California AHMCT Program
University of California at Davis
California Department of Transportation

REVIEW OF MN/IRIS SOFTWARE & TEST CASES FOR CALTRANS DISTRICT 10 IRIS DEMONSTRATION STUDY*

Michael T. Darter[1], Stephen M. Donecker[1],
Kin S. Yen[1], Bahram Ravani[1], &

Ty A. Lasky[1], Principal Investigator

AHMCT Research Report
UCD-ARR-07-12-31-01

Interim Report of Contract IA 65A0210 - Task Order 06-22

December 31, 2007

Affiliations:
1. AHMCT Research Center, Department of Mechanical & Aeronautical Engineering, University of California, Davis, CA 95616-5294

* This report has been prepared in cooperation with the State of California, Business and Transportation Agency, Department of Transportation and is based on work supported by Contract Number 65A0210 - Task Order 06-22 through the Advanced Highway Maintenance and Construction Technology Research Center at the University of California at Davis.

Review of Mn/IRIS Software & Test Cases for Caltrans District 10 IRIS Demonstration Study

Technical Documentation Page

1. Report No. F/CA/RI-2006/10	2. Government Accession No.		3. Recipient's Catalog No.
4. Title and Subtitle Review of Mn/IRIS Software & Test Cases for Caltrans District 10 IRIS Demonstration Study			5. Report Date December 31, 2007 6. Performing Organization Code
7. Author(s): Michael T. Darter, Kin S. Yen, Stephen Donecker, Bahram Ravani, & Ty A. Lasky			8. Performing Organization Report No. UCD-ARR-07-12-31-01
9. Performing Organization Name and Address AHMCT Research Center UCD Dept of Mechanical & Aeronautical Engineering Davis, California 95616-5294			10. Work Unit No. (TRAIS) 11. Contract or Grant IA 65A0210 - Task Order 06-22
12. Sponsoring Agency Name and Address California Department of Transportation Division of Research and Innovation 1127 O Street Sacramento, CA 94273-0001			13. Type of Report and Period Covered Interim Report October 2005 - June 2008 14. Sponsoring Agency Code
15. Supplementary Notes			
16. Abstract This document describes Task 5, Review of Mn/IRIS Software and Test Cases for Caltrans District 10 IRIS Demonstration Study, within the Open ATMS multi-year research project undertaken by the Advanced Highway Maintenance & Construction Technology (AHMCT) Research Center at the University of California, Davis. The Open ATMS project is implementing an open-source Advanced Traffic/Transportation Management System (ATMS) within the California State Department of Transportation (Caltrans) District 10 (D10) Transportation Management Center (TMC). This document is a brief review of Mn/DOT IRIS TMC software design, interfaces, modules, algorithms, and test cases.			
17. Key Words ATMS, Open-Source Software, OSS, TMC, RWIS, Highway operations, IRIS,		18. Distribution Statement No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.	
20. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 40	22. Price

Form DOT F 1700.7 (872) Reproduction of completed page authorized

(PF V2.1, 6/30/92)

Abstract

This document describes Task 5, Review of Mn/IRIS Software and Test Cases for Caltrans District 10 IRIS Demonstration Study, within the Open ATMS multi-year research project undertaken by the Advanced Highway Maintenance & Construction Technology (AHMCT) Research Center at the University of California, Davis. The Open ATMS project is implementing an open-source Advanced Traffic/Transportation Management System (ATMS) within the California State Department of Transportation (Caltrans) District 10 (D10) Transportation Management Center (TMC). This document is a brief review of Mn/DOT IRIS TMC software design, interfaces, modules, algorithms, and test cases.

Table of Contents

List of Figures

Disclaimer/Disclosure

The research reported herein was performed as part of the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center, within the Department of Mechanical and Aeronautical Engineering at the University of California Davis, and the Division of Research and Innovation at the California Department of Transportation. It is evolutionary and voluntary. It is a cooperative venture of local, State and Federal governments and universities.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California, the Federal Highway Administration, or the University of California. This report does not constitute a standard, specification, or regulation.

Acronyms and Abbreviations

Acronyms used within this document are defined below.

AHMCT	Advanced Highway Maintenance & Construction Technology
ATMS	Advanced Traffic/Transportation Management System
Caltrans	California State Department of Transportation
CMS	Changeable/Dynamic Message Sign
D10	District 10
DMS	Dynamic/Changeable Message Sign
DRI	Division of Research and Innovation
ESRI	Environmental Systems Research Institute
HTTP	HyperText Transfer Protocol
I/O	Input/Output
IRIS	Intelligent Roadway Information System
JAR	Java ARchive
LDAP	Lightweight Directory Access Protocol
Mn/DOT	Minnesota Department of Transportation
Mn/IRIS	Mn/DOT IRIS
MPEG-4	Moving Picture Experts Group 4
OSS	Open-Source Software
PTZ	Pan Tilt Zoom
RMI	Remote Method Invocation

RWIS Remote Weather Information System / Roadway Weather Information System

TAG Technical Advisory Group

TLS Transport Layer Security

TMC Transportation Management Center

TMS Transportation Management System

UI User Interface

URL Uniform Resource Locator

UTM Universal Transverse Mercator

WGS-84 World Geodetic System 84

XML eXtensible Markup Language

Acknowledgments

The authors thank the California State Department of Transportation for their support; in particular, the guidance and review provided by the Open ATMS project team and Technical Advisory Group. The authors also acknowledge the dedicated efforts of the AHMCT development team members. Special thanks to Caltrans Headquarters Traffic Operations (Stan Slavin and Alan Benson), Caltrans Division of Research and Innovation (DRI) (Fred Yazdan, Roya Hassas, and Sean Campbell), Caltrans District 10 Traffic Operations (Clint Gregory, Veronica Cipponeri, Arlene Cordero, Mohammad Battah, Joe Silvey, and James Collins), Caltrans District 12 Traffic Operations (Omid Segal and Morteza Fahrtash), Larry Tjoelker (Caltrans Headquarters), and Minnesota Department of Transportation (Mn/DOT) (James Kranig, Doug Lau, Timothy Johnson, and Ralph Adair).

Chapter 1

Introduction

1.1 Document Purpose

This document is a brief review of Mn/DOT Intelligent Roadway Information System (IRIS) TMC software design, interfaces, modules, and algorithms. It is a technical description of the existing IRIS code base as it relates to the implementation of the IRIS ATMS for Caltrans D10. The intended audience is developers, engineers and IRIS implementers. Readers seeking a higher-level functional overview of the IRIS ATMS should consult the IRIS As Built System Design Document, which complements this document[7]. These findings are the result of a single task within a multi-year research project undertaken by the Advanced Highway Maintenance & Construction Technology (AHMCT) Research Center at the University of California, Davis[1].

1.2 Project Background

In October 2005, DRI and AHMCT initiated a research project to study the potential benefits Open-Source Software (OSS) might provide for Caltrans in the ATMS and TMC domains. OSS can be used in two ways and provides correspondingly different potential benefits:

1. Transportation agencies may benefit from using OSS products such as Linux, MySQL[2], etc. as part of ongoing transportation projects. These transportation projects may be open or closed-source projects.
2. Transportation agencies may benefit from creating open-source transportation projects that share software source code, data sets, test results, documentation, resources, etc. with a community of users and transportation agencies.

[1]For AHMCT see http://ahmct.ucdavis.edu

[2]All company and product names listed herein are the trademarks of their respective companies.

A number of completed and ongoing transportation projects are using OSS in both ways and have reported corresponding benefits [1]. The ongoing Open ATMS project is the second type–it is an OSS project using OSS software products, implementing the Minnesota Department of Transportation (Mn/DOT) IRIS open-source ATMS software, within Caltrans D10.

1.3 Research Tasks and Progression

The Open ATMS project task order is shown below. Software development uses an evolutionary linear process model, which iteratively follows the waterfall sequence of requirements definition, design, development, and testing. The project tasks are:

1. Formation of the advisory group,
2. Literature Review of National Developments in ATMS and Open Source Software [1],
3. Review of current Caltrans D10 ATMS operations and equipment,
4. Development of demonstration open-source ATMS implementation requirements,
5. Review of Mn/DOT IRIS source code and documentation,
6. D10 IRIS design,
7. D10 IRIS implementation,
8. Test plan development,
9. Lab testing, field testing, and system demonstration,
10. Documentation.

1.4 Additional Documentation

For further information consult documents in the references (Section 2.10 on page 23) and the following:

- Intelligent Roadway Information System (IRIS) As Built System Design Document [7],
- IRIS JavaDoc source code documentation[8],
- IRIS Source code documentation: some modules contain additional documentation. See the doc sub-directory within each module's repository [9],

- AHMCT Open ATMS Task 2 Report [1],
- AHMCT Open ATMS Task 3 Report [2],
- AHMCT Open ATMS Task 4 Report [3],
- AHMCT Open ATMS Task 5 Report [4],
- AHMCT Open ATMS Task 6 Report [5].
- AHMCT Open ATMS Final Report [6].

1.5 Document Scope

This document was developed by the researchers using software source code, project documentation, research reports, software and hardware specifications, software documentation, field and laboratory data, and guidance and information gained through collaboration with the project engineers and staff from the Technical Advisory Group (TAG), D10, and Mn/DOT.

1.6 Summary of Material to Follow

The following chapter discusses IRIS design, class and interface organization, and existing test cases. The document is organized by IRIS Java module.

Chapter 2

Review of Mn/DOT IRIS Software

2.1 Introduction

This chapter provides a brief review of Mn/DOT IRIS TMC software design, interfaces, modules, algorithms, and test cases. The chapter is organized by IRIS module. For detailed interface, class, and method descriptions, refer to the project JavaDoc. Class, object, method, and interface names referenced below are assumed to be in the namespace us.mn.state.dot.

Object-oriented Programming Terms

The following terms are used in the rest of the document and are object-oriented programming concepts.

- Class: a programming language construct used to group related fields (or attributes) and methods under a single name and within a namespace.[1].
- Interface: defines the communication boundary between two entities, such as a piece of software, a hardware device, or a user. It is also a programming language concept used to define a class without fields (or attributes).[2].
- Method: a subroutine exclusively associated with a class or interface[3].
- Module: a unit of functionality that groups related compiled Java classes into a single Java ARchive (JAR) file[4][5].

[1]See http://en.wikipedia.org/wiki/Class(computerscience).
[2]See http://en.wikipedia.org/wiki/Interface(computerscience).
[3]See http://en.wikipedia.org/wiki/Method(computerscience).
[4]See http://en.wikipedia.org/wiki/Java_Module_System
[5]See http://en.wikipedia.org/wiki/Module(programming)

- Namespace: an abstract container that provides context for names. It also provides disambiguation of items having the same name and residing in different namespaces[6].

- Object: an instance of a class, allocated within memory.[7].

IRIS Structure

The IRIS server is composed of a single application and associated HyperText Transfer Protocol (HTTP) servlets. The IRIS client is a single application that connects to the IRIS server and servlets. All IRIS modules are written in Java. Module interdependencies are shown in Figure 2.1. The modules are discussed in sections 2.2 to 2.10 on pages 7–17.

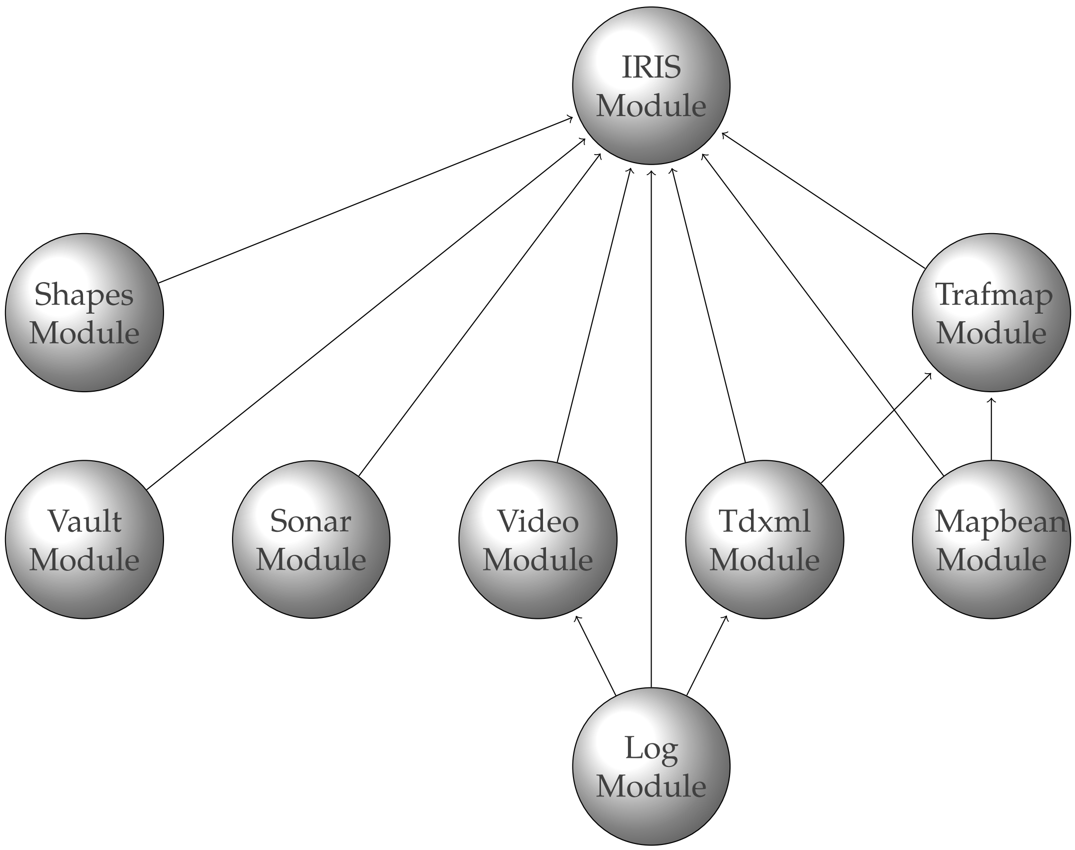

Figure 2.1: IRIS internal module dependencies

[6]See http://en.wikipedia.org/wiki/Namespace(computerscience).
[7]See http://en.wikipedia.org/wiki/Object(computerscience).

2.2 IRIS Server and Client Module (Iris)

Overview

This module consists of the IRIS client and server. It contains approximately 72,000 lines of code. Functionality can be broken down by namespace, as shown below. Significant documentation exists within the IRIS module repository. This includes documentation on:

- Stratified zone metering,
- Roadway network geometry,
- Mn/DOT unified traffic data file format,
- Travel time estimation.

Namespaces

- us.mn.state.dot.tms
- us.mn.state.dot.tms.comm
- us.mn.state.dot.tms.comm.canoga
- us.mn.state.dot.tms.comm.manchester
- us.mn.state.dot.tms.comm.mndot
- us.mn.state.dot.tms.comm.ntcip
- us.mn.state.dot.tms.comm.pelco
- us.mn.state.dot.tms.comm.smartsensor
- us.mn.state.dot.tms.comm.vicon
- us.mn.state.dot.tms.client
- us.mn.state.dot.tms.client.proxy
- us.mn.state.dot.tms.client.camera
- us.mn.state.dot.tms.client.device
- us.mn.state.dot.tms.client.dms
- us.mn.state.dot.tms.client.incidents

- us.mn.state.dot.tms.client.lcs
- us.mn.state.dot.tms.client.meter
- us.mn.state.dot.tms.client.monitor
- us.mn.state.dot.tms.client.proxy
- us.mn.state.dot.tms.client.roads
- us.mn.state.dot.tms.client.security
- us.mn.state.dot.tms.client.toast
- us.mn.state.dot.tms.client.tour
- us.mn.state.dot.tms.client.warning
- us.mn.state.dot.tms.log
- us.mn.state.dot.tms.utils

Details

The IRIS server is responsible for:

- Authenticating users and providing access based on privilege level,
- Communicating with all hardware field devices, including periodic polling,
- Reading state information from the PostgreSQL database on start-up (tms),
- Instantiating in-memory Java objects using database state information,
- Communicating with IRIS clients and providing real-time Transportation Management System (TMS) object state information,
- Writing state changes to the database,
- Logging events to a PostgreSQL databases (log),
- Scheduling and executing jobs.

The comm namespace (us.mn.state.dot.tms.comm) contains functionality related to communicating with field devices such as Dynamic/Changeable Message Sign (DMS), ramp meters, loop traffic detectors, microwave detectors, and cameras. It consists of approximately 19,800 lines of code. The primary classes and interfaces are shown in Figures 2.2 on page 19, 2.3 on page 20, and 2.4 on page 21[8]. Individual hardware drivers

[8]For an explanation of class diagrams see http://en.wikipedia.org/wiki/Class_diagram.

are in sub-directories (e.g. smartsensor) and contain extended classes with functionality specific to the device. Figure 2.4 on page 21 shows key hardware driver classes. The ControllerOperation class is subclassed by device. Each extended ControllerOperation contains a number of internal Phase classes, which provide a finite state machine structure, using responses from the hardware devices to determine the next state.

The IRIS server calculates travel times, which are displayed on Changeable/Dynamic Message Signs (CMSs). The travel time calculation algorithm is described in detail within the IRIS source code documentation[9]. The IRIS As Built document also describes travel time functionality[7]. Classes and interfaces involved in travel time calculations include: BadRouteException, CorridorTrip, DMSImpl, DMS, DMSListImpl, HolidayImpl, Holiday, MultiString, RouteBuilder, Route, SegmentListImpl, SignTravelTime, StationImpl, and TMSImpl. These classes and interfaces are within the us.mn.state.dot.tms namespace. The use of travel time calculations outside of Minnesota will require route definitions for the areas of interest.

Important Server Classes:

- MainServer: the main IRIS server class, contains main(). Container for TMSImpl.
- Main: IRIS client entry point, reads the client property file, instantiates an IrisClient object.
- TMSObjectImpl: abstract base class of all TMS objects, extended by TMSImpl. Extends UnicastRemoteObject, implements TMSObject, Storable. Container for a single ObjectVault. Container for static instances of: Scheduler, CommunicationLineList, SQLConnection, NodeGroupList, RoadwayListImpl, DetectorListImpl, StationListImpl, StationMapImpl, SegmentMapImpl, R_NodeMapImpl, TimingPlanListImpl, RampMeterListImpl, among others.
- TMSImpl: an Remote Method Invocation (RMI) object, contains all the global TMS object lists. Extends TMSObjectImpl, and implements the TMS interface. It is responsible for scheduling tasks such as the five-minute timer job.
- Storable: an interface implemented by TMSObjectImpl for objects that store their state in a database (e.g. CircuitImpl, R_NodeImpl, and other objects subclassed from TMSObjectImpl).

Important Client Classes:

- Main: the client entry point, contains main(). It also reads the IRIS client property file on start-up. Container for IrisClient.
- IrisClient: the visual IRIS client, extends JFrame. Container for the Session object.

[9] See iris/docs/travel_time.html within the IRIS source code.

- Session: is a single IRIS client login session. Contains TmsConnection, Properties, Logger, multiple base layers, ViewLayer, StationLayer, TmsIncidentLayer, TmsMapLayer, multiple IRIS tab objects.
- MapTab: container for a MapBean, extends IrisTab.
- IrisTab: abstract base class of all tabs used within the IRIS client. Extended by RoadwayTab, DMSTab, MapTab, etc.
- client.proxy.TmsMapLayer: the camera layer, extends Layer, implements DynamicLayer.
- Scheduler: extends Thread, multiple instances are contained by TMSImpl. An internal class Jobs is defined and stored in a TreeSet.

Video related client classes:

- client.camera.CameraViewer: extends javax.swing.JPanel, uses the Video module to provide CMS video images within the client on the DMS tab. Contains Camera, VideoMonitor instances. When the play button is pressed on the client, a new Camera object is created and a new RepeaterImageFactory, which is contained by the VideoMonitor contained by CameraViewer.

Test Cases

The module contains a number of Java classes used for testing. This includes TestLaneControlSignal, TestTrafficDevice, TestDevice, and TestTmsObject. Other classes relate to the client and Lightweight Directory Access Protocol (LDAP) authentication.

2.3 Application Logging Module (Log)

Overview

The log module contains convenience classes related to logging program messages and states. It also contains convenience methods for interacting with databases, eXtensible Markup Language (XML) files, and HTTP clients. The module contains approximately 700 lines of code.

Namespaces

- us.mn.state.dot.log: main namespace,

- us.mn.state.dot.util: convenience classes,
- us.mn.state.dot.util.db: database functionality,
- us.mn.state.dot.util.xml: XML functionality.

Details

The Log module is used by multiple modules (see Figure 2.1 on page 6). Logging functionality is ultimately provided through the Java Logger class (java.util.logging.Logger).

Classes used by other modules:

- TmsLogFactory: static convenience logging methods.

Test Cases

There are no test cases for the Log module.

2.4 Mapping and Shapefile Module (Mapbean)

Overview

The Mapbean module provides the general ability to display Environmental Systems Research Institute (ESRI) shape files within the IRIS client. This functionality is used by the Trafmap module (see Figure 2.1 on page 6). It contains approximately 3,900 lines of code.

Namespaces

- us.mn.state.dot.map: main namespace,
- us.mn.state.dot.map.event: functionality for flagging map changes requiring a repaint,
- us.mn.state.dot.map.marker: map markers,
- us.mn.state.dot.map.shapefile: functionality for viewing, parsing, manipulating shape files. This includes ShapeLayer, an extended Layer class, which provides the ability to view layers created from shape files.

Details

Relevant classes and interfaces:

- MapBean: a Java Swing visual component, extended from the javax.swing.JComponent class. It is a container for a MapPane. It is contained by a MapTab object in the IRIS client.
- MapObject: interface for objects drawn on a MapBean.
- MapPane: extends Thread. Container for multiple Layer objects.
- Layer: an abstract class, extended by layers in the Trafmap module. Also extended by classes within the shapefile namespace.
- DynamicLayer: is an interface that must be implemented by extended Layer classes. Layers that do not implement DynamicLayer are assumed to be static.
- Symbol: an interface, a graphical representation of a map object.
- Theme: a container of Symbol objects for a single layer and is responsible for selecting which symbol to use for a particular map object.
- StyledTheme: a theme with a Style container. Extends Theme.
- ShapeTheme: a theme which uses a single Symbol to draw all java.awt.Shape objects. Extends StyledTheme.
- shapefile.ShapeLayer: contains a list of ShapeObject objects read from a shape file. Extends Layer.
- shapefile.ShapeObject: a record from an ESRI shape file. Implements MapObject.

Test Cases

There are no test cases for the Mapbean module.

2.5 Map Module (Shapes)

The Shapes module contains maps used by the IRIS client. It contains no procedural code.

2.6 Client and Server Communication Module (Sonar)

Overview

The Sonar module provides a communications protocol and implementation used by IRIS clients and server. The IRIS documentation for this module provides an overview and details about the protocol–see the sonar module within the IRIS source code. Sonar uses Transport Layer Security (TLS) for communication between client and server. It provides client notification when an object on the server changes. The Sonar module contains approximately 5,000 lines of code.

Namespaces

- us.mn.state.dot.sonar
- us.mn.state.dot.sonar.client
- us.mn.state.dot.sonar.server
- us.mn.state.dot.sonar.test

Details

Sonar uses the Java NIO package for client and server communication, which provides multiplexed Input/Output (I/O) using channels and selectors.

Relevant classes and interfaces:

- server.Server: extends Thread, provides communication with all connected clients. Contained by MainServer.

Test Cases

A test client and server exist.

2.7 XML Data Reader Module (Tdxml)

Overview

The Tdxml module provides the ability to receive periodic updates of station and incident data. This functionality is used by the Trafmap module and the IRIS client (see

Figure 2.1 on page 6). Class clients add themselves as listeners to receive notification that station and incident data has changed. Incident and station data syntax is XML and is received through the Uniform Resource Locator (URL) specified in the client properties file. The module also contains convenience methods that provide Universal Transverse Mercator (UTM) and World Geodetic System 84 (WGS-84) coordinate conversion. It contains approximately 3,200 lines of code.

Namespaces

- us.mn.state.dot.tdxml: main namespace,
- us.mn.state.dot.tdxml.geo: convenience methods related to coordinate conversion.

Details

To use module functionality, an object creates an instance of an XmlClient subclass or retrieves a reference to an existing object. The caller also implements the DdsListener interface, through which the XmlClient notifies the caller when a new Station or Incident is received.

Relevant classes and interfaces:

- XmlClient: this abstract class is the primary interface to module functionality. An instance is a thread, implementing java.lang.Runnable. Subclasses are XmlStationClient and XmlIncidentClient. Instances are threads, run continuously, and are assumed to have a lifetime of the application. Only one instance is created of each subclass. Other classes add themselves as listeners to the single XmlClient subclass instance. An instance is created by trafmap.IncidentLayer and trafmap.StationLayer. The object client.incidents.IncidentTab adds itself as a listener to the IncidentLayer instance. Instances contain a linked list of DdsListener objects.
- DdsListener: this interface is implemented by the class that instantiates XmlClient subclasses and passes a reference of itself to XmlClient subclasses using method addDdsListener(this).
- Incident and StationSample: instances are created by XmlClient subclasses when new data is received. Instances are passed to the client.
- LatLongUTMConversion: provides convenience methods to convert between UTM and WGS-84 coordinates.

Test Cases

No test cases are available for the Tdxml module.

2.8 Map Layer and Theme Module (Trafmap)

Overview

The Trafmap module provides IRIS-specific mapping functionality which is used by the client application. The Trafmap module uses the functionality provided by the Mapbean and Tdxml modules. The module contains approximately 1,700 lines of code.

Namespaces

- us.mn.state.dot.trafmap

Details

MapBean Module functionality is provided primarily by classes extended from Layer and ShapeTheme. Approximately eleven map layers are provided, e.g. IncidentLayer, WaterLayer, TunnelLayer, RoadLayer, MuniLayer, etc. These are contained by a MapPane object, which is contained by a MapBean object, which is contained within a MapTab in the IRIS client. The primary Trafmap class used by the IRIS client is the Layer class, which is subclassed a number of times in the client.

Relevant classes and interfaces:

- BaseMapLayer: base map on the LCS tab. Extends ShapeLayer, implements DynamicLayer.
- HighwayMarkerLayer: layer for displaying highway markers. Extends ShapeLayer.
- IncidentLayer: displays map incidents. Extends Layer, implements IncidentListener.
- MuniLayer: muni layer. Extends ShapeLayer.
- RoadLayer: roadway network layer. Extends ShapeLayer.
- StationLayer: displays traffic stations. Extends ShapeLayer, implements StationListener.

- TunnelLayer: displays tunnels. Extends ShapeLayer, implements DynamicLayer.
- ViewLayer: tracks view extents. Extends ShapeLayer.
- WaterLayer: displays water boundaries. Extends ShapeLayer.

Test Cases

There are no test cases for the Trafmap module.

2.9 Persistent Object Storage Module (Vault)

Overview

The Vault module provides the ability for Java objects to save and retrieve object state information using a PostgreSQL database. When the IRIS server starts, TMS objects such as Circuits, R_Nodes, etc. are instantiated using object state information retrieved from the PostgreSQL database. Objects that are stored in the database using this functionality are "Vault objects." The module contains approximately 3,100 lines of code.

Namespace

- us.mn.state.dot.vault

Details

Each Vault object is subclassed from TMSObjectImpl, which contains a single static instance of ObjectVault, through which Vault functionality is provided. Relationships between primary classes are shown in Figure 2.5 on page 22. Vault uses database transactions.

Classes and interfaces used by other modules:

- ObjectVault: provides the ability for Java objects to store themselves in a PostgreSQL database. A single static instance is contained by TMSObjectImpl. It is a container for the database connection and overhead housekeeping information (e.g. a HashMap for types and type identifiers). The save() and load() methods are used to store and retrieve single objects to/from the database. Contained objects are also implicitly saved/loaded. The update() method is used by Vault objects to update the database when an object field value has changed.

- FieldMap: container (HashMap) for database table column names mapped to database values. Instances are created by Vault users and passed to each subclassed ObjectVault through its constructor.

- ObjectVaultException: extends Exception, provides detailed Vault exception information.

Test Cases

A stand-alone application for testing Vault operations is provided.

2.10 Video Module (Video)

Overview

The Video module consists of a client and server. The server is a Tomcat HTTP Java servlet. The client jar is distributed as part of the IRIS client. The Video module contains approximately 7,700 lines of code. The Protozoa server is a stand-alone server responsible for Pan Tilt Zoom (PTZ) camera control.

Namespaces

- us.mn.state.dot.video
- us.mn.state.dot.video.client
- us.mn.state.dot.video.dev
- us.mn.state.dot.video.server

Details

The IRIS client classes that interface with the Video module are CameraViewer and VideoMonitor, both in the namespace us.mn.state.dot.client.camera. Both subclass javax.swing.JPanel and are embedded on the DMS tab in the client. The video server reads a properties file on start up (/etc/tms/video.properties). All video server functionality is provided through extended video servers which are subclasses of javax.servlet.http.HttpServlet.

Video classes and interfaces used by the IRIS Client (CameraViewer):

- AbstractImageFactory: a subclass connects to an HTTP video stream from the stream server and notifies each of its listeners whenever there is a new image. Provides a list of video stream URLs to clients.
- Camera: a physical camera, contains geographic coordinates, cross street, highway number, ID, etc. Contained by CameraViewer.
- Client: a video stream parameter container, contained by CameraViewer.
- RepeaterImageFactory: extends AbstractImageFactory. Created within the CameraViewer playPressed() method and contained by a VideoMonitor, which is also contained by CameraViewer within the client.
- VideoException: extends Exception.

Relevant video server classes and interfaces:

- VideoServlet: abstract, extends HttpServlet, base class for all video server functionality.
- ArchiveServer: extends VideoServlet.
- ImageServer: extends VideoServlet, is a still video server.
- NvrServer: abstract, extends VideoServlet, Moving Picture Experts Group 4 (MPEG-4) support is provided through subclasses.
- StreamServer: extends VideoServlet, continuously provides a stream consisting of the image size and image data.

Relevant video client classes and interfaces:

- VideoMonitor: contained by CameraViewer. display a video stream, extends JPanel, implements ImageFactoryListener, ListSelectionListener. Contains a RepeaterImageFactory which is created when the play button is pressed on the client User Interface (UI).

Other relevant classes and interfaces:

- ImageFactoryListener: an interface, implemented by VideoMonitor, contains a method imageCreated(byte[]) that is called when a new video image arrives.
- ThreadMonitor: extends Thread, monitors other threads, used for debugging.

Test Cases

The IRIS client contains a class us.mn.state.dot.video.dev.VideoMonitorTest that provides a stand-alone application to test a video server.

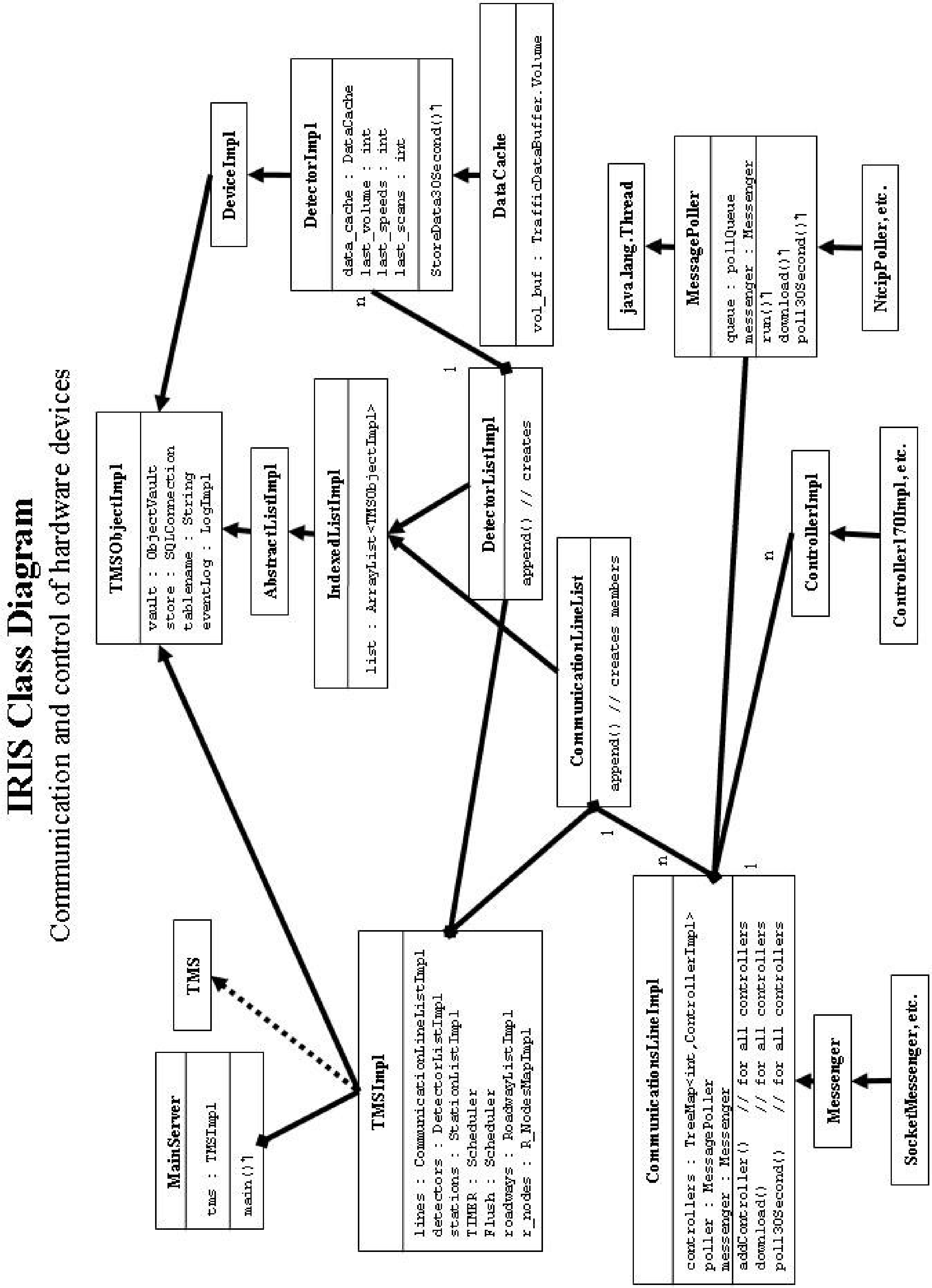

Figure 2.2: IRIS Server class diagram

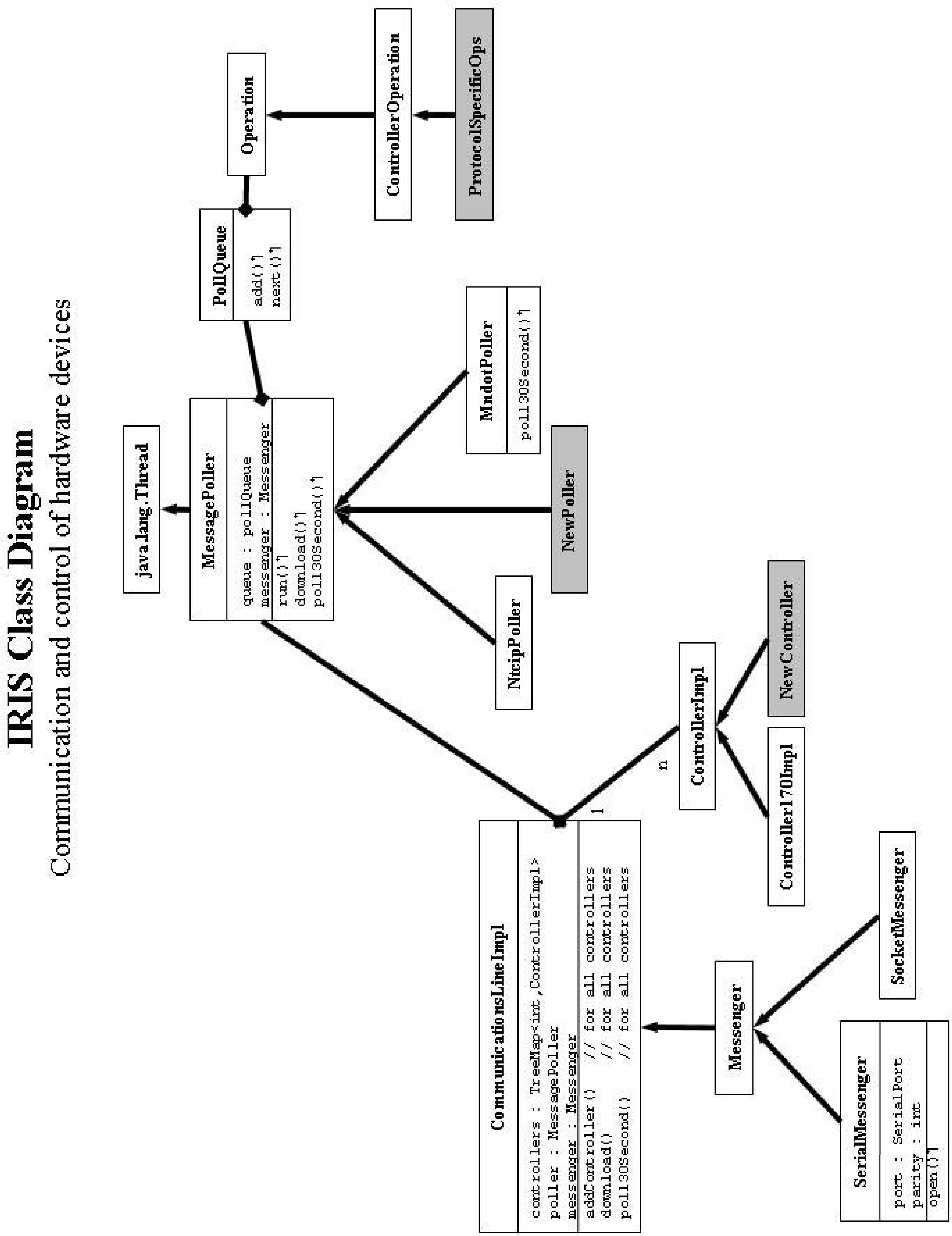

Figure 2.3: IRIS Server class diagram, hardware device communication

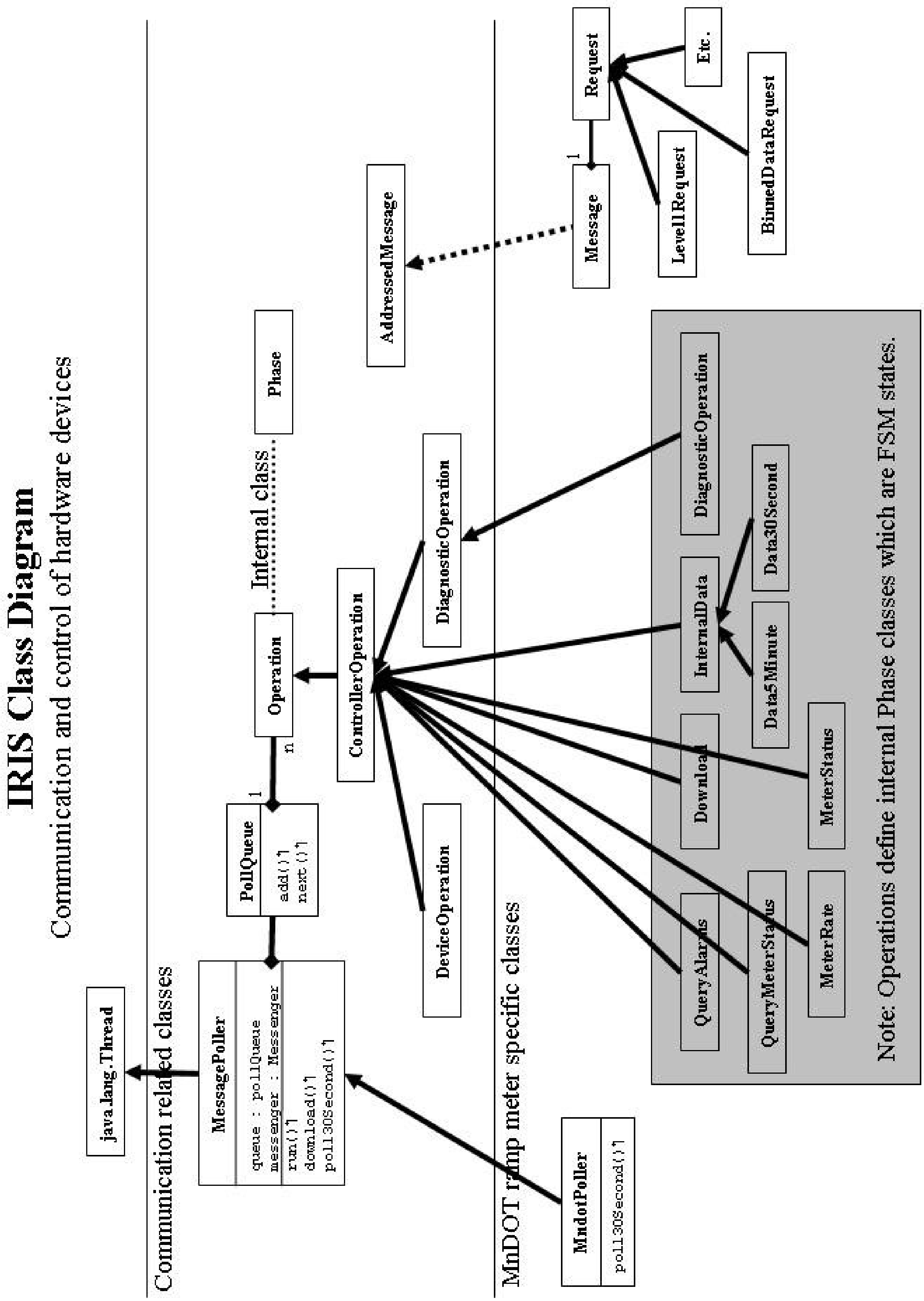

Figure 2.4: IRIS Server class diagram, hardware device driver

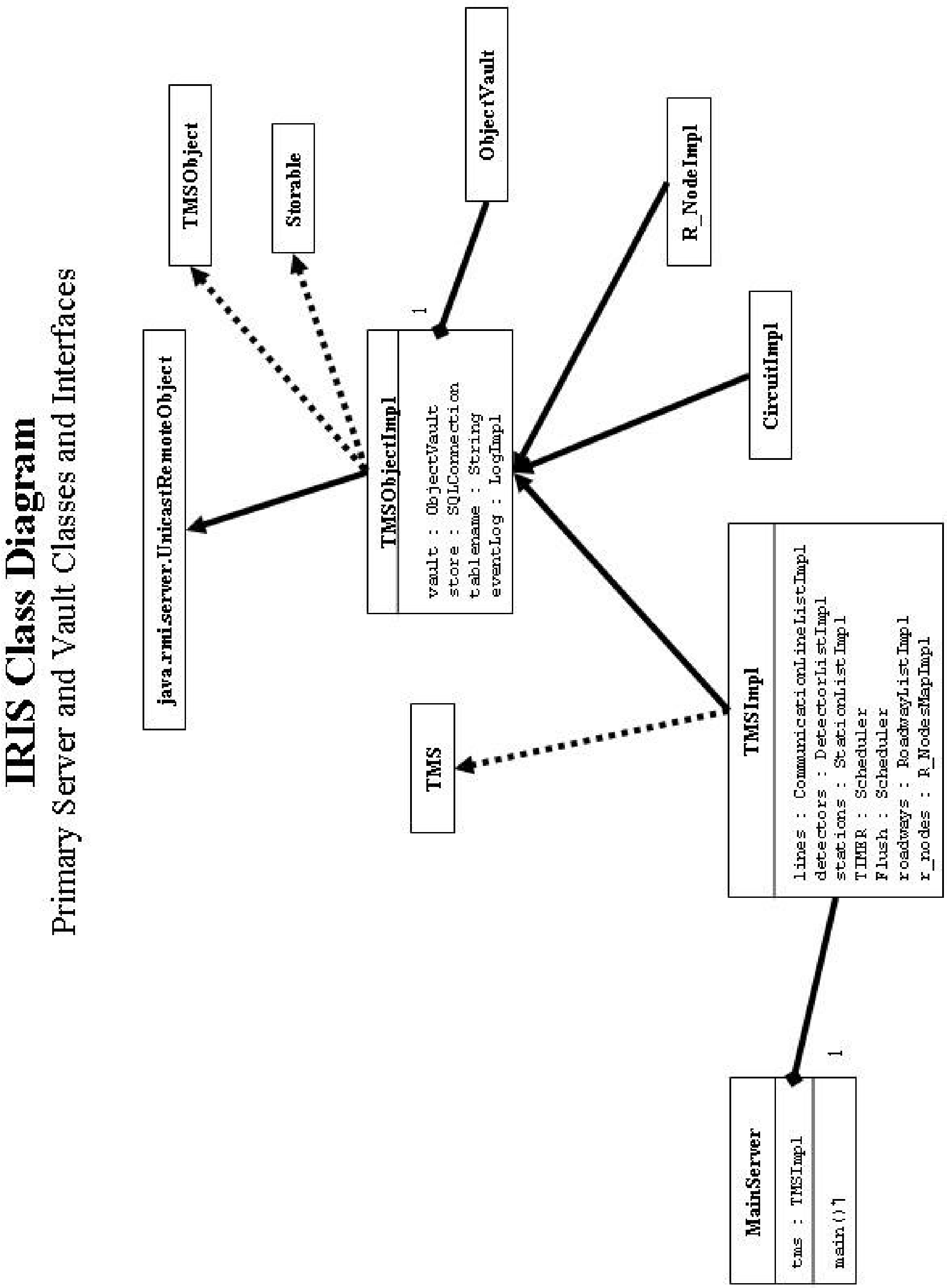

Figure 2.5: IRIS primary Server and Vault classes and interfaces

References

[1] M.T. Darter, K.S. Yen, B. Ravani, and T.A. Lasky. Literature Review of National Developments in ATMS and Open Source Software. Technical Report UCD-ARR-06-12-08-01, AHMCT, December 2006.

[2] M.T. Darter, S.M. Donecker, K.S. Yen, B. Ravani, and T.A. Lasky. Review of Caltrans District 10 Transportation Management Center Operations and Equipment. Technical Report UCD-ARR-07-09-30-01, AHMCT, September 2007.

[3] M.T. Darter, S.M. Donecker, K.S. Yen, B. Ravani, and T.A. Lasky. Review of Mn/IRIS and Caltrans District 10 TMC Compatibility and Functional Requirements for D10 IRIS Demonstration Study. Technical Report UCD-ARR-07-09-30-02, AHMCT, September 2007.

[4] M.T. Darter, S.M. Donecker, K.S. Yen, B. Ravani, and T.A. Lasky. Review of Mn/IRIS Software and Test Cases for Caltrans District 10 IRIS Demonstration Study. Technical Report UCD-ARR-07-12-31-01, AHMCT, December 2007.

[5] M.T. Darter, S.M. Donecker, K.S. Yen, B. Ravani, and T.A. Lasky. Overview of Caltrans District 10 IRIS Demonstration Design. Technical Report UCD-ARR-07-12-31-02, AHMCT, December 2007.

[6] M.T. Darter, S.M. Donecker, K.S. Yen, B. Ravani, and T.A. Lasky. Research and Development of Open-Source Advanced Traffic Management System Hardware and Software Components. Technical Report UCD-ARR-08-06-30-01, AHMCT, July 2008.

[7] Minnesota Department of Transportation. Intelligent Roadway Information System (IRIS) As Built System Design Document. Technical report, Minnesota Department of Transportation, June 2007.

[8] Minnesota Department of Transportation IRIS Project. Mn/DOT IRIS JavaDoc, December 2007.

[9] Minnesota Department of Transportation IRIS Project. Mn/DOT IRIS source code, December 2007.