface signifying that they can register with the CtnetTcpClientWorker to be notified when a new CtnetClientCommandMessage is received.
- **CtnetSessionManager**: a CtnetClientComponent that handles login functions for a client over a given TCP connection to a Server.
- **Ab3418eUdpClientWorker**: A chain of responsibility worker for (AB3418e) messages received on the UDP client port.
- **ChainOfResponsibilitySocketWorker< M >**: A socket worker that processes messages of type M using a simplified Chain of Responsibility pattern.
- **CtnetFunctionalComponent**: is an abstract base class that encapsulates all common functions associated with a CTNET client-initiated client/server logical interaction. In the CTNET model, the client connects to the server listening on a TCP/IP port and establishes a TCP link for socket communications between the processes. All client-initiated actions begin with the client sending messages to the server over this pipe and

(possibly) waiting for a response on the TCP port (or possibly on other communications ports as well). The functional components implemented the following:

  - **LoginManager**: sends and receives messages related to authenticating the client with the CTNET CommServer. This class provides methods for initiating and terminating logins and handling any disconnections from the server that may require re-authentication.
  - **FieldElementManager**: The FieldElementManager is responsible for creating and maintaining a set of local FieldElement objects representing elements managed by the connected CTNET CommServer. This task is accomplished by querying the CommServer for a list of field elements, creating a collection of objects representing them, processing Ab3418e messages received on the UDP port, and updating the local FieldElement object representations to reflect the state reported by the most recent measurement. This component can also send directives to field elements via the CommServer, including commands to update the active timing plan (assuming the user is authorized by the CommServer).

- **CtnetTcpClientWorker**: A TCP socket worker for processing CTNET messages between the client and the CommServer. This worker processes messages by using a very simple version of the chain of responsibility (CoR) pattern (Gamma et. al., 1995). Basically, we form a chain of message processors from objects that register to process messages. When a message is received, it is passed to each processor in the list (in the order that the registered) until one of them is able to process (consume) the message.
- **CtnetClientBasicImpl**: A default implementation of the CtnetClient interface. This implementation is intended to be CtnetSessionManager reference implementation, but should suffice for most purposes. It offers the capability to perform any implemented functions from the command line.
- **CtnetClientCommandMessage**: The top-level abstract class for encapsulating CTNET Client/Server messages. CTNET messages are fixed-length packets with the first byte identifying the message type and the remaining bytes dedicated to a tab-delimited string of parameters for that message. This class implements code that can be used to process all message types. Subclasses of CtnetClientCommandMessage need only implement

get/set methods for the interpreting individual parameters (stored as strings in the parameters Vector).

6.3.3 Verification

As with the Jab3418e library, we tested the JCTNET library to confirm that it could both successfully send and receive messages to and from the CommServer. Since the JCTNET library must also handle AB3418e messages received from the CommServer using UDP, the testing protocol was structured atop the Jab3418e test suite, with the simulated field element sending the same predictable pattern to the CommServer as described in section 6.2.3. The library was used to construct a JCTNET command-line client (**CtnetClientBasicImpl**). The test protocol then proceeded to have the JCTNET client connect and interact with the CommServer (instead of using the official CTNET Client) to evaluate that core CTNET protocol functions worked properly. The testing configuration was carried out as follows:

1. The CommServer was configured to connect to the Jab3418e test element described in section 6.2.3.
2. The JCTNET client logged into the CommServer using an existing user name and password combination entered on the CommServer.
3. The JCTNET client then requests the list of all field elements connected to the CommServer
4. The JCTNET client then processes all UDP messages forward by the CTNET CommServer for those connected field elements and compares the messages received to the expected pattern

The results of JCTNET testing were used to verify the core functions of the library including:

- The command line client successfully logged in and logged out from the CommServer
- The client obtained an accurate and complete listing of all field elements in the active CommServer configuration.

- The data streamed from the CommServer for connected field elements were redirected to a field and confirmed to match known data set by the testing element.
- The client sent messages via the CommServer to update the timing plan of the testing element, which was confirmed to match plan sent by the client.

Chapter 7

Creating the Evaluation Environment

7.1 Background

The CARTESIUS/CTNET projects (TOs 5313, 5324, and 6324) were intended to perform a live field operational test of the combination of CARTESIUS and CTNET providing advisory information for incidents that occurred on the monitored corridor. The main purpose of the FOT was to evaluate the CTNET/CARTESIUS combination on a number of axes. We discuss each of these below.

The first axis was whether CTNET could be used effectively as a subsystem for (a) obtaining data from the field and (b) updating controller settings based upon CARTESIUS outputs. Because of the ATMS Testbed's existing agreements with Caltrans and the City of Irvine, as well as the existence of Testbed-owned controllers in the Irvine corridor, the use of the Irvine corridor (refer to Figure 8.1 in the next chapter) was an original part of the proposal and was agreed upon early in the course of the project.

Once development work started, however, problems gradually arose with the use of this corridor and with the subsequent development of the field operational test that eventually led to a focus on a simulation-based evaluation. Each is summarized below along with the actions taken to resolve them through either technical means or by changing the scope of the project

7.1.1 City of Irvine Controllers unavailable

The Testbed's agreements with the City of Irvine proved unable to support the type of FOT proposed. One of the Testbed's original promises---that of breaking down jurisdictional boundaries---proved difficult to achieve in practice. Other District 12 sites were considered, such as Beach Boulevard (CA 39), but were found to lack a well-defined corridor that would allow the evaluation of freeway/arterial operations. At this point, the possibility of performing a closed-loop FOT using field controllers was dropped from the project scope.

7.1.2 Testbed shadow 2070s incompatible

A secondary approach was proposed to bypass the jurisdictional problems by using the collection of “shadow” 2070 controllers owned by the Testbed and connected in read-only fashion to the actual field hardware. In this process, CTNET would connect to these controllers to obtain the state of the system (signal phasing, timing plans, and signal data), which would in turn be forwarded to the CARTESIUS Client(s) in order for them to develop a model of the system.

In pursuing this approach, however, it was discovered that the field hardware was incompatible with the TSCP program required by CTNET. Instead, the Testbed’s 2070 controllers were limited to running a customized program called the Input Acquisition Software (IAS) that was developed on the Testbed to stream data from the field controllers into the Testbed laboratories. IAS provides read-only data that is limited to obtaining the presence and system detector outputs because the hardware configuration in the cabinet uses a custom wiring harness to cross-connect the detector wiring into the 2070 controller using a 2070-2A or 2070-2N Field I/O unit. Because the Testbed’s 2070s have no direct access to the actual controller and its internal logic, they are unable to obtain any information about prevailing phasing. As such, the IAS program does not provide this functionality.

Several options to overcome these shortcomings were considered, including adding an additional custom harness to read the outputs to the signal heads directly and infer some phasing information from that. However, a combination of jurisdictional constraints (the anticipated difficultly getting City of Irvine to allow an additional harness) and technical issues (doubts that direct signal head readings would provide sufficient information to mimic TSCP requirements) led to a further downgrading of the plans. In particular, the team realized that the inability to know what signal plans were active during an observed incident would render CARTESIUS incapable of making any meaningful inference about the state of the arterial subsystem. At this point, the team decided to establish the link in order to minimally obtain system detector data from the field, which could still offer some information about the state of one major arterial in the corridor.

```
[INFO] IASData: 1211570080 : 1211570080000 : Fri May 23 12:14:40 PDT
[INFO]  1122222-      --334444      --556666      66--7788      88------
[INFO]  1356712-      --134512      --135671      23--1345      12------
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
[INFO]  00000000[  0] 00000000[  0] 00000000[  0] 00000000[  0] 1000000[
```

7.1.3 IAS bridge

7.1.3.1 Development and Testing

Since the main point of the original research was the integration of CTNET and CARTESIUS, the research team remained committed to providing AB3418E messages to the CTNET CommServer about the status of real-world field elements. The team chose a compromise by developing an AB3418e bridge between the traffic signal feeds provided by the IAS software and the CTNET CommServer. The IAS feeds on the Testbed come in over the Testbed Intertie over a fiber-optic link to the Institute of Transportation Studies' server room and from there is routed to the Testbed LAN. The IAS program is a socket-based client/server program whereby the client connects to the IAS program using a predefined IP address and socket. Once the connection is established, IAS begins streaming data packets to the client that contain actuations read from all detector inputs at a frequency of 10hz. Typical output from IAS is shown in the listing above. The first line shows the time stamp of the data. The second line shows the phase (approach) associated with each column. The third shows the detector number (for the phase) associated with the column. And the next 10 columns show presence (0=false or 1=true) for each 1/10th of a second during the 1-second sample period. In the example, the first detector associated with phase 8 on the controller (as defined by the wiring diagram) is showing presence for the entire second of May 23, 12:14:40 PDT.

The Testbed provides a CORBA event service to allow multiple clients to receive the data streams provided by IAS. The service simply passes the packet stream to any and all clients that connect. This layer of middleware provides better fault tolerance and scalability. The

Table 7.1 Connection status of Testbed shadow controllers in the City of Irvine.

IP	Intersection	Status
192.168.30.101	Jeffrey @ Quail Creek	100% packet loss
192.168.30.102	Alton @ Jeffrey	0% packet loss
192.168.30.103	Alton @ Royal Oak	0% packet loss
192.168.30.104	Alton @ Valley Oak	0% packet loss
192.168.30.105	Alton @ Sand Canyon	0% packet loss
192.168.30.106	Alton @ Hospital	0% packet loss
192.168.30.107	Alton @ Laguna Canyon	0% packet loss
192.168.30.108	Alton @ Jenner	0% packet loss
192.168.30.109	Alton @ Telemetry	0% packet loss
192.168.30.110	Alton @ Banting	100% packet loss
192.168.30.111	Alton @ Pacifica	0% packet loss
192.168.30.112	Alton @ ICD	100% packet loss
192.168.30.113	Alton @ Gateway	0% packet loss
192.168.30.114	ICD @ Barranca	100% packet loss
192.168.30.115	ICD @ Gateway	100% packet loss
192.168.30.116	ICD @ Spectrum	0% packet loss
192.168.30.117	ICD @ Pacifica	0% packet loss
192.168.30.125	Spare	100% packet loss

implications for this project were simply that for CTNET to access field data. We had to implement a "software bridge" that transformed the data feed from a stream of IAS packets into a stream of Ab3418e packets that are accessible from a server implementing the AB3418e API.

A custom server program was written in Java using the Jab3418e library to act as a mock controller implementing the Ab3418e API. The program is designed to connect to a single CORBA event channel representing a single Testbed 2070 shadow controller. Once the connection is established, the program opens a TCP/IP socket for receiving a connection from a CTNET CommServer using the Ab3418e. When a CommServer connects to this socket, the server streams messages to the CommServer using Ab3418e messages as discussed in section 2.4

7.1.3.2 Problems and Implications

There were different problems with different sections not working which were related to the build out of the City of Irvine network infrastructure. At the close of the project, connections to most intersections were up but still exhibited some instability. The final status is summarized in Table 7.1 and shows the lingering unreliability of the links provided by the Testbed Intertie.

The limited coverage, the read-only nature of the data, and the relative instability of the connections meant data obtained via the bridge, though live, offered limited use for evaluation of CARTESIUS. As a result, we focused our efforts on developing a simulation-based platform for evaluating the CARTESIUS/CTNET system.

7.2 Improvements to Paramics

The problem at this stage was to provide a simulation system that CTNET could connect to for reading data including sensor measurements, active signal phasing, and available timing patterns as well as writing new timing plans to the controller under the direction of CTNET clients which could now include CARTESIUS clients using the JCTNET libraries discussed in Section 6.3. The Testbed has long maintained an advanced version of the Paramics traffic microsimulation software system (Cameron and Duncan, 1996) including a complete model of the Testbed network (partially developed as part of the related PATH TO-5313) and a collection of custom plugins for modeling Intelligent Transportation System (ITS) capabilities---most notably an actuated controller plug-in from earlier research on TO-5324 providing the bulk of features required to mock a 2070 controller. Consequently, this model was selected to provide the simulation evaluation platform.

7.2.1 Paramics Plug-in Architecture

One positive result of the work on the IAS bridge software was the development of the Java-based Ab3418e server code that allowed CTNET to connect to a Java program acting as a mock 2070 controller implementing the Ab3418e protocol. This code offered a great starting point for developing an Ab3418e interface to the Paramics model, but additional work was necessary to allow the Java code to work with the Paramics microsimulation.

The first requirement was developing the ability to execute Java code in a Paramics plug-in. Paramics is written in C and its plug-in architecture offers APIs in C. The Testbed's plug-in library has extended the C interface and offers a plug-in that wraps the C API in an object-oriented interface using C++ that is automatically generated from the C header files that define the API. This project added to this library by developing a plug-in that uses the C++ API to map API function calls to Java classes. This Paramics Java plug-in allows developers to write Java plugins for Paramics without having to use C or C++.

7.2.2 Ab3418e Paramics Plug-in

With the ability to write Java plugins directly, we were then able to reuse the Java Ab3418e server code we developed for the IAS bridge to implement an AB3418e plug-in in Paramics. This plug-in creates a TCP/IP socket for each of a user-specified set of signals that are modeled with the Testbed's actuated controller plug-in. Once the Paramics simulation is started with the AB3418e plug-in, a CTNET CommServer can connect to the simulated signals to receive and send a fundamental subset of Ab3418e messages allowing CTNET clients to read sensor data and phasing status and write new timing plans to the controllers.

A notable limitation of the using the AB3418e plug-in is that it requires that the Paramics model be run such that the simulation clock runs at or slower than the real-world clock. This is a reasonable restriction, however, given CTNET and CARTESIUS use cases, which will typically involve operators using the systems to provide active monitoring and control. Such interactive use precludes the requirement for faster than real-time operation that is often needed for complex simulation models.

We verified the functioning of the Ab3418e plug-in using a series of structured tests to evaluate weather the AB3418e communications were successfully transmitting data between the simulation an CTNET clients. We performed three basic verifications: one for sensor readings, one for active phasing readings, and one to see if CTNET clients could successfully update the timing plans being used in the simulation.

To verify the sensor readings, the Paramics simulation was configured to dump sensor readings to a file. Similarly, CTNET was configured to log sensor readings for specific intersections. The simulation was run for a 30 minute sample with the CTNET CommServer connected. Then, the two sample files were converted and compared and found to be in agreement for the sample period. Since CTNET only logs system detector readings, we also performed a visual verification by zooming the Simulation GUI into a intersection , noting when the stop line presence detectors were actuated by simulated vehicles, and confirming via the CTNET Client interface that a call was registered on the appropriate phase at the appropriate time. These tests were repeated for several intersections to the point that we were satisfied that Ab3418e data stream was functioning properly for sensor reads on the CTNET Client side.

To verify that the connection was providing accurate data on active signal phasing in the simulation to CTNET clients, we were also limited to a visual inspection using the Simulation GUI and the CTNET client interface. Since the simulation can run in a single-stepping mode, however, it was easy to confirm that the phasing shown in the simulation matched that shown in the CTNET client. As above, we repeated this verification process on several intersections under multiple simulation conditions until we were reasonably convinced that the plugins and communications infrastructure were operating correctly.

Finally, to verify that the connection was allowing CTNET clients to update timing plans in the simulation, we used the CTNET client interface to make changes to the prevailing plan and set the updates via the CommServer to the simulation. Using a combination of debugging statements in the Ab3418e plug-in and further visual inspections using the Paramics GUI, we again established that the data was being sent properly between the clients and servers and that the timing plans were correctly updated in the simulation.

7.2.3 ATMS Testbed Paramics Plug-in

Since CARTESIUS also requires access to freeway sensor and ramp metering functions, we also adapted a set of existing Paramics plugins that provide freeway loop data as well as read/write capabilities to ramp meters and CMS modeled in the Simulation. These plugins use CORBA interfaces that were developed as part of the TRICEPS system that was used in the original CARTESIUS evaluation (Logi and Ritchie, 1997).

7.3 The Complete Evaluation System

With the Paramics simulation suitably extended to interface with the middleware systems of interest we were able to finally integrate all the systems into a function evaluation system, albeit one with capabilities that fall short of the original FOT evaluation originally planned for this project. Figure 7.1 shows the complete evaluation environment established for this research. CARTESIUS lies at the heart of this system. Its role in this scenario is to monitor and manage disruptions in the system. As discussed in Section 5.3.2, CARTESIUS is configured to act as a back-end process that can provide information to other clients that may serve as the primary operator interface. For evaluation purposes, the back-end process was configured to implement any recommendations made by the system automatically, rather than requiring operator input. Furthermore, the multi-agent version of the system was not evaluated because its completion was deferred for later development (see Section 5.3.3.1).

In this evaluation configuration, the simulation is connected to various components in the SOA via the CTNET and TRICEPS plugins discussed above. To bring the data from the simulation into the SOA, ESB connectors were developed using the Jab3418e client library and common CORBA interfaces to connect with the simulation plugins (via Ab3418e and the TRICEPS CORBA interfaces respectively), receive streaming sensor data, and push that data into a Postgresql database. This operation is similar to how the Testbed is currently configured to interact with the Caltrans District 12 ATMS system. These sensor data are exposed to other components in the architecture using DAO and other JDBC connectors (where appropriate).

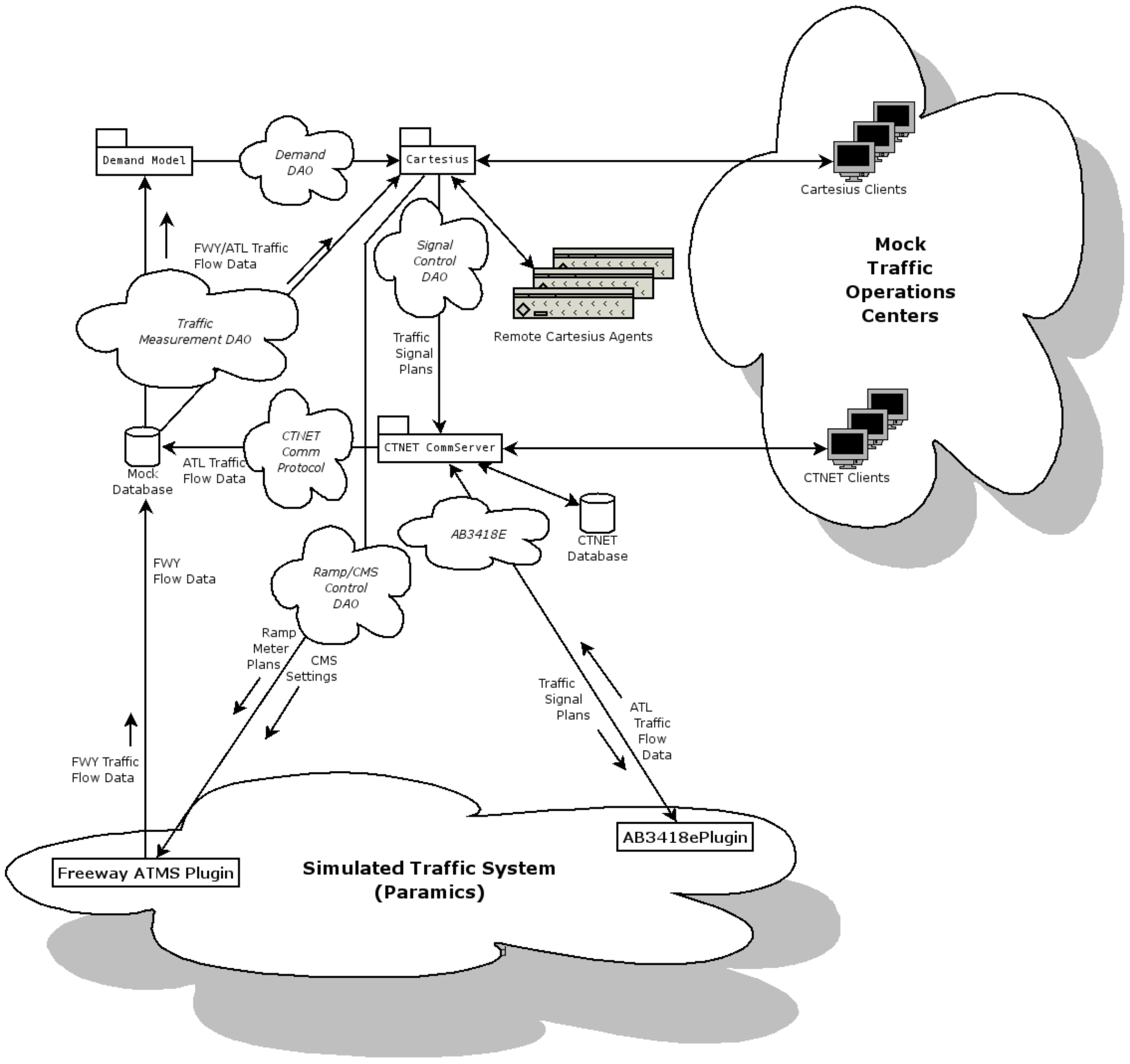

Figure 7.1 Structure of the CARTESIUS/CTNET evaulation environment

The notable exception to the above information flows is when incidents occur in the simulation. In this case, the event is both logged in the database and passed directly to the ESB. The ESB is configured to route these events to the CARTESIUS monitor (see section 5.3.1.1), which initiates CARTESIUS analysis and response procedures. The analysis and response procedures, in turn, use their configured DAOs to access sensor data, active timing plans, and prevailing demand estimates from the database in order to build a system state snapshot.

Note that while it was a goal to integrate the CARTESIUS system with a dynamic path estimator under development on the Testbed, the latter project proved too challenging to incorporate under the scope of this project. As such, demand estimates for CARTESIUS were limited to using path demand inputs derived from the simulation model by loading the network without disruptions and saving the equilibrium paths used and the number of vehicles using those paths. Such perfect demand estimates will not be available in practice and the results of this research should be evaluated with this in mind.

Once the analysis and response procedures execute their models, search for solutions, and identify solutions, CARTESIUS response actions were configured to push controller actions to the ESB, and CORBA connectors were configured to forward these actions to the simulation either using the TRICEPS connections (in the case of ramp meter and changeable message sign actions) or the CTNET CommServer (in the case of signal timing changes). The performance of the entire system could then be evaluated using the capabilities of the Paramics simulation.

Chapter 8

Evaluation of the CARTESIUS/CTNET system

8.1 Study Network

CARTESIUS is a data intensive system. It requires a knowledge base that includes a complete representation of the network under management in a somewhat non-standard format that represents the network as a collection of network objects that cluster related facilities into objects that can be used in the diagnostic process that is unique to CARTESIUS. In addition to this network representation, CARTESIUS needs to map the available sensors in the network onto these network objects so that its algorithms can estimate the state of the system. Furthermore, CARTESIUS requires an enumeration of all reasonable paths in the network along with estimates of the historical and prevailing flows on these paths. Finally, CARTESIUS requires a complete set of possible response strategies and the means to translate the impact of these strategies into estimated changes in system capacities and path demands. These intense data and modeling requirements limited the choice of a study network to the already well developed models for the ATMS Testbed.

The compact study network used in this analysis is shown in Figure 8.1 and consists of the triangular region roughly circumscribed by I-5, I-405, and Jeffrey Road in Irvine, CA. The network contains a total of 74 actuated intersection controllers and 25 ramp meters. This network has the advantage that it contains a freeway corridor (I-405) with a well-defined, high-capacity parallel arterial (Alton Blvd, shown in blue) that offers true diversionary potential in the event of significant capacity restrictions on the I-405 freeway. This network also has the advantage that the Testbed provides live data from the actual network for calibration, validation, and general comparison with the simulated network. Finally, because this same network was used in the original CARTESIUS evaluation, we were able to compare the results of the analysis with the results of the original CARTESIUS to obtain a rough validation that the re-implementation is consistent with the original work.

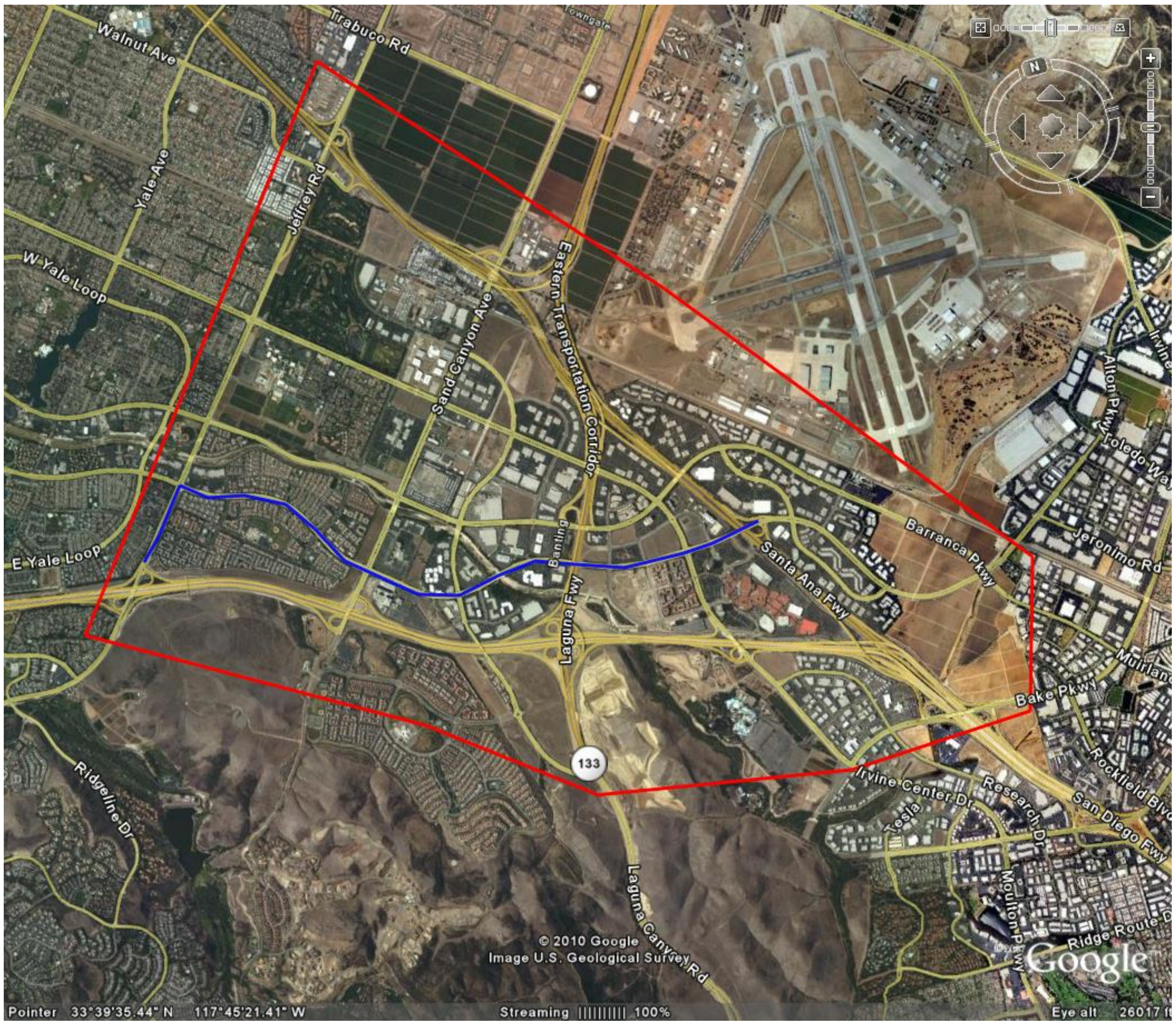

Figure 8.1 The CARTESIUS study network.

8.2 Evaluation

The specification of response strategies is one of the biggest challenges with deploying CARTESIUS. The response algorithm requires that every response strategy can be translated into direct impacts on the modeled system. This requirement is particularly challenging when evaluating the effects of diversion since diversion models are notoriously unreliable in simulation due to the many factors that impact the choice to divert. Because of the intensive data requirements and challenges with modeling a complete range of response strategies in simulation, we limited the evaluation to consider the impacts of incidents of varying severity on the northbound I-405 freeway between I-5 and Jeffrey Road.

We also limited the available response strategies to a limited set of pre-defined signal timing, ramp metering, and diversion scenarios. In addition to the default signal timing plans calibrated for normal demand conditions, several sets of alternative timing patterns were developed *a priori* for two major diversionary paths on northbound Alton Parkway. Both of the paths begin at the Alton off-ramp on I-5 northbound and follow Alton Parkway north to Jeffrey Road. The first re-enters the freeway at Sand Canyon, while the second continues to Jeffrey and re-enters there. For each of these diversionary routes, alternative timing plans were pre-computed that increased the splits on the diversionary movements at each intersection by 10%, 25%, and 50%, with a proportional decrease in the other movements, subject to certain operational constraints. Alternative ramp metering plans were also developed for all ramps that offered minimum flow (max green) and maximum flow (green ball) options to the prevailing time-of-day pattern. Finally, a Paramics routing plugin was used to create a set of diversionary routes associated with pre-defined messages and addressable (for activation) via the Testbed Paramics plugin. Each of these routes and their associated diversion rates were designed to mimic prescriptive CMS diversions with various levels of compliance.

To evaluate the ability of the re-implemented CARTESIUS to identify consistent and complete diversionary strategies in the face of capacity restrictions, we developed a set of incidents in various locations on the NB I-405. This facility was specifically chosen because diversionary routes exist for CARTESIUS to consider. The modeled incidents varied in severity from short-term (5-minute) single lane blockages to long-term (60-minute), multi-lane blockages. A typical incident scenario is shown in Figure 8.2.

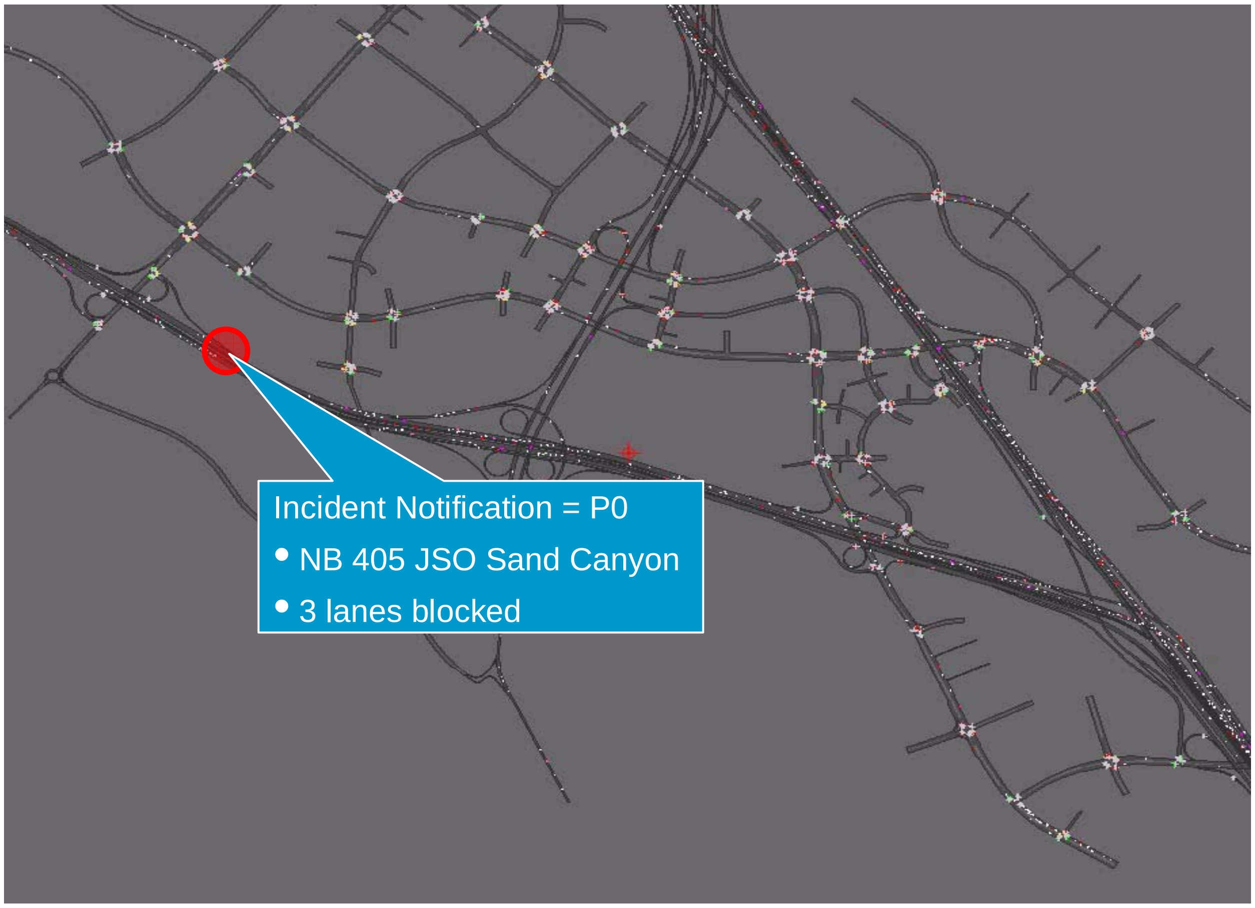

Figure 8.2 Typical severe incident scenario in CARTESIUS evaluation suite.

The Testbed Paramics plugin was used to push incident onset as an Event onto the CARTESIUS ESB, which was in turn passed to the monitor causing CARTESIUS analysis to begin. Where the original CARTESIUS implementation would require operator intervention to diagnose and respond to incidents, the re-implementation can be configured with policies that dictate the default actions that should be taken. In this evaluation, the default policies of CARTESIUS were set to assume all CARTESIUS diagnoses were correct and that the highest-rated response actions should be chosen. The ESB was configured to translate the recommended CARTESIUS response actions (signal timing, ramp metering, and CMS messages) into control directives that were then implemented in the simulation via the Paramics Testbed plugin. To replicate realistic conditions for CARTESIUS, the simulation was throttled to operate no faster than the wall clock, since faster-

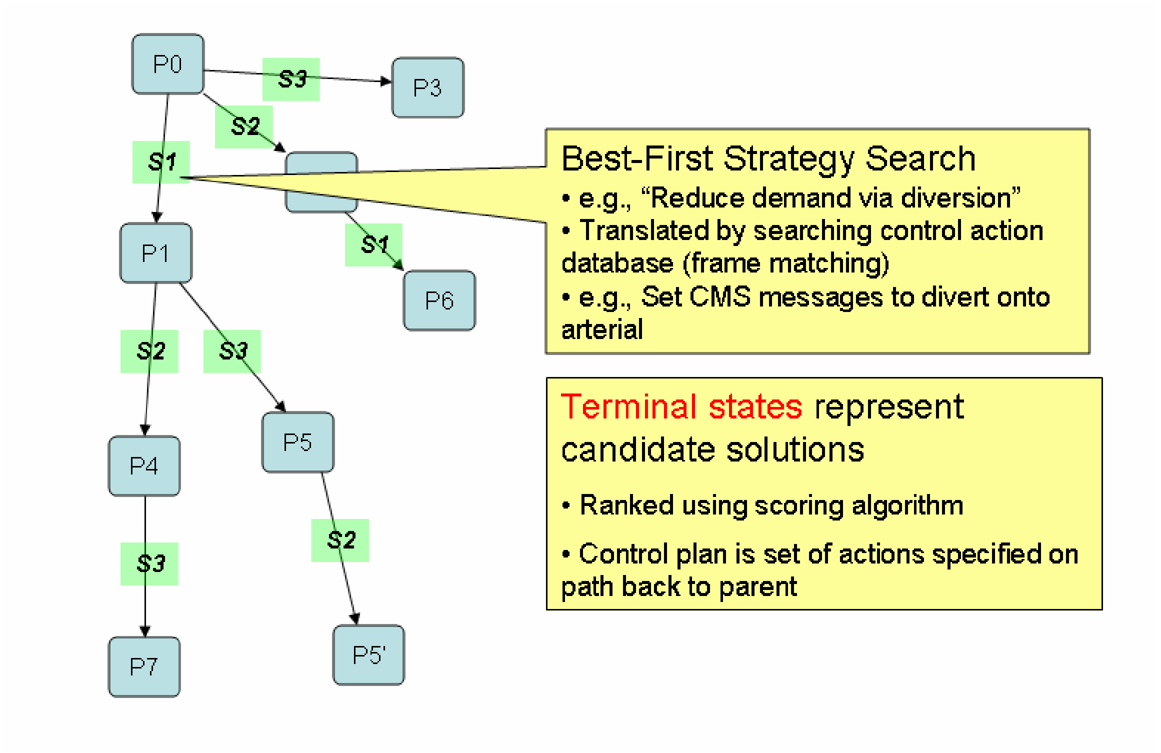
P0
S3
P3
S2
S1
P1
S1
P6
S2
S3
P4
P5
S3
S2
P7
P5'
Best-First Strategy Search
• e.g., "Reduce demand via diversion"
• Translated by searching control action database (frame matching)
• e.g., Set CMS messages to divert onto arterial
Terminal states represent candidate solutions
• Ranked using scoring algorithm
• Control plan is set of actions specified on path back to parent

Figure 8.3 Typical CARTESIUS search tree for severe I-405 incident

than real-time operation might result in significant additional delays that would not actually occur in a real-world application.

A total of 36 scenarios were executed with incidents occurring at three different locations along the northbound I-405 with 12 different severities (1, 2, or 3 lanes) blocked and (10, 25, 40, and 60 minute durations). A typical (idealized) search tree for a severe, long-duration incident on northbound I-405 is shown in Figure 8.3. Here, the response algorithm is presented with a problem (P0) whereby the demand on the northbound 405 exceeds its current capacity due to the incident. CARTESIUS considers its available strategies using its Best-First Strategy Search. The first strategy (S1) represents the strategy of reducing demand on the critical incident section by using diversion. CARTESIUS selects from the available control actions for achieving this diversion---the predefined set of CMS messages implemented in the simulation---and determines that applying the diversion will produce a secondary problem state (P1). Similarly, CARTESIUS considers strategies to increase capacity through signalization (S2) or to decrease demand through metering (S3) producing problem states P2 and P3 respectively.

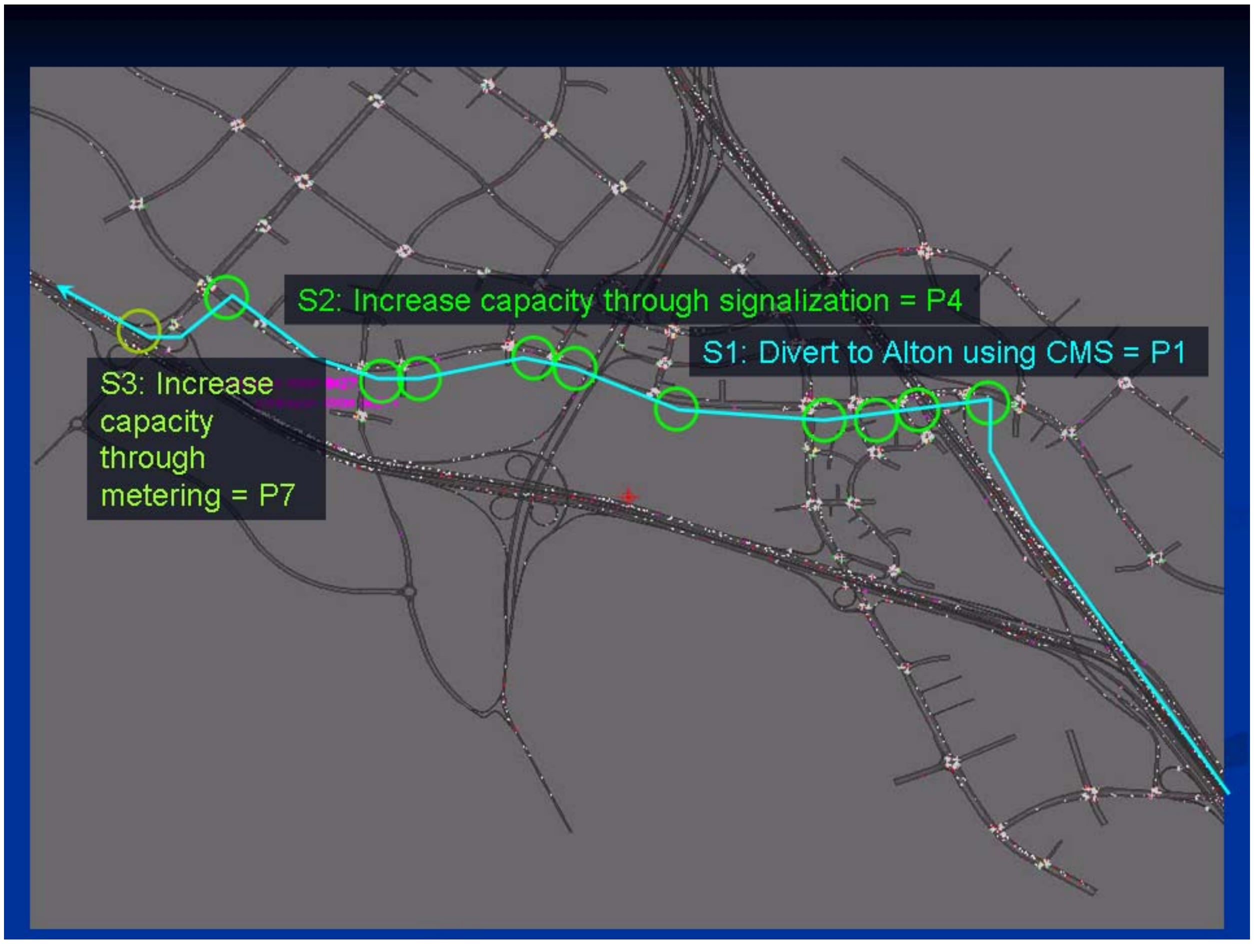
S2: Increase capacity through signalization = P4
S1: Divert to Alton using CMS = P1
S3: Increase capacity through metering = P7

Figure 8.4 Typical response strategy for an incident on I-405 N.

Remaining strategies are considered for each of the secondary problem states and the search expands until no problems (according to the internal models) or no remaining strategies exist. In this case, the diversionary strategy is implemented by a CMS that routes traffic away from the 405 by using Alton Parkway. These leads to the secondary problem (P1) in which the intersections along Alton Parkway become oversaturated due to the increased demand from the diversion. CARTESIUS chooses strategy (S2) to increase capacity of those intersections using signalization. This strategy is translated into timing plans that increase splits along the diversionary route, leading to problem state P4. CARTESIUS continues searching and discovers that further improvements can be achieved using metering (S3) to further reduce demand at the critical section. This is translated into low metering rates (low flow) upstream of the incident resulting in problem state P7. Since CARTESIUS can no longer expand the search tree, P7 is considered a terminal state and its estimated performance is compared to other terminal states (P3, P5, and P6). Since P7 offers the best predicted performance, the collection of control actions associated with P7 is selected and returned to the ESB for implementation. The resulting response is shown in Figure 8.4.

Although the number and diversity of the experiments were relatively restricted due to the complexity of defining a control action database, the results support the conclusion that the CARTESIUS re-implementation performs similarly to the original implementation.

In particular, across all 36 incident scenarios:

- Total delay versus baseline case reduced 12%
- Duration of queuing reduced

It is notable that some of the less severe incidents considered (single lane blockages for short durations) did not always trigger a response from CARTESIUS because the frame matching algorithm was unable to identify a problem from the given sensor data. This does not necessarily indicate a fault with the algorithm since similar behavior was observed in the original implementation and such low severity incidents are unlikely to warrant mitigation involving active controls.

At the operational level, the CARTESIUS performed consistently and was stable. Communications with the simulated traffic signals were reliably provided by the JCTNET/Jab3418e connections to the CTNET CommServer and to the underlying Paramics implementation of an AB3418e-compatible controller.

Chapter 9
Conclusion

The report describes the results of several years of research into the integration of the CTNET traffic signal management system and the CARTESIUS incident management system. The research focused on three major tasks. First we determined that the best way to integrate CTNET and CARTESIUS was to create a pair general purpose interface libraries (Jab3418e and JCTNET) that allow third-party applications to connect with CTNET and perform client actions ranging from receiving sensor and timing data to setting the active signal timing patterns operating in managed field elements.

The designs of these libraries were influenced by the second of the major tasks---the re-implementation of CARTESIUS. This proved to be a significant undertaking, but was justified by a number of significant problems with the original CARTESIUS implementation that made its continued use problematic at best. The new version of CARTESIUS was implemented using the Java platform and a number of off-the-shelf frameworks for communications, persistence, and security. The system satisfies nearly the entire set of core functional requirements established for the re-implementation.

At the close of the project, the new CARTESIUS does not meet the user-interface requirements initially laid out. This is due to difficulties in defining the use cases for how the system might be integrated into already overloaded TMC processes. As a result, the system was re-imagined as a tool for generating data and recommendations that could be integrated into TMC interfaces as they evolve over time through the larger development cycles at Caltrans.

The new implementation also does not meet the external-interface requirements for multi-jurisdictional cooperation. Expanding the original two-agent implementation of CARTESIUS to a general-purpose N-agent system proved to be a challenge that was beyond the constraints of this project. Additional efforts will be necessary to use hooks placed in the current implementation to create a true multi-agent version of CARTESIUS. Furthermore, some external components required for general purpose deployment---such as an operational path demand estimator---are still unavailable for general deployment. As such, CARTESIUS still requires significant preprocessed inputs to become operational for a specific deployment.

These problems notwithstanding, the resulting integrated CARTESIUS/CTNET system still achieves the core goals of the project: to connect CTNET with CARTESIUS. The design has CARTESIUS act as a client to CTNET in order to interact with the traffic signals that it manages, finally providing a viable deployment path for CARTESIUS that has been lacking.

While the initial proposal called for a full-fledged field operational test of combined system, we were unable to create a suitable real-world environment to support this effort. The reasons for this were both technical and institutional. The original proposal called for performing the field operational test in the Caltrans ATMS Testbed to leverage the significant hardware and assets it provides. Ultimately, however, we were unable to achieve closed-loop control with the traffic signals in the City of Irvine because such capabilities currently aren't possible as the Testbed's controllers in the city offer read-only access for institutional reasons which are translated into technical prohibitions that cannot be overcome.

As an alternative to the FOT, we instead created custom plugins for the Paramics microsimulation model to allow CTNET to interact with the system. This proxy real world was then used to evaluate the functional performance of the integrated system in general, and the specific performance of the CARTESIUS re-implementation.

Finally, it has been clear in this project and on other projects related to the Testbed, that Caltrans TMC processes have evolved over time to achieve certain levels of efficiency with available human and technical resources. Altering these processes with new software and analytical tools is not a trivial task. Development of clear processes for achieving such transformations could help speed the integration of new technologies into Caltrans operations. We recommend that future field operational tests go through formal systems engineering steps in order to establish the specific modifications that will be made to active TMC operations and to obtain buy-in from stakeholders at all level prior to the start of the research effort.

References

Alex, Ben and Taylor, Luke (2008). Spring Security Reference Documentation. Accessed on the world wide web on 08/06/2008 at: http://static.springsource.org/spring-security/site/docs/2.0.x/reference/springsecurity.html.

Bell, Michael (2008). *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley & Sons.

Burberk, Steve (1992). Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). Unpublished manuscript. Accessed on the world wide web on 07/04/2007 at: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

Caltrans (1995). Standard Communications Protocol for Traffic Signals in California, Specification and Implementation Requirements. Technical Report. Division of Traffic Operations.

Caltrans (2003). CTNET Field Protocol Specification, AB3418 Extended (AB3418E). Technical Report. Division of Traffic Operations.

Cameron, G. and Duncan, I. (2005). PARAMICS, parallel icroscopic simulation of road traffic. Journal of Supercomputing 10(1), 25-53.

Dave Chappell (2004). *Enterprise Service Bus*, O'Reilly.

Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John M. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gensym (1995). G2: Reference Manual. Version 4.0. Gensym Corporation, Cambridge, MA.

Gudgin, M. Hadley, M., Mendelsohn, N., Moreau J.-J., Nielsen, H.F., Karmarkar, A., and Lafon, Y. (eds.) (2007). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Viewed on the world wide web on 2/3/2008 at: http://www.w3.org/TR/soap12-part1/.

Iteris, Inc (2002). National ITS architecture. Technical report, US Department of Transportation. URL http://itsarch.iteris.com/itsarch/.

King, Gavin, Bauer, Christian, Rydahl, Max Andersen, Bernard, Emmanuel, and Ebersole, Steve (2004). HIBERNATE - Relational Persistence for Idiomatic Java. Accessed on the world wide web on 7/3/2008 at: http://docs.jboss.org/hibernate/core/3.3/reference/en/html/.

Lesser, V. R. and Corkill, D. D. (1981). Functionally Accurate, Cooperative distributed systems. IEE Transactions on Systems, Man, and Cybernetics, SMC-11(1):81-96.

Logi, F. (1999). CARTESIUS: A Cooperative Approach to Real-time Decision Support for Multi-jurisdictional Traffic Congestion Management. PhD thesis, University of California, Irvine.

Logi, F. and Ritchie, S. G. (1997). Verification, validation and evaluation of TCM, a knowledge-based system for traffic congestion management. Technical report, Institute of Transportation Studies, University of Califorina, Irvine.

Pope, Alan (1997). *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*. Addison Wesley.

Postgresql Global Development Group(2008). Postgresql Documentation. Accessed on the world wide web on 08/03/2008 at: http://www.postgresql.org/docs/8.3/static/.

Rindt, C. R., and McNally, M. G. (2006). Field Deployment and Operational Test of an Agent-based, Multi-Jurisdictional Traffic Management System. Final Report for PATH TO 5313. Institute of Transportation Studies, University of California, Irvine.

Rindt, C. R. and McNally M. G. (2008). CARTESIUS and CTNET: Integration and Field Operational Test, Final Report for PATH TO 5324. Institute of Transportation Studies, University of California, Irvine.

Rindt, C. R. and Recker, W. W. (2008). A Measurement-based System for TMC Performance Evaluation: Interim Report and Recommendations. Technical Report, Institute of Transportation Studies, University of California, Irvine.

Sun Microsystems (2007). Core J2EE Patterns - Data Access Object. Accessed on the world wide web on 06/03/2007 at http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

St. Laurent, S., Johnston, J., and Dumbill, Ed. (2001). Programming Web Services with XML-RPC. O'Reilly. First Edition.

Appendix: CARTESIUS Requirements

In this appendix we list all of the requirements specified in the Final Report of PATH TO 5324 for the re-implementation of CARTESIUS so that the reader may refer to them in the text.

<u>Core Functional Requirements</u>

FR1: CARTESIUS must have access to speed and flow information for all parts of the network as well as active control settings for all control devices in the system.

FR2: CARTESIUS should be able to operate using historical data or planned (assumed) control settings.

FR3: CARTESIUS must interface with external systems that can predict the demand on all paths through the managed system as well as the available capacity on all sections of the network.

FR4: CARTESIUS's core functions must use these externally obtained capacity and demand estimates to predict the impact of an event on traffic operations

FR5: CARTESIUS must implement a function that can identify the onset of an FR 5 event in the system

FR6: The system monitoring function must send any identified event to components that have asked for notification, including the event analysis module

FR7: CARTESIUS must identify the cause of the problem in order for CARTESIUS to determine a course of action.

FR8: the outcome of the event characterization process must produce one of two results: it will either identify a cause or it will not

FR9: The CARTESIUS agent must continue processing to diagnose the event if the characterization process successfully determines a cause and its characteristics

FR10: a failure to identify an event's cause should be logged along with the associated inputs so that the system can be analyzed and improved to handle a similar case in the future.

FR11: The event diagnosis function must generate for a given event characterization the final data object that will be used by the event response functions

FR12: The CARTESIUS diagnostic function must consider the possibility of spillback from the neighboring jurisdictions

FR13: CARTESIUS must pass its event diagnosis as a problem characterization to neighboring jurisdictions so that they can take action if necessary (cooperatively or unilaterally)

FR14: CARTESIUS must update its active problem characterization reflecting the new event diagnosis;

FR15: CARTESIUS must identify the local response set by consulting jurisdiction-specific knowledge that defines available control actions that can be used to mitigate the operational problems caused by active events in the system

FR16: CARTESIUS must access a knowledge database of control actions for the jurisdiction.

FR17: The estimated impacts for each control action must be part of the jurisdiction-specific knowledge base.

FR18: CARTESIUS must communicate the feasible local response set to any neighboring jurisdictions to incorporate as constraints into their local decision processes.

FR19: This communication must be performed in a way that does not render the local CARTESIUS unresponsive to the operator.

FR20: CARTESIUS must incorporate any constraints broadcast by remote agents based upon their local response sets

FR21: The CARTESIUS agent must wait for any remote constraints until any synchronization conditions are met, after which it will modify its response search tree

FR22: CARTESIUS must determine the complete set of feasible global responses by exhaustively searching the modified tree until all constraints are incorporated.

FR23: The set of globally feasible responses must be presented to the operator.

FR24: CARTESIUS should support a filtering function that can remove certain response options based upon attributes of the responses

FR25: CARTESIUS should be able to send any potential response to an external simulation evaluator to produce detailed measure of effectiveness (MOE) forecasts

FR26: Once all filtering and evaluation is complete, the final response must be chosen by CARTESIUS.

FR27: CARTESIUS should provide support for the specification of policies that allow the system to automatically choose an action in the absence of operator input.

FR28: CARTESIUS should allow the operator to unilaterally select any globally consistent response via the GUI

FR29: The local choice interface presented for the operator to make a local choice must also make possible for the operator to broadcast suggested response choices to neighboring jurisdictions.

FR30: The local choice interface should receive and display any suggested response choice sent by neighboring agents.

FR31: CARTESIUS must be able to implement any collection of control actions that are available in a given response, but the actual mechanics of setting that control must be performed by external control subsystems, such as CTNET.

FR32: The CARTESIUS core algorithms should be shielded from implementation details as much as possible.

FR33: CARTESIUS could include functionality to simplify model calibration by providing an interface for after-the-fact comparisons of model predictions to observations of the system.

User Interface Requirements

IR 1: The CARTESIUS agent must separate user interface functions from the analytical code

IR 2: The primary user interface must be implemented as a GUI.

IR 3: A heavy client must be provided that mimics the majority original functions of the CARTESIUS prototype.

IR 4: The map display must provide high-level contextual information regarding IR 4 the state of the transportation system.

IR 5: The map interface should display detailed map data that provides useful contextual information to the operator.

IR 6: The map interface should include a network layer that visually displays the underlying active state of the analytical network model by changing the colors of the links in the network based upon user-selected parameters.

IR 7: The user should be able to click on individual components of the network to obtain additional contextual information.

IR 8: The map interface must include a device layer that visually displays the known devices in the infrastructure including loop detectors, traffic signals, ramp meters, and changeable message signs.

IR 9: Clicking individual device icons should provide additional information specific to that device.

IR 10: The analysis interface should provide the operator with a view of the combined problem and solution space being considered by the system.

IR 11: This analysis interface must be consistent with the CARTESIUS approach to representing the problem space as a hierarchical tree where nodes represent problem states and edges represent control actions that lead to new estimated problem states.

IR 12: CARTESIUS should allow the user to interact with analysis interface to view all candidate solutions developed by the system, possibly the ability to filter out candidate solutions based upon particular criteria

IR 13: The analysis interface should also allow the user to activate a particular solution for exploration.

IR 14: Selection of a problem node in the analysis pane should reflect the map display.

IR 15: The effects should be reflected in the network display on the map by changing the underlying network and device display datasets used by the map display.

IR 16: The display of the problem characterization should be capable of representing critical sections and the demand/supply imbalance estimated by CARTESIUS.

IR 17: A basic CARTESIUS web client could be developed that shows the status of a jurisdiction's CARTESIUS agent.

External Interface Requirements

ER 1: An external interface for diagnostic information sharing must be provided. This interface must allow for diagnostic information sharing that allows for non-blocking information propagation between agents regarding conditions that could potentially produce spillback effects across jurisdictions.

ER 2: An external interface consistent with CARTESIUS core logic for DPS must be provided.

ER 3: The DPS information sharing interface must offer non-blocking operation.

ER 4: The DPS information sharing interface must support a protocol for information sharing that allows the agent collective to determine whether or not all agents have exhausted their feasible search space, and therefore will not be propagating further constraints across jurisdictions.

ER 5: CARTESIUS must interface with available event notification systems to obtain information regarding new and evolving events in the system.

ER 6: CARTESIUS must interface with the CHP CAD system

ER 7: The D12 agent should also interface with the D12 TMC activity log (see section 2.5.2) to get more detailed information regarding incidents in that jurisdiction.

ER 8: CARTESIUS must interface with all traffic control subsystems used by a particular jurisdiction to measure and control traffic.

ER 9: The CARTESIUS control and measurement interfaces must import external information into data objects used to implement the core CARTESIUS logic.

ER 10: The CARTESIUS control interfaces must be able to translate control settings from data objects used to implement the core CARTESIUS logic.

ER 11: The CARTESIUS D12 agent must interface with all measurement and control functions provided by the Caltrans D12 Real-time data Intertie

ER 12: The CARTESIUS Caltrans City of Irvine agent must interface with all measurement and control functions provided by the CTNET traffic signal management system

ER 13: CARTESIUS must obtain time-varying estimates of the traffic demand on all paths through the managed network.

ER 14: The demand should be represented as the total number of vehicle-trips using each path per time period.

ER 15: Demand estimates should be provided to CARTESIUS whenever new estimates are available.

ER 16: CARTESIUS should also be able to request that a new demand calculation be performed.

ER 17: CARTESIUS might obtain estimates of roadway capacity from external models or databases that have performed such estimates.

ER 18: CARTESIUS may need to communicate with external control or model systems in order to estimate available capacities.

ER 19: CARTESIUS should have access to the historical data from the system for all inputs required by CARTESIUS in real-time operation.

Logical Database Requirements

DR 1: CARTESIUS must use a database to store all possible data related to its operation.

DR 2: All information stored must be kept until it is explicitly removed from DR 2

the database.

DR 3: The data must also remain consistent throughout its lifetime such that removal of particular entities from the database, such as a user, will also remove any static and dynamic information that is exclusively associated with that entity.

DR 4: Deletions of linked data should not be allowed and the system should report an error condition rather than carry out the operation so that consistency can be maintained.

DR 5: CARTESIUS must maintain an internal data model for representing traffic networks.

DR 6: CARTESIUS network data model should be compatible with analytical representations of current microsimulation models.

DR 7: CARTESIUS must maintain an internal data model for representing traffic

signal timing plans on the 2070 controllers.

DR 8: CARTESIUS must maintain an internal data model for representing ramp meter control settings consistent with those used in D12 operations.