Headway Monitoring and Layover Management Extensions to the OrbCAD Transportation Management System

Justin Ethier
Orbital Sciences Corporation
Transportation Management Systems
7160 Riverwood Drive
Columbia, MD 21046
[ethier.justin@orbital.com | jethie1@towson.edu]

1 Background

This paper discusses a suite of extensions to the OrbCAD Transportation Management System that were developed for SBS Transit (SBST). These extensions are part of the Automated Vehicle Management System (AVMS) that SBST will be using to manage their expanding fleet of over 2500 public transit busses.

Singapore is a small country located in southeastern Asia between Malaysia and Indonesia. The country has a land size of just over three times the size of Washington DC and a population of over four million [1]. As such, Singapore is highly urbanized, with the majority of its population depending on public transit as they go about their daily lives. In order to support the demand, there are several transit operators and many options for public transit, including bus, rail and taxi.

Figure 1: Map of Singapore

SBST is one of two major bus operators in Singapore. Its bus services have an average daily ridership of over two million people [2]. Prior to the deployment of AVMS, SBS did not have a computerized system for managing their bus fleet in real time (although an existing system called SCS is used to track vehicle/drive assignments). SBST was primarily focused on managing their fleet via staff at each interchange. These personnel would generally perform management by instructing busses to leave early or late. One of

the primary goals of the AVMS system was to consolidate these responsibilities into management by a central dispatch center.

Maintenance of bus headway[1] is especially important for SBS Transit. Not only is an even spacing between busses crucial for providing good service to riders, but as part of their agreement with the Singapore Government SBS Transit is required to maintain a specific headway on each of their routes. The required headway is typically between 10 or 15 minutes, although it varies by service. Failure to meet these requirements will result in penalties such as decreased funding from the government. Thus it is important that the AVMS system allows SBST to view bus headway in near real-time, and gives dispatchers the ability to make adjustments in a timely manner.

The existing OrbCAD system provided a large number of route and schedule adherence features. However its features were developed for various transit agencies in the United States, and as such these out-of-the-box features did not quite meet the needs of SBST. For example, OrbCAD was optimized to perform schedule adherence by scheduled times – early, late, and normal (on-time) – and had only a limited capacity for monitoring headway[2]. Thus there was a compelling need to develop new headway monitoring features for the Singapore AVMS project. In addition, Singapore requested the ability to perform schedule adjustments in real time in order to correct scheduling problems as they occur on the street – requiring additional changes to the OrbCAD system.

The primary goal of these new features is to provide an innovative, simple way for SBS transit to manage the headway between busses actively performing work on a service. To this end, three primary features have been developed – Headway Monitoring, Layover Management, and Service Restoration.

Headway Monitoring provides the ability to graphically monitor headway between vehicles currently performing work on a route. Headway Monitoring will track the headway between vehicles and report headway deviations to the dispatcher. Although the existing AVL display provides the ability to view the locations of vehicles across the city, it does not give any indication of the headway between vehicles on a route – thus a specialized display was required.

Layover Management may be used to graphically monitor the departure times of vehicles scheduled to depart from a common point. The display shows the projected status of departures in real time and warns if these departures are likely to be delayed. There are several reasons why a vehicle may not depart on time, including traffic congestion, schedule deviations, or minimum layover requirements (union-negotiated rules

[1] Headway is defined as the time interval between two vehicles traveling the same direction on the same route.

[2] An example may help to illustrate why. If busses arrive at a stop every 30 minutes (for example), riders will expect a bus at a specific time, else they will be required to wait another 30 minutes for a bus. However, if busses arrive at a stop every 10 minutes, the specific time the next bus will arrive is less important as long as one shows up within a reasonable amount of time. Across most of the US, busses do not run as frequently as in Singapore and thus Schedule Adherence becomes more important than Headway.

guaranteeing a minimum length for certain driver breaks). In this capacity, Layover Management essentially predicts the future headway deviation between vehicles, and is intended primarily for use in monitoring departures from interchanges – locations that, operationally, are most ideal from which to instruct vehicles to take corrective action.

Service Restoration allows a controller to perform adjustments to vehicle service. This functionality is provided from the Headway / Layover front-ends as a set of control functions, in addition to existing features provided by the OrbCAD system. Service Restoration will provide tools to manage the headway on service between interchanges. The main goal is to detect headway issues (using the Headway/Layover features) and to correct them by allowing real-time service adjustments to the applicable vehicles.

Each of these features will be presented and explained in depth in the following sections of this document.

2 Technology

A wide array of technologies was employed during the development effort. This section describes the primary technologies we utilized.

2.1 Technologies

2.1.1 .NET Framework

All of the new processes developed for this initiative were written from the ground-up to run on top of Microsoft's .NET Framework. .NET was been developed by Microsoft to replace its aging Windows 32 API's. The .NET Framework is currently the platform that Microsoft is encouraging newer Windows applications to be written for.

.NET allows for platform-independent and language-independent development. In order for a language to be compatible with .NET, that language must meet a set of criteria that allow it to be compiled down to Microsoft Intermediate Language (MS-IL). At runtime, MS-IL code is executed by the Common Language Runtime (CLR) which uses Just-in-time (JIT) compilation so that the intermediate code may be executed as native code for maximum performance.

The development of this software utilized version 1.1 of the framework, however the current version is 2.0, with the release of version 3.0 rapidly approaching.

Finally, although Microsoft's .NET Framework is only written to run on Windows, there are open source versions of .NET that support other platforms. The most popular of these is Mono [3], which supports several platforms including Linux, Windows, and Mac OS X.

2.1.2 Web Services

Web Services are programmatic interfaces exposed via HTTP. These interfaces allow a producer to expose an API (set of methods) via the web. A remote host may then query that service and consume data returned by such requests. In effect this is quite similar to a Remote Procedure Call (RPC), but is much more powerful in that it is cross-platform, standardized, and operates over the ubiquitous HTTP protocol.

Several core standards are used by Web Services, including:

- Simple Object Access Protocol (SOAP) is used to transfer data between Web Services. SOAP provides an XML specification for data representation, which enables web services to work on any platform implementing the protocol.
- Web Service Description Language (WSDL) allows remote interfaces to be described.
- Universal Description, Discovery, and Integration (UDDI) is a protocol used to publish and discover web services. UDDI allows web services to be found by

applications at run-time or design time – For example, Visual Studio uses UDDI to dynamically generate proxy classes at design time for use by clients.

All in all, these specifications standardizes web services, allowing (for example) a Java client to consume services written in C# and hosted on Microsoft's IIS Web Server. These specifications are maintained and developed by the World Wide Web Consortium (W3C).

In C#, web service methods may be declared using the `[WebMethod]` directive from a class inheriting from the web service namespace. The compiled code may then be hosted by a web server to allow such a method to be called by a remote host.

2.1.3 Smart Clients

Smart Clients is a term use to refer to rich-client applications developed for the .NET Framework.

2.2 Languages

2.2.1 C#

C# was developed from the ground-up by Microsoft as the primary language for writing applications using the .NET Framework. C# is a modern object-oriented, garbage-collected language. As compared to C/C++, C# allows much greater programmer productivity by providing integration with Microsoft's richly-featured .NET Framework as well as running as Managed Code, which alleviates the programmer from the burden of memory management.

Given the relatively young age of C#, the language has been able to leverage innovations in other languages over the past several years, including those of Java. In fact, C# has many similarities to Java, although the languages do differ in many aspects, for example Checked Exceptions (Java) versus Unchecked Exceptions (C#). In any case, C# is rapidly gaining in popularity. For example, *Introduction to C#* is – as of July 2006 – the most popular online course at the ACM's Online Professional Development Centre [8].

C# is the primary language used in the development of the Headway Monitoring and Layover Management applications.

2.3 Tools

2.3.1 Microsoft Visual Studio 2003 .NET

Microsoft Visual Studio 2003 .NET is an IDE developed by Microsoft for developing .NET, Web, and Windows-based applications. Prior to the release of Visual Studio 2005, VS 2003 was the premier development environment for .NET applications. As such, it was utilized for the development of all .NET and Web-based features.

2.3.2 Microsoft SQL Server 2000

Microsoft SQL Server 2000 is the database used for this project. It is a fully featured relational database, although this is no longer the latest version of SQL Server, as SQL Server 2005 has recently been release.

2.4 Third Party Packages

2.4.1 Piccolo Toolkit

From day one, the Headway and Layover applications were conceived as rich UI applications having graphical canvases with custom UI components. In order to realize this goal, it would be necessary to incorporate many fundamental graphical features such as object selection, Zooming, and Panning. In order to incorporate these features, we either had to write them ourselves or find an off-the-shelf framework supporting them – which is where Piccolo comes in.

Piccolo [4] is an open-source graphics toolkit developed by researchers at the University of Maryland's Human-Computer Interaction Lab (HCIL). Piccolo is designed for development of rich user interfaces, in particular Zoomable User Interfaces (ZUI). It is a rewrite of Jazz, a package providing similar features but which was somewhat cumbersome [5]. Not only does Piccolo support all of the fundamental graphics features we needed but a native .NET implementation is also available, making it an attractive choice.

2.4.2 Developer Express Windows Forms Collection

The Windows Forms Collection is a suite of controls developed by Developer Express for use in .NET applications. The controls are developed in pure C#, easing integration with .NET applications. In addition, the tools are more feature-rich and visually appealing than the standard Windows Forms controls. In addition, the controls are available with full source code. More information is available from the Developer Express website [6].

3 Contributions

Due to the large scope of this project, there were several contributors throughout its development. The following people made significant contributions:

Justin Ethier wrote the design document Headway Monitoring (Smart Client / Web Service / Database only), Layover Management, and Service Restoration. This document outlined the features and high-level design for each of them, along with selected aspects of the detailed design. The document also provides the specification for a new .NET based architecture that is flexible enough both for these features and for future extensions to OrbCAD. The document also covers changes to the existing OrbCAD system to make this architecture possible. The architecture includes Smart Clients, Web Services, and a central back-end process (the Current Status Engine). In addition, he selected the Piccolo Graphics Framework for implementing custom graphics on our Smart Client applications. He implemented prototype Headway and Layover Smart Clients as proofs of concept, and implemented a common framework used by both the Headway and Layover smart clients. He implemented the Layover Smart Client, Engine (back-end), existing server code changes (C/C++), and layover database changes. He took the initial Service Restoration GUIs handed off by TJ and completed their development. Finally, he performed miscellaneous bug fixes across all system components and is currently providing support for the deployed production system.

Joseph Harvatine created concept drawings of the Headway Monitoring and Layover Management displays, and provided a significant amount of input during the requirements phase.

Blake Streeter provided valuable input on all aspects of the design, especially the Service Restoration components.

Gordon Cone implemented the Headway Smart Client and many changes/enhancements to the underlying OrbCAD system to support these new features, including changes to server code (C/C++), database, and Schedule Import (VB). He conceived and implemented countless enhancements to OrbCAD as part of this effort, including many that will be of substantial benefit to other projects. In addition he performed miscellaneous bug fixes across all system components and is currently providing support for the deployed production system

Anne Winston wrote the design document for the headway algorithms, and changes to the existing OrbCAD system. She implemented the Current Status Engine, a process that hosts the other back-end modules, and wrote the initial implementation of the Headway Engine. She also implemented the initial version of libraries that allow interoperability between the existing C/C++ DLL’s and the new .NET applications. An important aspect of this change was writing code to allow the proprietary IPC system used by the C/C++ code to talk to the CSE

Ching Tsai implemented the Web Services and the related database changes to support them.

TJ Chen implemented the initial version of the Service Restoration GUI's.

Larry Jones selected the Developer Express custom controls to improve our Smart Client applications and reduce development time. He provided significant input to the Web Services design, including guiding the format of data returned from the various Web Services. He wrote a design document for and implemented the Service Summary feature, a separate feature using the architecture developed for Headway/Layover. Additionally, he implemented portions of the Headway Smart Client and back end. Finally, with over 3 years of experience developing .NET applications, Larry served as a technical advisor during the middle – late stages of development, and provided valuable input regarding .NET and the Visual Studio IDE.

Project

4 System Overview

OrbCAD is a fleet management system designed for use in mid to large size public transit fleets. The system generally manages fleets of busses, although there are capabilities for Light Rail, Heavy Rail, On-Demand service (such as Para transit), and other vehicles such as supervisor cars. The main capabilities of the system are GPS location-reporting, voice/data communications, incident reporting, route/schedule adherence (RSA), and passenger counting. In addition there are several auxiliary features such as reporting. The software is intended for use across a transit organization, including use by bus operators, dispatchers, road supervisors, IT staff, managers, bus scheduling personnel, and customer service.

OrbCAD consists of two major systems: the Fixed End and the Vehicle. The vehicle system consists of an Advanced Mobile Data Terminal (AMDT), which consists of an on-board computer, LCD screen with buttons, and a headset that may be used for communications between the driver and dispatchers at the Fixed End. The AMDT unit runs embedded software on the QNX platform and communicates over the air (OTA) to the Fixed End dispatch system. OTA communications may be transmitted over a conventional (dedicated) radio system, or via a public communications system such as a cell phone network provider's.

The features described in this document are only applicable to the OrbCAD Fixed End, and thus this part of the system will be the main focus. Unless otherwise noted, all features described from here on are with respect to the Fixed End.

5 Existing Architecture

The Fixed End consists of several components. Proprietary TMD/RMD devices are used to process raw data received OTA from a conventional radio system. For cell phone network integration, these devices are replaced by a GTMD process, which runs on a linux machine and processes raw messages received via GPRS. Raw communications are then forwarded to the Data Communications Controller (DCC) which is responsible for processing message received from the vehicle or messages that the fixed end wishes to transmit to a vehicle. An application server is used to perform data processing, save received data to the to the database in real-time, and perform any other required processing. Finally, a suite of applications run on the workstations to provide a graphical interface to dispatchers, supervisors, system administrators, and other personnel. An overview of the Fixed End processes is shown below:

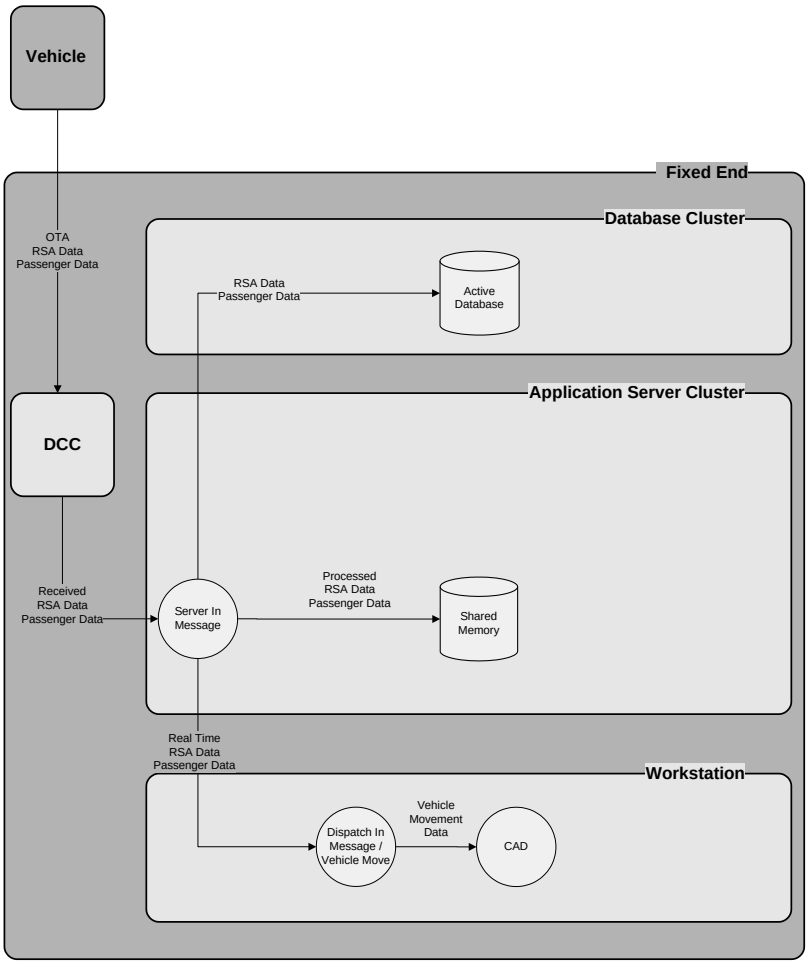

Figure 2: OrbCAD High-Level Architecture

An important aspect of the OrbCAD system is that it is highly flexible and modular, allowing customizable. Each OrbCAD system is custom configured for a particular customer. In addition, the infrastructure may be distributed across multiple sites throughout a city, or even across city/county lines. It is even possible for multiple transit agencies to share the same OrbCAD system.

In addition it is worth considering interprocess communication between Fixed End processes, since each machine runs many processes, and processes must communicate with each other both on the same machine and across machines. The existing OrbCAD architecture uses a proprietary TCP/IP communication library written in C/C++ called ZIPC. ZIPC works by establishing a message mailbox (essentially a queue) for each process. Messages may then be sent to and read from these mailboxes using the ZIPC API functions.

An important aspect of this approach is that it uses a “push” model for communication. That is, messages received from a vehicle will be processed by the DCC and send directly to the server/workstation processes, the latter of which forwards them directly to the

front-end CAD GUI. Although this approach provides excellent response times in the nominal case, it does not scale well. For example, large fleets will generate a constant volume of messages that must be processed by the entire system (including the front-end clients) in near real-time.

6 New Architecture

6.1 Overview

Given the performance requirements of the new Headway/Layover features, it was deemed that significant architectural changes would need to be made to support them. In particular, there was the desire to use a new architecture for these features that would be both more scalable and more flexible than the current OrbCAD architecture. This architecture is called OrbCAD Extended Edition (OrbCAD-EE).

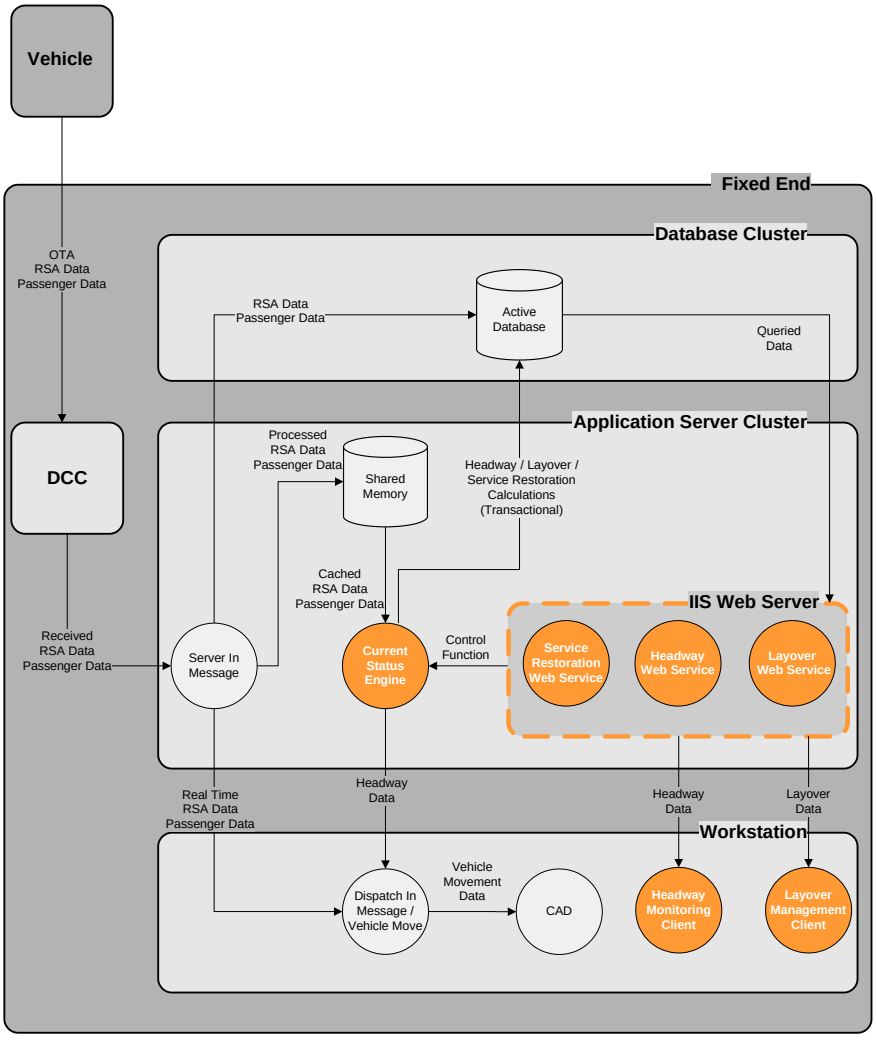

Figure 3: OrbCAD-EE High-Level Architecture (with EE components highlighted)

The diagram shown above presents a high-level overview of the OrbCAD-EE architecture. New modules are highlighted in the diagram to differentiate them from existing OrbCAD components.

An important aspect of OrbCAD-EE is the communication model, which is fundamentally different from that of OrbCAD. Communications between OrbCAD-EE components are performed using Web Services, two key aspects of which are that data is transferred using a client-based "pull" model and that communications are using an industry standard.

By using a pull model, OrbCAD-EE will scale gracefully as more vehicles are added to the fleet, because processing is centralized to server hardware that can be upgraded as-needed. The current OrbCAD client-server architecture farms processing out to client machines, which is less flexible and (for example) makes hardware upgrades more difficult.

By using an industry standard for communications, OrbCAD-EE can more easily integrate with other systems – for example, by exposing methods to be consumed by an application running on a customer's web server. In addition it allows future projects to leverage tools developed by outside vendors and the open source community. By contrast, the existing OrbCAD communication model is proprietary and must be maintained by the in-house development staff – as such it is inflexible and cannot be used for communication with other systems.

6.2 Data Flow

The following context diagram presents an overview of the functionality provided by OrbCAD-EE. As can be seen in the diagram, the three major functions provided by the OrbCAD-EE features are to view headway status, view layover status, and apply service adjustments functions. A vehicle entity is largely provided for context, as no changes were made to the vehicle software for this effort. However, as will be explained later on in the document, messages are sent to the vehicle when certain service adjustments are applied to a vehicle. And, of course, existing functionality enables status information to be automatically collected from each vehicle.

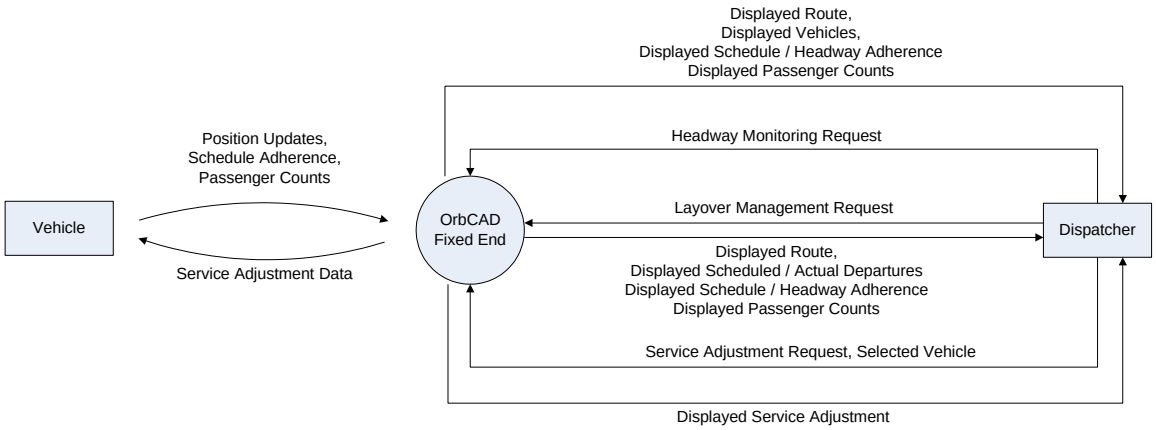

Figure 4: OrbCAD-EE Context Diagram

The Data Flow Diagram (DFD) Level 0 illustrated below provides a high-level view of the new OrbCAD-EE functionality. Again, the majority of the changes are provided for the dispatcher, and thus this is the primary entity in the diagram. Existing OrbCAD functionality is only depicted in the diagram when necessary, to simplify the diagram and to allow the new features to be explained as clearly as possible.

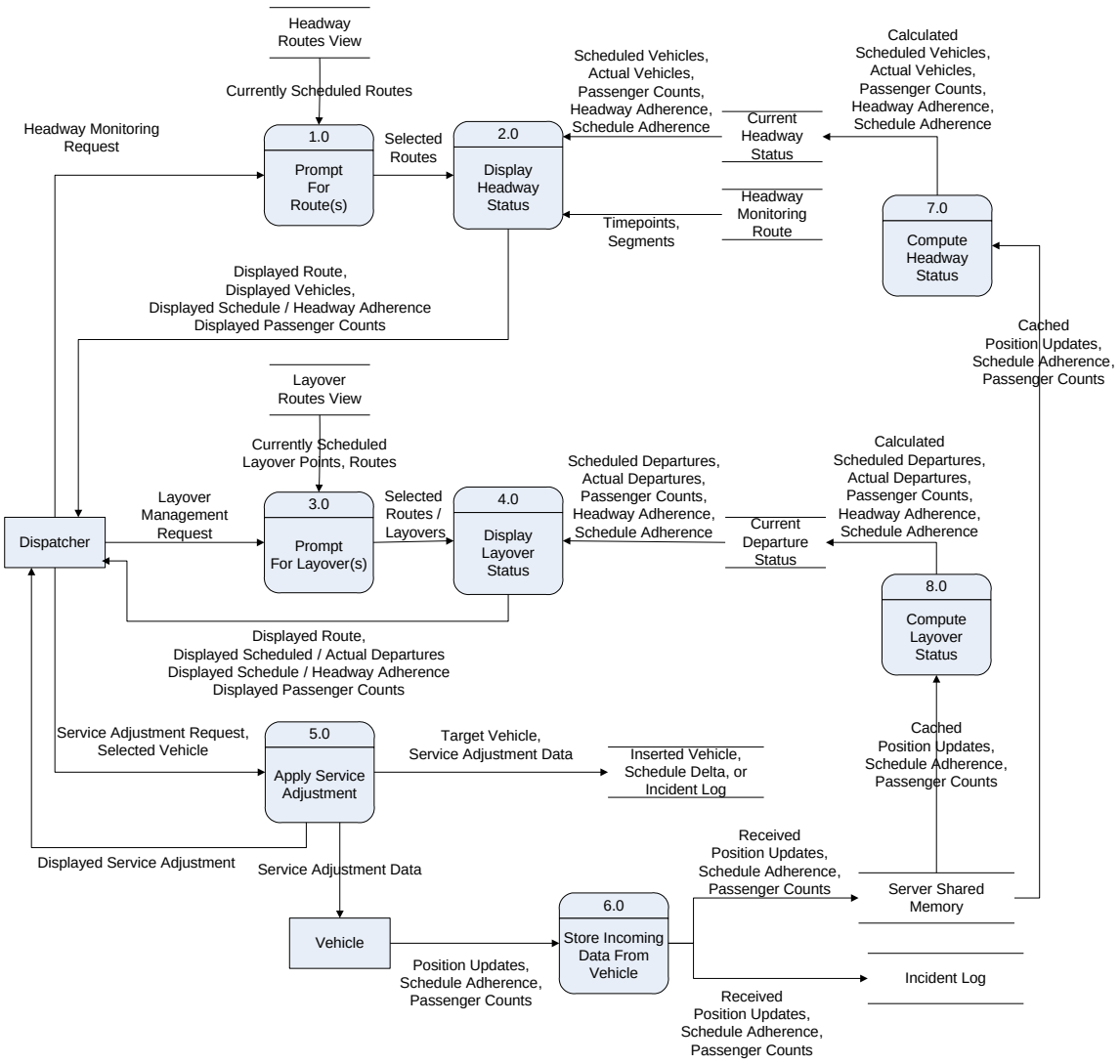

Figure 5: OrbCAD-EE Data Flow Diagram (Level 0)

Before diving into the details of OrbCAD-EE processes, we will first examine the existing vehicle functionality, depicted in process 6.0. As can be seen in the diagram, this process receives incoming data from the vehicle and stores it to the database and server shared memory (which acts as a high-speed data cache).

Let us next consider the headway display, depicted in processes 1.0, 2.0, and 7.0 of the DFD. As can be seen in process 1.0 of the diagram, a user initiates a request to display headway monitoring, and then selects one or more routes to view. Once this selection has taken place, the software uses the selected routes to return a set of static data describing the route and a set of dynamic data corresponding to the schedule / actual vehicles on that route. The static data defines the structure of the route – timepoints, detailed timepoint information, route segments, and associated information. Dynamic data contains both the scheduled and actual vehicles on the route, as well as the headway between them and detailed vehicle information (passenger counts, adherence status, etc). Once all of this

data is available, the route is rendered and displayed to the user by process 2.0. In order to make all of this possible, process 7.0 precomputes headway adherence. This eliminates an additional processing delay when rendering the route, but increases CPU and storage requirements on the machine performing the computations.

Processes 3.0, 4.0, and 8.0 describe the layover management functionality, which is similar to that of headway. In process 3.0 the user selects one or more layover points along with their corresponding route(s). Dynamic data is then retrieved from the departure (layover) status view and displayed on the client. All required static data has already been retrieved in 3.0, so no further static information is required. At this point, process 4.0 may render the layover management view and present it to the user. Again there is a back-end process, 8.0, which is constantly precomputing layover status data for display.

Finally, service adjustment functions are depicted in process 5.0. A level 1 DFD diagram of this process is displayed below. At a high level the user may requests a service adjustment function, which brings up a GUI specifying additional information. Once all information is present, the user may then apply the function, which is saved to the database, which initiates a change on the front-end display, and which optionally sends a message to the affected vehicle. All service adjustment functions are covered in more detail under the Service Restoration section of this document.

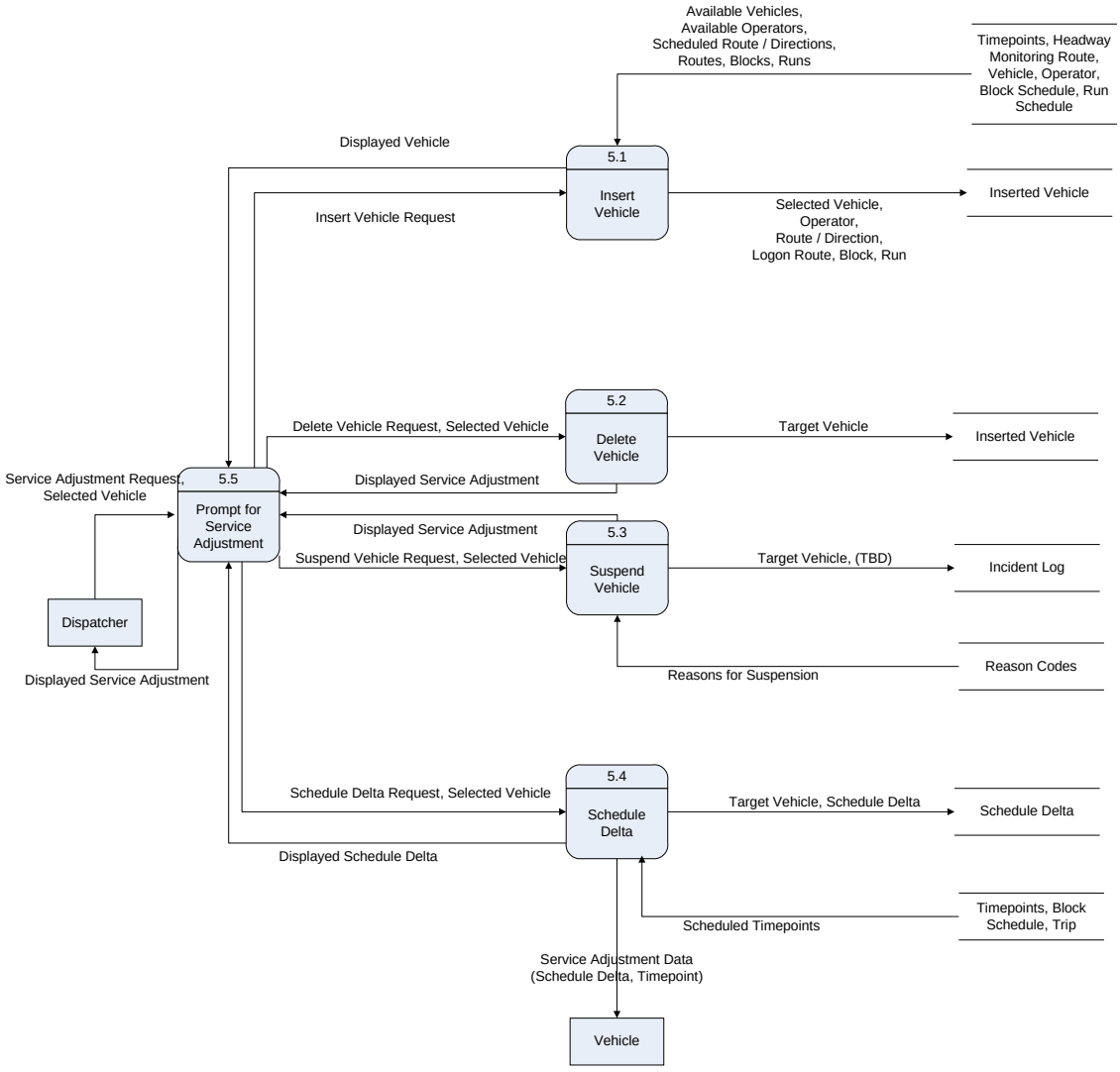

Figure 6: Service Restoration Data Flow Diagram (Level 1)

6.3 Tiers

OrbCAD-EE has essentially a 3-tier architecture composes of Smart Clients, Back-End processes, and the Web Services which connect them. The design section of this document covers the specific algorithms, GUI's, and actual processes in more detail. However this section is intended to introduce each set of components and set the context for each.

6.3.1 Smart Clients

Smart Clients are Graphical applications that run on the client workstations to display data to the end user. These applications are written as windows forms using the .NET Framework, and make use of third-party controls from Developer Express.
These controls provide more functionality than the build-in windows forms controls and helped to decrease development time.

6.3.1.1 Controller Services Framework

Headway and Layover both use a common framework that incorporates a Piccolo canvas along with a wide array of common objects and functionality for interacting with the canvas. The following UML sequence diagram illustrates the typical workflow of clients using this framework – although the headway application is presented, the workflow is representative of both the headway and layover clients, as well as any new clients using this framework.

The main components in the architecture are the User Interface (UI) thread, which contains Piccolo primitives responsible for drawing the canvas and providing user interaction. The Worker Threads are responsible for retrieving data in the background, without blocking the user interface – this improves the user experience by allowing the UI to remain responsive at all times. Finally, the Web Service provides a uniform interface for retrieving remote data.

First, during startup the user would request adding a new tab. The user would be presented with a dialog where they can select data for the new tab (a route, in this particular case). The user selects the data they would like displayed and then clicks OK to initiate a request by the worker thread to obtain data. The UI prompts the user to wait while the worker thread retrieves data from the web service. Finally, when data is available, it is cached in memory by the worker thread and passed to the UI for display. A similar control sequence occurs when the user changes the data displayed on a particular tab.

Data is kept up-to-date by periodic refreshes initiated by the worker thread. During such a refresh, data is retrieved at a regular interval by the worker thread, and then passed to the UI for display.

When a tab is closed, the worker thread is aborted. The thread frees resources while the UI closes the tabpage.

For a control function (used to pass an update to the OrbCAD system, EG: Insert Vehicle), the user selects the control function and initiates it, which updates the UI and triggers a worker thread to update memory and notify the external system via a Web Service call.

To change the zoom level, the user selects a new zoom level from the UI. The zoom level is then changed by the UI directly.

To request a manual refresh of the GUI, the user selects refresh from the UI. The UI then retrieves the latest snapshot of data from memory and immediately updates itself.

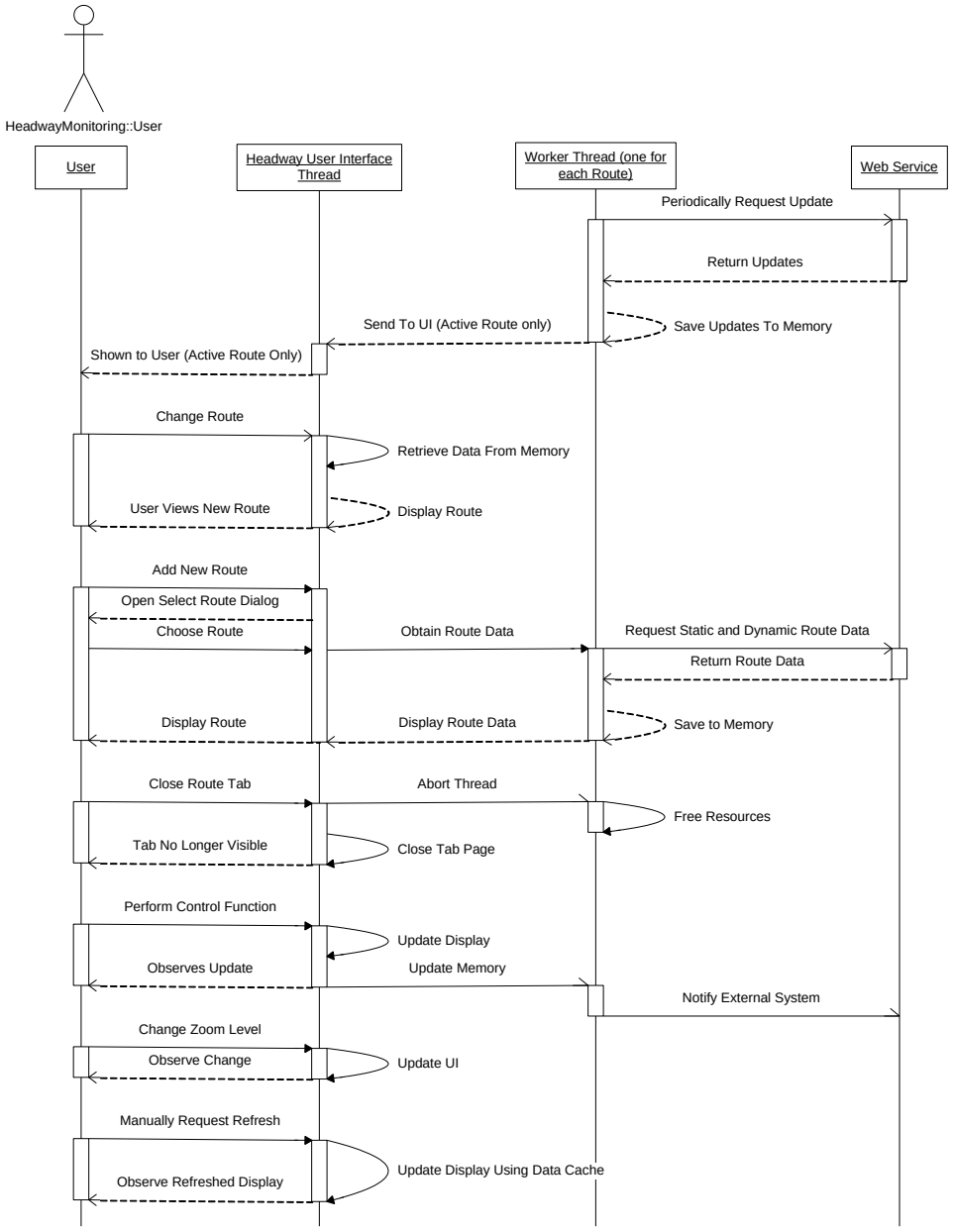

Figure 7: Sequence Diagram of Headway Monitoring

6.3.1.2 Periodic Updates

Since all client data is retrieved from web services, each client must call web service methods periodically. Unfortunately this presents a problem since Web Service methods are “expensive”, and may take a long time to complete, during which time the calling thread is blocked. Thus web service methods cannot be called directly from the UI thread.

In order to accomplish this task, a worker thread is created for each tab currently displayed. The thread shall periodically query the corresponding web service, return the latest snapshot of the route and apply the changes to the UI thread. The pseudo code is as follows:

```
void RouteWorkerThread (Route r){

        while (displayed ( r )) {

                // Query Web Service
                Dataset ds = RouteWebService ( r )

                // Update Data Cache
                Lock (mutex)
                RouteDataCache[r] = ds
                Unlock (mutex)

                // Update UI by
                // Invoking handler method on UI Thread
                if (IsSelected ( r ))
                        UI.Invoke (RouteUpdateHandler, ds)

                // Wait For Next Update Cycle
                Sleep (PollRate)
        }
}
```

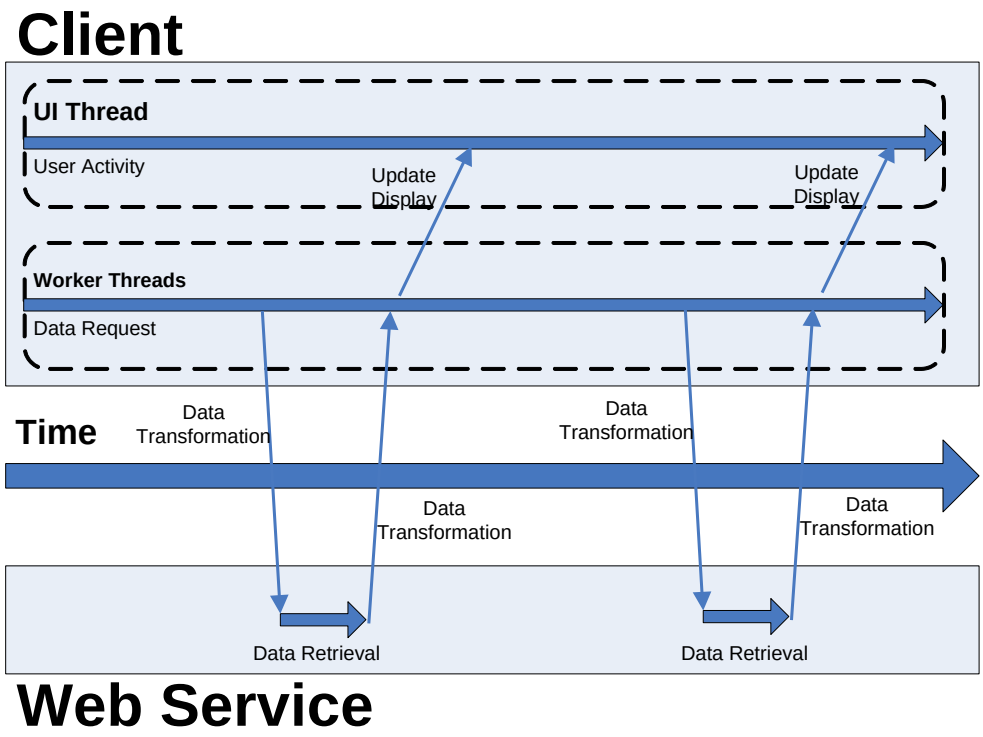

Figure 8: Smart Client Periodic Updates

The diagram above depicts updates made dynamically from the headway/layover smart clients via asynchronous web methods. As can be seen in the diagram, a worker thread will periodically request data from the web service, which retrieves data (generally from a database), transforms it to XML for transmission, and sends it back to the client, which unpacks the data and passes it back to the user interface to update the display. This model is similar to the asynchronous update model employed by AJAX-based web applications, and provides for a responsive, friendly end-user experience.

6.3.2 Server Components

A single new process, the Current Status Engine (CSE), is responsible for performing server computations for Headway, Layover, and Service Restoration.

The CSE is a multithreaded, .NET-based Server application written in C# that performs OrbCAD-specific business logic and data manipulation for use by OrbCAD-EE Smart Clients. The CSE consists of a .NET process running on the OrbCAD Server along with a suite of custom engine “plug-ins” for each EE feature

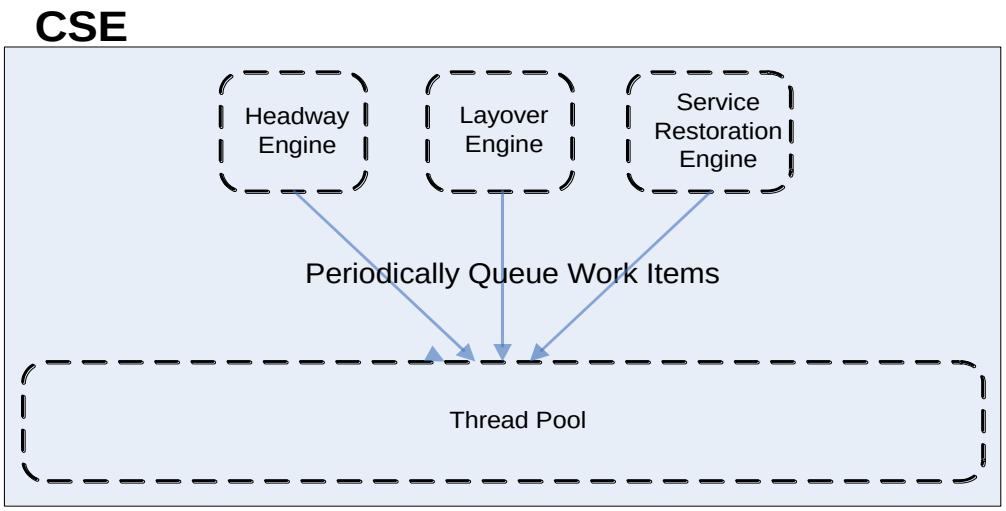

Figure 9: CSE Engines

At startup, this process loads other "plug-in" engines and runs them to perform any required processing. These engines include the Headway Engine, Layover Engine, Service Restoration Engine, and Service Summary Engine (not discussed in this paper). As can be seen in the diagram above, these engines periodically queue work items to the application thread pool, where they are executed.

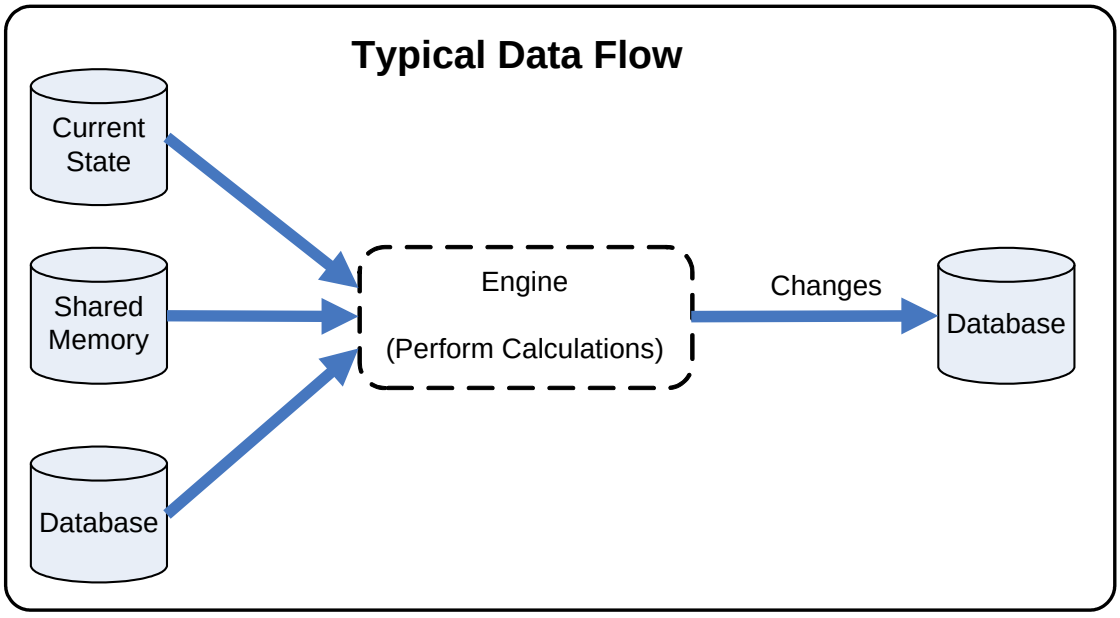

Figure 10: Overview of Typical CSE Engine

As depicted in the diagram above, these engines typically pre-compute data and store it to the database for use by the Web Services. Input data are typically from state variables, shared memory, and the database. State variables are stored internally by the engine and

provide a way of maintaining state between work item invocations. Shared memory is shared among all server processes and serves as a data cache to prevent expensive requests to the database. And finally, the database contains a wide array of information but is only accessed rarely to prevent it becoming a performance bottleneck.

In any case, once input data is collected, each engine performs custom processing and typically writes these changes back out to the database. Shared memory and the internal state may also be updated. Given that often there are dozens of threads constantly running, this architecture has potentially high CPU, thread, and Database resource requirements. However, the advantage is that this demand is centralized to a single server machine that can be upgraded as needed.

6.3.3 Web Services

Web services are used for communication between the OrbCAD backend and OrbCAD EE Clients. They provide an open, standard way of calling remote methods over HTTP and are one of the primary methods of interprocess communications in .NET

Upon a request initiated by a Smart Client, a Web Service will typically retrieve data written to the database by the CSE. Web services are provided for headway, layover, and service restoration. In addition a utility web service has been provided for generic data that may be requested by any application.

6.3.4 Database

6.3.4.1 Overview

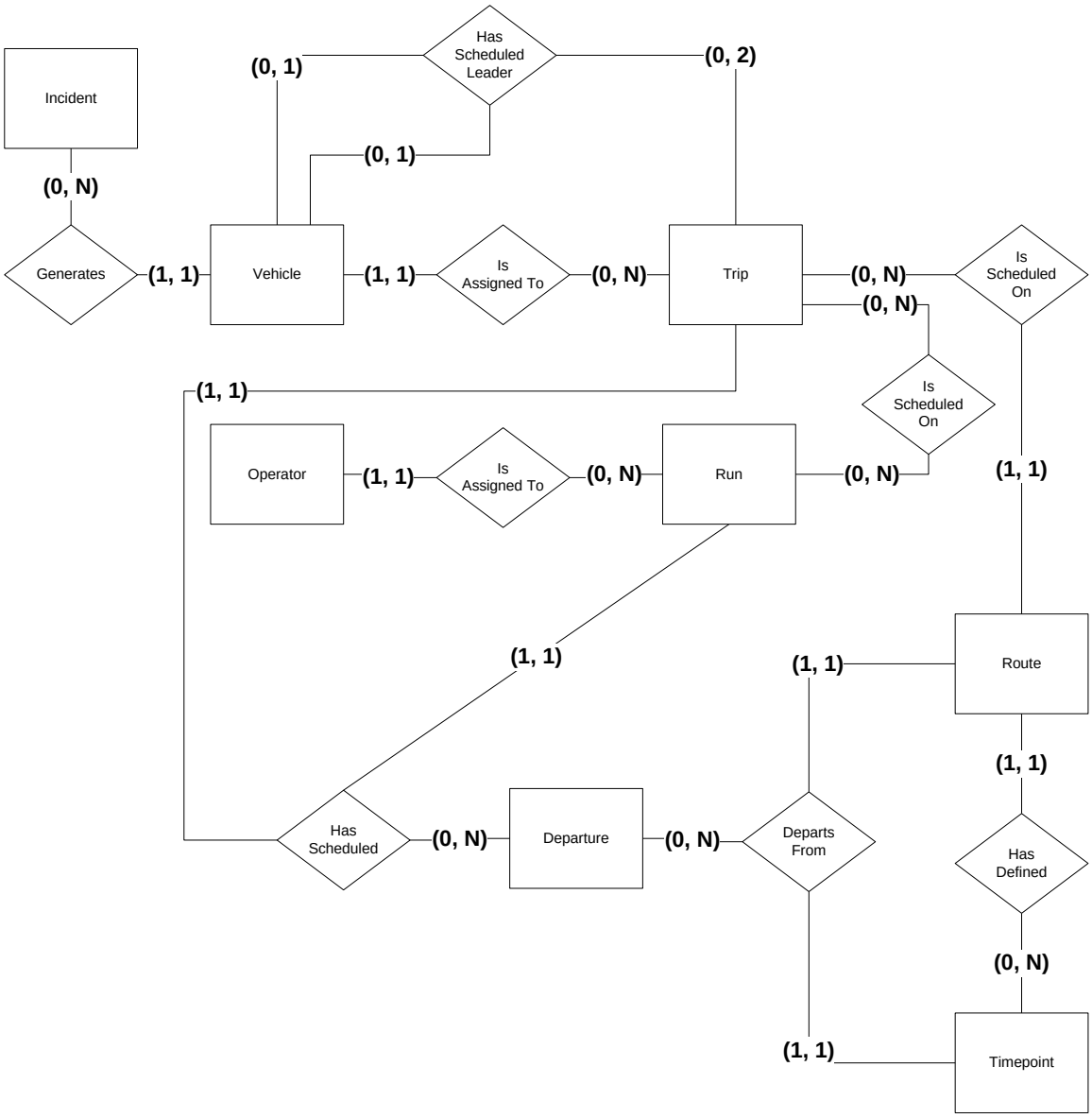

Figure 11: ER Diagram of the Headway / Layover Database

The ER Diagram shown above shows the entities used by both the headway and layover applications, and depicts the major relationships between them. Please note that this is only a high-level diagram and does not depict in detail the complex relationships between these objects that exist in the full OrbCAD product. However, this diagram is more than sufficient to describe things at a high level.

The entities relevant to Headway are Vehicle, Trip, Route, Timepoint, and Incident. Vehicles are of course the transit busses, and Trips are the trips that each vehicle is

scheduled to perform. As the diagram shows, each vehicle is assigned to many trips – this set of trips is typically referred to as a block of work performed by the vehicle on a given day. In turn, each trip is performed on a single route. And in turn, each route has multiple timepoints defined – these points are particular latitude/longitude locations along the route; when the vehicle reaches one it will send a message over the air to the Fixed End to indicate its progress along the route.

As the diagram shows, a leader relationship is defined between 2 consecutive trips, one of which contains the leader vehicle and the next of which contains the follower. This relationship is used by headway to determine the scheduled leaders along a route at any given time.

The diagram also depicts relationships relevant to layover management. Firstly, runs are the work an operator is scheduled to perform on a particular day. For example, an operator may be scheduled for a 4 hour run in the morning and another 4 hour run in the afternoon. An operator is assigned to one or more runs on any given work day. Since the main goal of layover management is to analyze vehicle departures, the departure relationship is very important to understanding the layover algorithms. As shown in the diagram, a departure has a scheduled trip and run – which have an assigned vehicle and operator, respectively. In addition, a departure will depart from a single route and timepoint – a central assumption to the layover application since the GUI prompts the user to select routes departing from a single timepoint.

6.3.4.2 Tables

This section briefly describes each of the tables used by the Headway and Layover applications. The following diagram shows the tables graphically and illustrates the foreign key relationships between them (excluding incident log, current departure status, and current headway status due to the complexity of their relationships). The sections after that list the columns of each table.

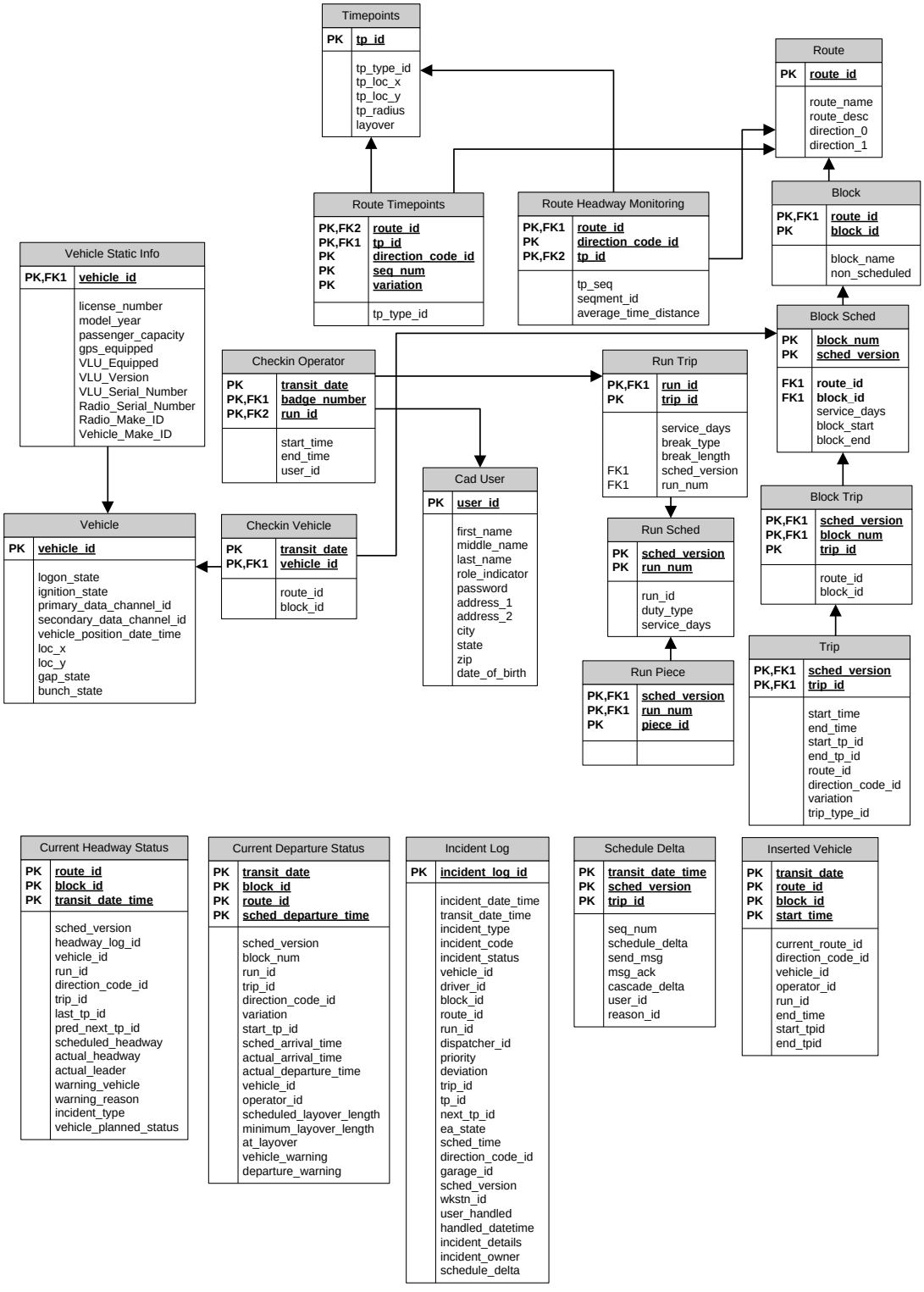

Figure 12: Database Tables used by Headway and Layover

6.3.4.2.1 Vehicle

The vehicle table contains dynamic information associated with a vehicle. Records are updated by system activity, as detailed in the individual field descriptions.

Field List

Name	Type
vehicle_id	int
logon_state	bit
ignition_state	bit
primary_data_channel_id	tinyint
secondary_data_channel_id	tinyint
vehicle_position_date_time	datetime
loc_x	float
loc_x	float
gap_state	bit
bunch_state	bit

6.3.4.2.2 Vehicle static info

The vehicle_static_info table contains static information associated with a vehicle.

Field List

Name	Type
vehicle_id	int
license_nbr	varchar (15)
model_year	smallint
passenger_capacity	smallint
gps_equipped	bit
vlu_equipped	bit
vlu_version	varchar (15)
vlu_serial_nbr	varchar (15)
radio_serial_num	varchar (35)
radio_make_id	tinyint
vehicle_make_id	tinyint

6.3.4.2.3 Timepoints

The timepoints table contains code to geographical coordinate information for timepoints in the system.

Field List

Name	Type
tp_id	int
tp_type_id	tinyint
tp_loc_x	float(8)
tp_loc_y	float(8)
tp_radius	smallint
layover	bit

6.3.4.2.4 Route

The route table contains code-to-text translation of routes defined in the system.

Field List

Name	Type
route_id	int
route_name	varchar (35)
route_desc	varchar (40)
direction_0	tinyint
direction_1	tinyint

6.3.4.2.5 Block

The 'block' table contains the defined resource data that is constant and changing status of all route/blocks defined in the system. A route/block is a unique logon identification number that defines the work that will be performed by a vehicle. Route defines the primary service area and route and block defines a particular schedule variation of the primary route.

Field List

Name	Type
block_id	int
route_id	int
block_name	varchar (35)
nonscheduled	bit

6.3.4.2.6 Block Sched

The 'block_sched' table contains statically defined schedule data that associates route/blocks to imported schedule versions. The combination of ‘sched_version’, ‘route_id’, ‘block_id’ create a unique key to the data in the table.

Field List

Name	Type
sched_version	tinyint
block_num	int
route_id	int
block_id	int
block_start	datetime
block_end	datetime
service_days	tinyint

6.3.4.2.7 Block Trip

Description

The ‘block_trip’ table contains statically defined schedule data that associates trips to imported schedule version route/blocks.

Field List

Name	Type
sched_version	tinyint
block_num	int
trip_id	int

6.3.4.2.8 Trip

The 'trip' table contains trip information for scheduled route/blocks in the system. Trips are either revenue, pullin, pullout or deadhead (non-revenue).

Field List

Name	Type
sched_version	tinyint
trip_id	int
start_time	datetime
end_time	datetime
start_tpid	int
end_tpid	int
route_id	int
direction_code_id	tinyint
variation	varchar (8)
trip_type_id	tinyint

6.3.4.2.9 Route_Timepoints

The 'route_timepoints' table contains all variations of route paths in the OrbCAD system. A route path contains a collection of timepoints, real and virtual, to be traveled in a certain sequence. It represents a physical path between two locations. Route paths are differentiated by route, direction, and variation (i.e. 21 Outbound AA). Route paths are downloaded to the vehicle and used for route adherence processing.

Field List

Name	Type
route_id	int
tp_id	int
tp_type_id	tinyint
direction_code_id	tinyint
seq_num	int
variation	varchar (8)

6.3.4.2.10 Cad_user

The 'cad_user' table contains statically defined resource data of all operational users defined in the system. Each user has a unique logon identifier that is validated by the system via this table. both CAD and vehicle users are in this table.

Field List

Name	Type
user_id	int
first_name	varchar (15)
middle_name	varchar (15)
last_name	varchar (35)
role_ind	smallint
passwd	int
address1	varchar (30)
address2	varchar (30)
city	varchar (30)
state	char (2)
zip	varchar (15)
date_of_birth	datetime

6.3.4.2.11 Checkin_Operator

'checkin_operator' table holds the most recent assigned work assignment (route/run) for the operator.

Field List

Name	Type
transit_date	datetime
badge_number	int
start_time	smallint
end_time	smallint
run_id	int

6.3.4.2.12 Checkin_vehicle

'checkin_vehicle' table holds the most recent assigned work assignment (route/block) for the vehicle

Field List

Name	Type
transit_date	datetime
vehicle_id	int
route_id	int
block_id	int

6.3.4.2.13 Current_Departure_Status

The Current Departure Status table stores the most recently calculated departure times for all vehicles scheduled to depart from an interchange in the next n (configurable) minutes, as calculated by the Layover Management Server.

Field List

Name	Type
transit_date	datetime
sched_version	int
block_num	int
block_id	int
run_id	int
trip_id	int
route_id	int
direction_code_id	int
variation	varchar(8)
start_tp_id	int
sched_arrival_time	datetime
actual_arrival_time	datetime
sched_departure_time	datetime
actual_departure_time	datetime
vehicle_id	int
operator_id	int
scheduled_layover_length	int
minimum_layover_length	int
at_layover	bit
vehicle_warning	int
departure_warning	int

6.3.4.2.14 Current_Headway_Status

'current_headway_status' table is a snap shot of headway information for all blocks in the system.

Field List

Name	Type
route_id	int
block_id	int
transit_date_time	datetime
sched_version	tinyint
headway_log_id	int
vehicle_id	int
run_id	int
direction_code_id	tinyint
current_route_id	int
trip_id	int
last_tp_id	int
pred_next_tp_id	int
last_tp_eta	datetime
pred_next_tp_eta	datetime

scheduled_headway	int
actual_headway	int
actual_leader	int
warning_vehicle	bit
warning_reason	varchar (80)
incident_type	smallint
vehicle_planned_status	smallint

6.3.4.2.15 Incident_Log

'incident_log' table is a historical log of all normal and adherence type incidents generated in the system. Normal incidents are generated from a vehicle and either indicate a operator request or an anomaly. Adherence incidents are generated by either the vehicle or server and indicate schedule/route adherence status.

Field List

Name	**Type**
incident_log_id	int
incident_date_time	datetime
transit_date_time	datetime
incident_code	smallint
incident_type	smallint
incident_status	tinyint
vehicle_id	int
driver_id	int
block_id	int
route_id	int
run_id	int
dispatcher_id	int
priority	smallint
deviation	smallint
trip_id	int
tp_id	int
next_tp_id	int
ea_state	tinyint
sched_time	datetime
direction_code_id	tinyint
garage_id	tinyint
sched_version	tinyint
wkstn_id	tinyint
user_handled	int
handled_datetime	datetime
incident_details	varchar (250)
incident_owner	int
schedule_delta	smallint

6.3.4.2.16 Inserted_vehicle

The ‘Inserted Vehicle’ table contains entries for vehicles that have been inserted into the Headway Monitoring or Layover Management display. Entries in this table will only affect these displays – no other areas of the system are affected by entries in this table.

Field List

Name	Type
transit_date	datetime
current_route_id	int
direction_code_id	int
vehicle_id	int
operator_id	int
route_id	int
block_id	int
run_id	int
start_time	datetime
end_time	datetime
start_tpid	int
end_tpid	int

6.3.4.2.17 Route_Headway_Monitoring

The 'route_headway_monitoring' table contains route headway information. Route headway information defines the inbound and outbound sequence of timepoints. These timepoints are sequenced chronologically for display purposes and include all variations. This table is used for the Headway Monitoring Display only.

Field List

Name	Type
route_id	int
direction_code_id	tinyint
tp_id	int
tp_seq	smallint
segment_id	int
average_time_distance	int

6.3.4.2.18 Run_Piece

The 'run_piece' table contains statically defined schedule data that associates pieces to imported schedule version runs.

Field List

Name	Type
sched_version	tinyint
run_num	int
piece_id	int

6.3.4.2.19 Run_Sched

The 'run_sched' table contains statically defined schedule data that associates runs to imported schedule versions and to garages. The combination of ‘sched_version’ and ‘run_num’ create a unique key to the data in the table.

Field List

Name	Type
sched_version	tinyint
run_num	int
run_id	int
duty_type	varchar(8)
service_days	tinyint

6.3.4.2.20 Run_Trip

The ‘Run Trip' table binds each run in the database to a specific set of trips.

Field List

Name	Type
run_id	int
trip_id	int
sched_version	tinyint
break_type	smallint
break_length	smallint

6.3.4.2.21 Schedule_Delta

‘schedule_delta’ contains the most recent data for each transit_date_time/trip that has a schedule delta associated with it. The table is populated via the schedule delta GUI.

Field List

Name	Type
transit_date_time	datetime
sched_version	tinyint
trip_id	int
seq_num	int
schedule_delta	smallint
send_msg	bit
msg_ack	bit
cascade_delta	bit
user_id	int
reason_id	smallint

7 Design

7.1 Headway Monitoring

Headway Monitoring provides dispatchers with a graphical overview of the headway between all vehicles on one or more routes. The Client GUI incorporates functionality required for Singapore Phase 2 Headway Monitoring and Service Restoration. The Headway Client enables a dispatcher to quickly detect headway issues – it is then the responsibility of the dispatcher to make adjustments and/or corrections using functionality provide both by the Headway Monitoring GUI (Service Restoration Adjustments) or the rest of the OrbCAD system (Detours, Logon to Specials, etc).

Using Headway Monitoring, a dispatcher is able to identify headway problems on any route being displayed. For example, if two vehicles have a scheduled headway of 10 minutes but the actual headway is 20 minutes, the acceptable headway for the route has been exceeded. The dispatcher would be alerted to the problem via a warning color and large gap between vehicles on the Headway Client, at which time he may choose to take a corrective action, such as to have the lead vehicle slow down.

Although Headway Monitoring provides many advanced features, its basic functionality was similar to that of the existing CAD Graphical Headway GUI. To provide a robust solution, existing code for Graphical Headway was selectively ported from the existing CAD code to a new .NET based application.

7.1.1 User Interface

7.1.1.1 Overview

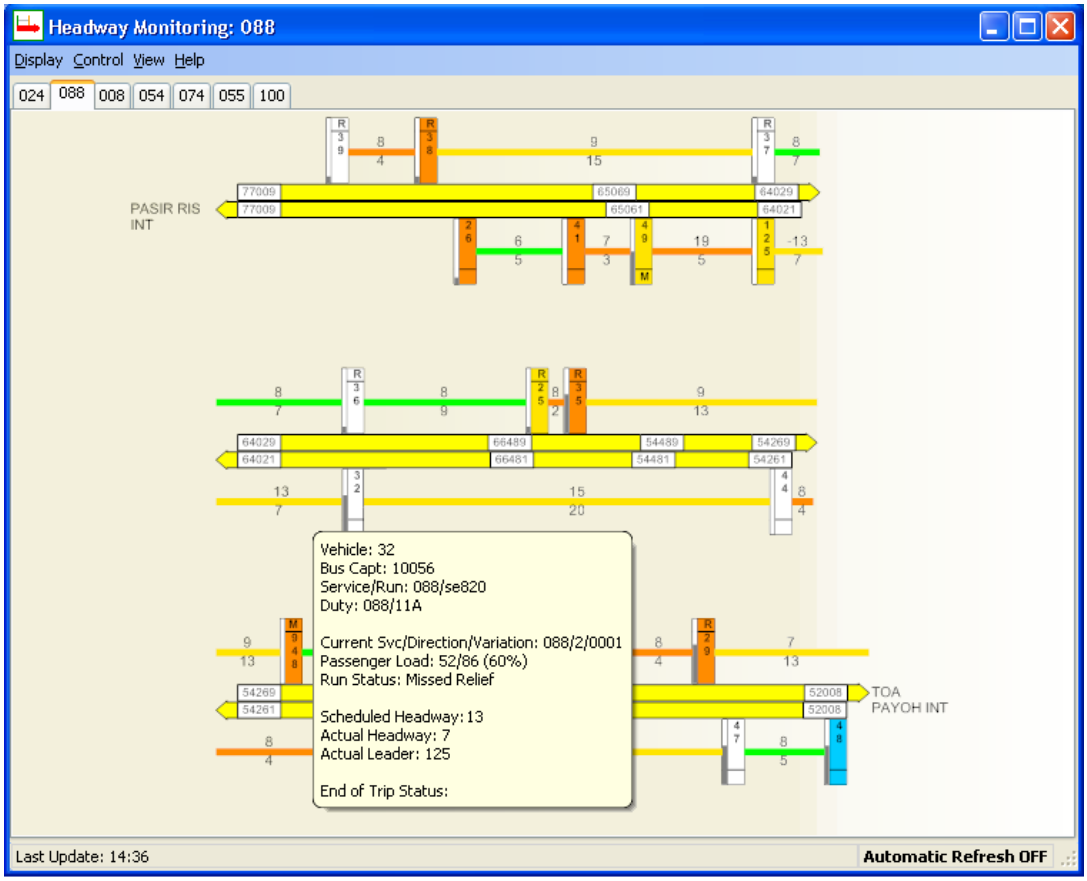

Figure 13: Headway Monitoring Client

The intent of the Headway Smart Client is to provide a graphical view of the headway between vehicles on a route. To this end, a single display will contain the route along with the vehicles performing work on that route. A route may consist of one or two directions, and is noted on the GUI by the line with an arrowhead (indicating the direction). Boxes are displayed within each route line to indicate timepoints. Timepoints are crucial to the headway back-end algorithms and as such are described in detail later in this document – for now they may be considered a subset of the bus stops along the route.

Vehicles are displayed on the GUI at their actual location on the route. As a visual aid, lines are displayed between all vehicles to indicate the actual headway between the vehicles – numbers are also provided for the scheduled headway. There are three possibilities for vehicle headway – vehicles may be meeting headway (normal), have too large of a headway between them (gapping), or too small of a headway (bunching). Headway adherence is between two vehicles and is calculated with respect to the leader vehicle.

If a vehicle is an invalid status (for example, No GPS) or logged off of the system, it will be displayed at its scheduled location with a smaller icon. This indicates that there is a problem, but allows the dispatcher to see what should be happening out on the road – the rationale being that most of these "warning" vehicles are still out on the road performing work.

Routes and vehicles are displayed on the GUI using particular colors, which are used to represent the current status. For example, a red route indicates that the critical threshold has been exceeded – a dispatcher probably needs to take corrective action in this case. Status colors are described in detail later in this section. The user may also mouse-over any object on the GUI to display a tool tip containing detailed status information.

The user may open additional routes, each of which will then be displayed in a new tab page. Each tab page will be labeled with its corresponding route description. Also, the title bar text of the Headway Monitoring window will change to reflect the name of the currently selected route.

The Headway Smart Client conforms to the standard application presentation and behavior of the windows environment. Headway Monitoring also conforms to OrbCAD GUI display standards, including support for Display Name Translation of property specific terms. This means that different words may be displayed on the GUI depending upon which transit property the application is configured for. For example, Singapore uses the term "Service" instead of the based-product term "Route".

Wherever possible, the user interface has been written generically, allowing configuration options to be placed in a local XML-based configuration file so that a customer may tweak them if a need arises.

7.1.1.2 Canvas

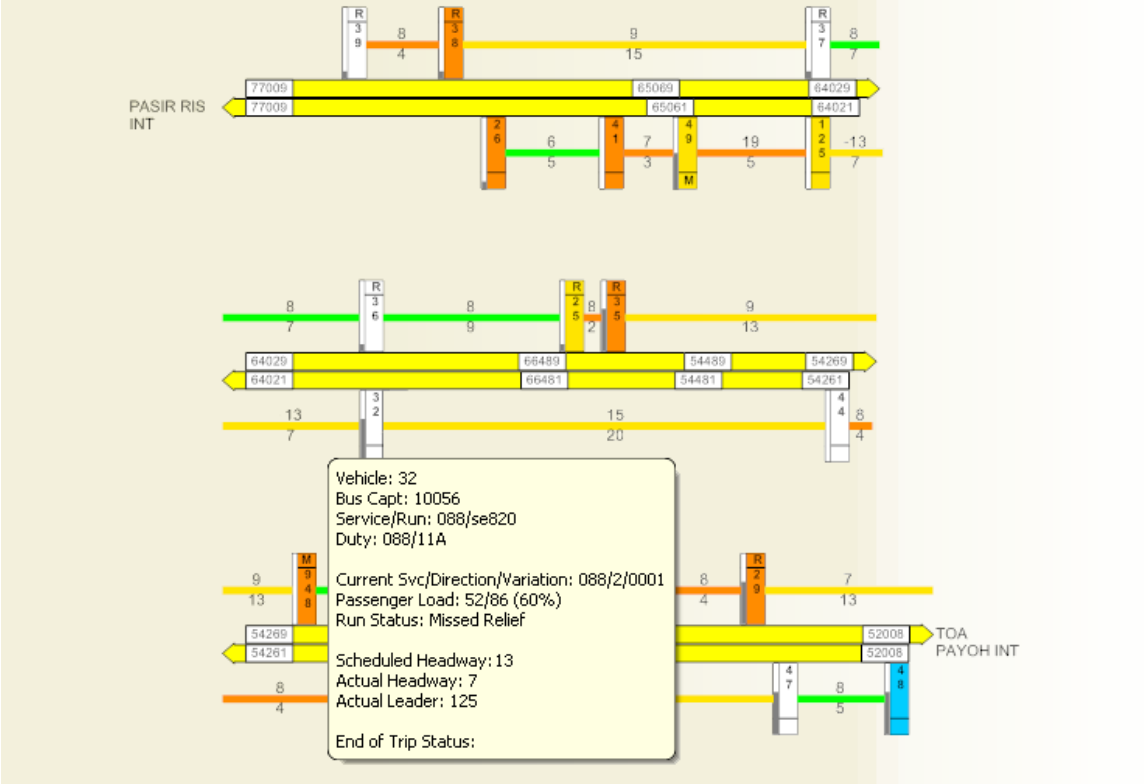

Figure 14: Headway Monitoring Canvas

The most distinctive feature of the Headway Monitoring Client is the canvas, which displays a snapshot of the current vehicle activity on a given route. Since the primary role of this GUI is to provide a view of the current headway between vehicles on a route, the client has been designed to convey this information in the most intuitive and simple way possible.

Again, given that the display monitors vehicle headway, the distance between timepoints and vehicles is time-based on the GUI. This allows for the most natural display of the information, since if the distance were based upon actual mileage, distances could be misleading. For example, a vehicle can typically travel 10 miles of a highway much faster than it can traverse 10 miles of busy downtown streets.

The distance units are calculated based upon the average time distance between timepoints for all route variations. Thus distance will remain constant for all hours of the day. The existing Graphical Headway display computes distances on the fly based upon the distance between timepoints for all active variations. The disadvantage of that approach is that the route lines will shrink and expand as the distance between timepoints changes. For example, the time between timepoints increases during peak hours (such as the morning rush hour) and decreases during off times (such as midnight on a Sunday). The decision was made not to follow this approach so that the GUI will be more

consistent to the user. This is especially important since a route may be divided into multiple horizontal segments.

Finally, the canvas consists of a PCanvas object, which is a basic primitive in the Piccolo framework. A PCanvas is a windows forms control, and is required for displaying objects using Piccolo. All objects in the Piccolo framework are rendered by the PCanvas control that contains them – without a canvas Piccolo could not display any objects to the user.

7.1.1.3 Graphical Objects

All graphical objects on the canvas are rendered using the Piccolo graphics framework. Each object is defined as a Piccolo object responsible for drawing itself and handling any defined user interactions. Headway Monitoring uses the following types of objects:

7.1.1.3.1 Route

Figure 15: Route as Displayed on the Headway Monitoring GUI

One or more Routes may be displayed at once from a single Headway Monitoring GUI. Within a single instance of the GUI, each route will be displayed in its own tab.

Each route will be drawn in one or two directions, depending on the underlying schedule data. Typically routes will have 2 directions, however a circular (looping) route may only have a single direction. Any such routes will have only a single direction displayed on the GUI – the other direction will be blank. A line, arrowhead, and corresponding text label will be used to specify each direction of the route.

The user may mouse-over a route to display textual Route and Direction descriptions.

The Headway Status of each direction of the route will be displayed using the same status types and color scheme as the Route Status on the Service Summary GUI. The line/arrow for each direction will be painted using the same Status color. The server will calculate the headway status using the same algorithm as Service Summary, and the GUI will simply reflect the status that it receives.

A route will be split into multiple segments as defined in the database:

1. Each Segment will have a unique starting and ending timepoint
2. Each Segment of the route will be displayed on its own line. All lines will be of the same length on the GUI.
3. System-level route segments will be determined at Schedule Import time

4. Each user may configure custom route segments (see Menu Items above).

In order to draw each route segment line with the same distance, times may need to be adjusted for lines that are shorter than others (IE, do not take as much time as other segments). Thus, it is important that any headway calculations are based upon the average time distance between timepoints, as provided by the schedule data, and not from time distances on the GUI. It is not expected that any calculations would be made based upon data from the GUI, but rather from actual timepoint ETA data stored in memory for each vehicle.

7.1.1.3.2 Timepoint

Figure 16: Timepoint as Displayed on the Headway Monitoring GUI

1. Timepoints will be displayed across each direction of the route.
2. The distance between each Timepoint will be expressed as time, not physical distance.
3. To avoid unnecessary runtime overhead, Schedule Import will pre-compute scale factors for the distance between timepoints.
4. Timepoints will not be filled with a color corresponding to their Headway Status. They will always be displayed using the same color, which will be configurable via an XML configuration file.
5. Timepoints will display a system-configurable text label. The text will be configured to display Fare Stage for Singapore.
6. The user may mouse-over a Timepoint to display detailed information, including:
    a. Timepoint Name
        a. Fare Stage (If applicable)
        b. Timepoint Name

7.1.1.3.3 Vehicle

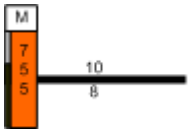

Figure 17: Vehicle (Block) Icon as Displayed on Headway Monitoring

Each vehicle (block) currently traveling on a displayed route will be displayed as an icon containing the following information:

1. Vehicle Text Label
    a. The vehicle text label will display the Vehicle ID, Block Alpha, or Operator ID depending upon the GUI's current vehicle text type setting. Warning Vehicles may not have a valid vehicle or operator ID, in which case no text will be displayed.
    b. To conserve space on the GUI, vehicle text labels will only display the applicable piece of text, and will not display a qualifying label such as "vehicle" or "route/block".
2. RSA Status – The vehicle display itself using the corresponding Performance Queue color depending on its RSA status.
3. Passenger Load – The vehicle will display a percentage full based upon its current load and maximum passenger capacity.
4. Planned Status at End of Trip – Planned Status is only displayed if one of the following cases apply:
    a. Meal Break
    b. Interline to New Route
    c. Last Trip for Block
    d. End of Service
    e. Relief
        i. Note that a relief could also occur mid-trip. In this case the icon will still be displayed in order to notify the controller that a relief is scheduled to occur.

    Each of these cases may be detected by looking at the underlying schedule data for the current / next trip.

A line connecting each pair of adjacent vehicles will indicate Headway Status. For vehicles at the start/end of a route, a line will be displayed between the vehicle and the first/last timepoint on the route. Scheduled Headway and Actual Headway will be displayed for each Headway Status line. Each time will be displayed in minutes. Each line will be filled with a color based upon the current Headway Status for the vehicle(s). Valid colors are Black (Normal), Yellow (Long), and Orange (Short) – these colors will match the colors in the database for Gap (Long) and Bunch (Short) incident types.

The user may mouse-over a vehicle to display detailed information via a balloon popup, including logon route, block, direction, variation, duty, headway, RSA, Badge Number, and Operator Name.

The user may mouse-over a line between vehicles to display detailed information about the headway via balloon popup. For example: "Scheduled Headway: 4 minutes / Actual Headway 8 minutes"

The user may select a group of vehicles by clicking or "rubber-banding" a selection rectangle over them. All selected vehicles will be unselected if the user clicks an unused area of the GUI while they are selected.

7.1.1.3.4 Vehicles With Warning / Error Status

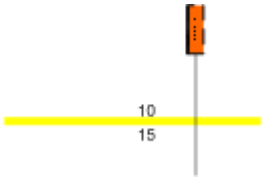

Figure 18: Vehicle (Block) Warning Icon as Displayed on Headway Monitoring

Vehicles may enter non-displayable states such as No GPS, Off Route, etc. These vehicles can no longer be displayed as normal vehicles on the GUI once they have entered such a state, because the fixed end will not receive PSA updates for them and hence cannot place them along routes on the Headway Monitoring display. Such vehicles will be displayed on the GUI using a warning icon instead of a standard vehicle icon, and headway status lines will not be drawn for them.

A warning icon will be displayed for a vehicle when any non-displayable condition applies, including: Off Route, No GPS, and No Response. Vehicles that are in layover will not be displayed on the GUI, but instead will be accessible via Layover Management (see below). Vehicles at the end of their scheduled work will not be displayed as warning icons, but will disappear from the GUI once they begin performing work on a deadhead trip.

A vehicle may also be displayed as a warning icon if a vehicle was scheduled for work during the current transit day, but the vehicle never logged on. In this case the vehicle warning icon will progress along the route according to its scheduled times.

When a vehicle is displayed as a warning icon, the regular vehicle icon will be removed and headway lines will not be drawn to the vehicle. The icon will be drawn above the other vehicle icons to prevent obscuring the rest of the GUI, and a semi-transparent line will be drawn from the vehicle-warning icon to the route line so that the controller may know the estimated position of the vehicle. Vehicle warning icons will disappear from the GUI when the vehicle logs off and/or completes work on the route (unless otherwise noted). The icons will move in accordance with the Headway Violation Incidents design.

The user may mouse-over a warning icon to obtain detailed information about the vehicle and its warning status. The following information will be provided, where applicable / available:

1. Warning Message / Reason – This will be a text description briefly explaining why a warning has been generated for the vehicle. For example, “No GPS for vehicle”.
2. Vehicle ID
3. Operator
4. Logon Route / Block

5. Run
6. RSA Status
7. Passenger Count
8. Passenger Capacity

Finally, warning icons will not be displayed for vehicles on Layover. These vehicles will not be displayed on Headway Monitoring – it is assumed that the controller would open the Layover Management GUI if they wish to obtain more information about vehicles currently on layover.

7.1.1.3.5 Color Schemes

This section specifies the color schemes used by various aspects of the Headway Monitoring GUI.

1) RSA Status

   Headway Monitoring will use the same colors for RSA status as the performance queue. To determine which color to use, a given vehicle's RSA Status will be used to look up the corresponding color in the Incident Attributes table.

2) Headway Status

   Headway Monitoring will use the same color scheme as the Service Summary GUI in order to display an overall headway status. At this time, the possible colors are Red (error), Yellow (warning), and Green (normal).

7.1.1.3.6 Scheduled Cutovers

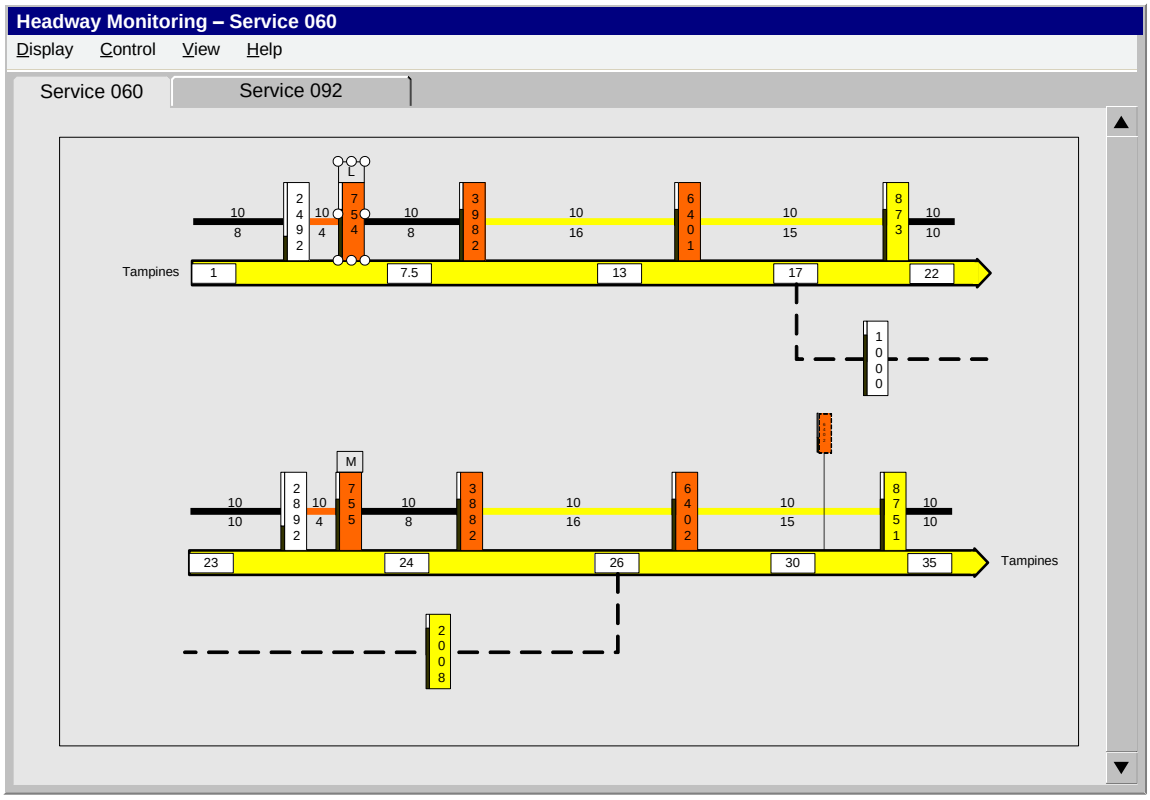

Figure 19: Display of a Scheduled Cutover

The Headway Monitoring GUI will display scheduled cutovers per the diagram above. A separate line will be displayed connecting the source/destination timepoints involved in the cutover. Note that this may be the same timepoint, but in the opposite direction and hence further down the route.

Vehicles currently on a cutover will be removed from the main route line, and will progress along the dotted cutover line. Once these vehicles reach the destination timepoint and the cutover is complete, they will once again be placed back on the route line. Headway calculations will not be performed between vehicles on a Scheduled Cutover.

7.1.1.4 Menus

7.1.1.4.1 Main Menu

Figure 20: Headway Monitoring Main Menu

The Headway Smart Client provides a set of menu options for the user. The following sections described each option in detail.

7.1.1.4.2 Display Menu

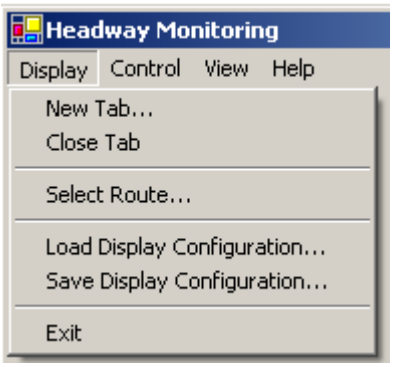

Figure 21: Headway Monitoring Display Menu

The display menu contains a set of basic Headway Monitoring operations.

New Tab allows the user to open another route in a new tab page. When it is selected, the Select Route GUI will be opened. Once a route has been selected, a new tab will be created and the route will be opened in the new tab page. If the user cancels out of the Select Route GUI, a new tab will not be added.

Close Tab will close the current tab page, and unload all data associated with that tab page. If no tab pages are open the menu item will be grayed-out and will not be selectable.

Select Route will bring up the Select Route GUI and allow the user to adjust the Routes for the current tab page.

Load Display Configuration will allow the user to open a saved configuration. The configuration will be loaded in the current tab page and will overwrite all information for that tab page.

Save Display Configuration will allow the user to save the current tab page's configuration. All details for the tab page will be saved, including Interchange, Routes, zoom level, and the automatic refresh setting.

Exit will terminate the current instance of Headway Monitoring.

7.1.1.4.3 Control Menu

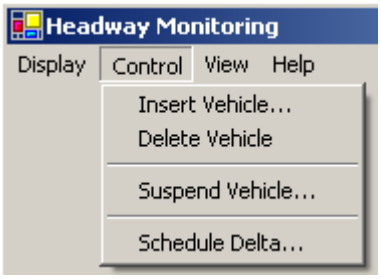

Figure 22: Headway Monitoring Control Menu

The control menu allows the user to perform service adjustments on the GUI. Each of these service adjustment functions is discussed below in the Service Restoration section.

7.1.1.4.4 View Menu

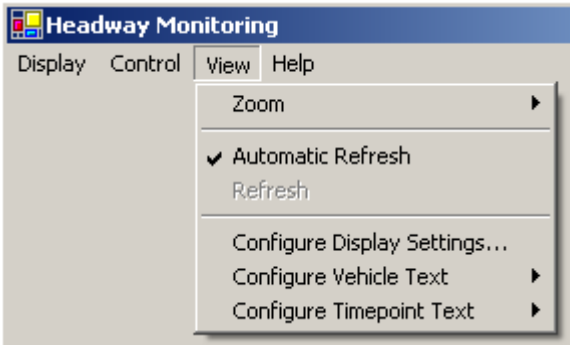

Figure 23: Headway Monitoring View Menu

The view menu allows the user to change various display aspects of the Headway Monitoring GUI. The current setting of each sub item in this menu will be saved as part of the GUI configuration, and will be restored when the user loads a previously saved configuration. Each of these settings applies only to the current tab page (route), unless otherwise noted.

Zoom allows the user to specify different zoom levels. By default, the following zoom levels are shown (in order): To Fit, 400%, 200%, 150%, 100% (default), 75%, and 50%. A checkbox is shown next to the active zoom level. For zoom levels that cannot display the entire route, the user may pan the display. Horizontal and Vertical scrollbars are attached to each headway canvas to support panning.

If a specific zoom level percentage is selected, as the GUI is resized the active route will maintain its same size – the route will not be scaled up/down accordingly. Instead, scroll

bars will indicate that the user may scroll to see the rest of the route. However, if zoom to fit is enabled, the display will be resized so that all graphical objects are displayed on the canvas – as the GUI is resized these objects will be scaled up or down in order to remain as large as possible on the display. This functionality is achieved using Zoomable user interface technology provided by the Piccolo framework.

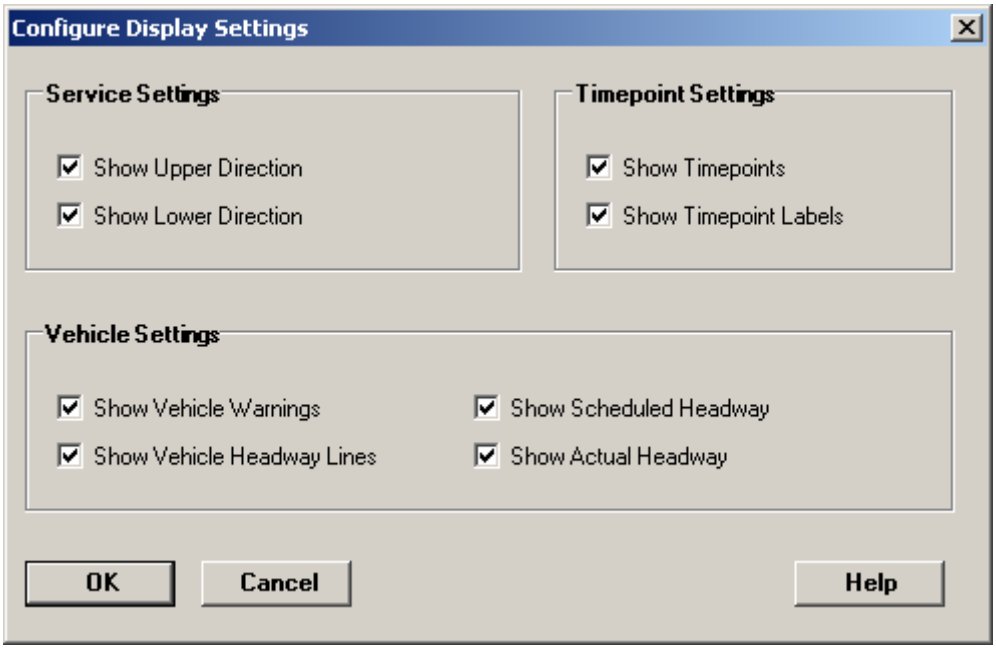

Figure 24: Configure Display Settings GUI

Change Display Settings allows the user to disable various objects on the Headway Display. Clicking the menu item will open the Configure Display Settings GUI, which allows the user to configure the objects displayed on Headway Monitoring. The GUI will display a set of checkboxes and allow the user to turn display elements on or off based upon the state of the corresponding checkbox. The following display elements may be enabled or disabled:

1. Route Directions
2. Timepoints
3. Timepoint Labels (at end of route)
4. Vehicles
5. Vehicle Headway Lines
6. Scheduled Headway Labels
7. Actual Headway Labels

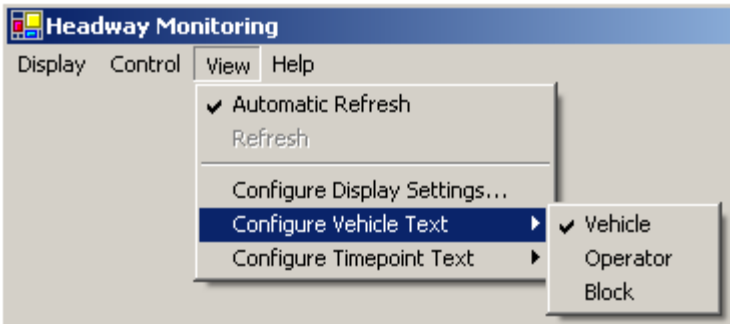

Figure 25: Configure Vehicle Text Submenu

Configure Vehicle Text allows the user to toggle the vehicle text labels between vehicle id, block alpha, and operator id. Clicking the menu item will bring a submenu containing each option. Only a single text label type may be selected at any time, and the active text type will be denoted by a checkmark. Vehicle text will default to Vehicle ID.

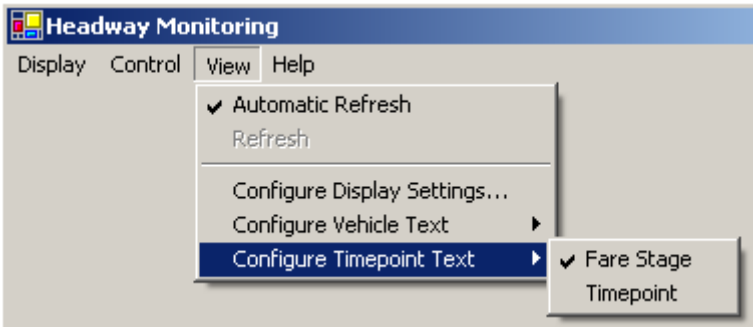

Figure 26: Configure Timepoint Text Submenu

Configure Timepoint Text allows the user to toggle the timepoint text between fare stage, and timepoint short name. The menu will behave in the same manner as Configure Vehicle Text, and the timepoint text will default to fare stage.

Automatic Refresh enables and disables automatic refreshing of Headway Monitoring. A checkbox will be displayed next to the menu item, and automatic refreshing will be enabled / disabled based upon the state of the checkbox. By default, Headway Monitoring will periodically refresh itself to update the position and status of all vehicles on the GUI. The refresh is automatic and occurs every 30 seconds (this is configurable). Note that this menu item will affect the refreshing of all routes displayed on the current instance of Headway Monitoring. Access to the automatic refresh menu item will be controlled by a classmark – users without the classmark will not have the option of disabling automatic refresh.

Refresh manually refreshes the GUI. This option will only be available if manual refresh is unchecked.

7.1.1.4.5 Help Menu

Figure 27: Headway Monitoring Help Menu

When the user selects the Headway Monitoring Help menu item, the Headway Monitoring help file will be displayed.

The "About Headway Monitoring" menu item will open the About dialog box, which will contain copyright and version information, as well as a large application icon.

7.1.1.4.6 Context Menu

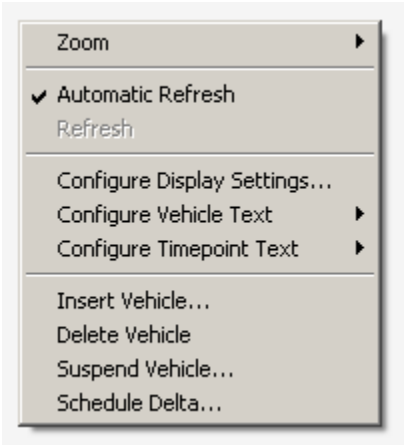

Figure 28: Headway Monitoring Context Menu

Finally, the user may right-click on the GUI to display the context ("popup") menu shown above, which contains a subset of the menu items already discussed. The context menu is provided as a convenience to the user.

7.1.2 Business Logic

7.1.2.1 Predicted Next Timepoint

When calculating vehicle headway, predicted next timepoint is used instead of the actual next timepoint. This decision was made in order to support the ability to automatically progress a vehicle passed a timepoint (based on its PSA value) even though a timepoint encounter has not yet been received for the timepoint. In this case, Headway Monitoring must know the time distance to the vehicle's predicted next timepoint in order to place it at the proper percentage distance along the line.

One of the key reasons for this decision was that certain cities use large vehicle poll rates that may not make it practical to use the actual timepoint encounters received by the fixed end. For example, with a poll rate of 5 minutes, a vehicle may take up to 5 minutes to report an encounter. This makes it impractical to rely upon the accuracy of this information for the headway display.

The calculation of predicted next timepoint uses the last PSA (predicted schedule adherence) value reported by the vehicle. This value is the number of minutes the vehicle is deviating from schedule (eg 1 = 1 minute early and -2 = 2 minutes late). And, importantly, the PSA is reported very frequently by the vehicle – so it is much more likely that this value will be up-to-date on the Fixed End as opposed to the last timepoint encounter. In any case, the predicted next timepoint is determined by simply taking the vehicle's PSA value (if available), adding it to the current time, and searching the list of timepoints (in order) to find the first timepoint that has an ETA of greater than this value. This timepoint is the next the vehicle is predicted to arrive at.

7.1.2.2 Leader and Follower

Headway calculations are performed with respect to 2 vehicles, not just one. Consequently each vehicle will be considered either a leader or follower. The leader is the vehicle is just ahead of the follower in the route path the follower is going to take. This is the actual leader – the scheduled leader is the vehicle that is scheduled to be the next one ahead of a follower along the same route path.

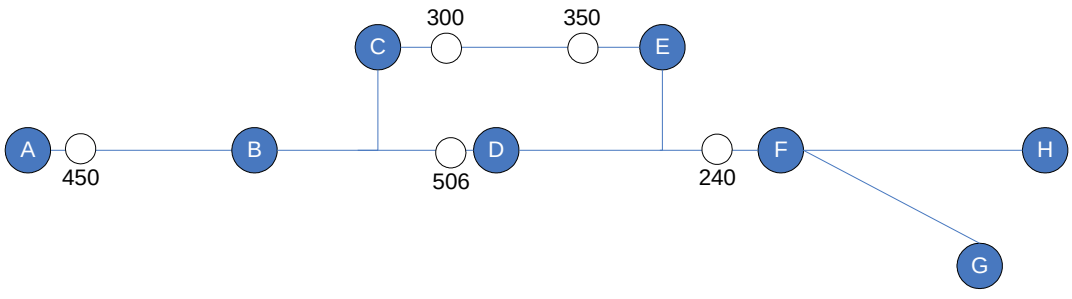

Figure 29: Leader / Follower Example

Consider the diagram above. If vehicle 450 is going to take the route path BCEFH, then 300 is its leader. If 450 goes through BDFH, then 506 is its leader

506 and 214 are headed to F. 350 will eventually come to F. In this snapshot, 214 is the leader of 506. Once 350 comes and joins the main variant behind 214, then 350 will become 506's leader.

Actual Leader and Scheduled Leader can be different. For example, if a vehicle is NORESP (no communications) or Off Route, headway will not be calculated for that vehicle. In this case actual leader would be calculated to be the next vehicle along the route path.

And of course it is possible that a vehicle does not have a leader (examples: first vehicle on the route, vehicle at end of schedule, deadheading vehicle). When a vehicle does not have a leader, it cannot get charged with gap or bunch (as explained in the next section).

7.1.2.3 Bunching and Gapping

In addition to showing headway deviations on Headway monitoring, the server will generate headway incidents for cases where the actual headway is outside of configurable thresholds.

A vehicle having an excessively large headway with respect to its leader is said to be gapping, and likewise a vehicle with an excessively low headway is said to be bunching with its leader. Note that these headway violations are always calculated with respect to the vehicle's leader – a convention chosen during development.

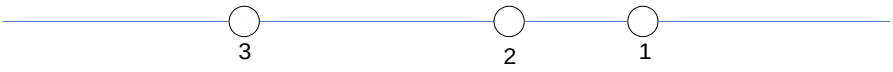

Figure 30: Example of Gapping and Bunching

For example, in the above diagram, vehicles 1 and 3 are on schedule but 2 is early. 1 and 2 are bunching and 2 and 3 are gapping. When server calculates headway for all the vehicles on this route, it will generate a gap incident for 3 and bunch incident for 2.

7.1.2.4 Headway Calculations

Given the previous definitions of and leader/follower vehicles and gapping/bunching incidents, the following equations will be used to calculate headway.

1. **Scheduled Headway** for the vehicle will be calculated using the following formula:

$$ScheduledHeadway = VehicleSchedTime - LeaderSchedTime$$

where:

Vehicle Sched Time = Vehicle's Scheduled Time @ predicted next timepoint

Leader Sched Time = Leader's Scheduled Time @ the same timepoint

2. **Predicted Time** for a vehicle at any timepoint is calculated using this formula:

   Scheduled Time @ TP – PSA

3. **Actual Headway** for the vehicle will be calculated using the following formula:

If the vehicle's predicted next TP is X,
Vehicle's Current Position = CurrentTime + Vehicle's PSA
Leader's Current Position = CurrentTime + Leader's PSA
Vehicle's Time to reach X = Vehicle's Scheduled Time @ X - Vehicle's Current Position
Leader's Time to reach X = Leader's Scheduled Time @ X - Leader's Current Position

Scenario 1:
If the vehicle and it's leader, both, haven't yet hit the vehicle's predicted next Timepoint (X), then actual headway is calculated this way:
Actual Headway = Vehicle's Time to reach X - Leader's time to reach X

Scenario 2:
If the leader has passed the vehicle's predicted next timepoint (that is, Leader's Time to reach X is negative) then actual headway is calculated this way:
If leader's predicted next TP is Y,
Leader's Time to reach Y = Leader's Scheduled Time @ Y - Leader's Current Position
Time Distance between X and Y = Leader's Scheduled Time @ Y - Leader's Scheduled Time @ X
Leader's distance from X = Time Distance between X and Y - Leader's Time to reach Y
Actual Headway = Vehicle's Time to reach X + Leader's distance from X

4. **Headway Deviation** is calculated using the following equation:

Headway Deviation = Scheduled Headway – Actual Headway

If this is a positive value greater than or equal to the bunching threshold, a BUNCH incident is generated. If this is a negative value greater than or equal to the gapping threshold, then a GAP incident is generated. Otherwise, vehicle headway is NORMAL and no incident is generated.

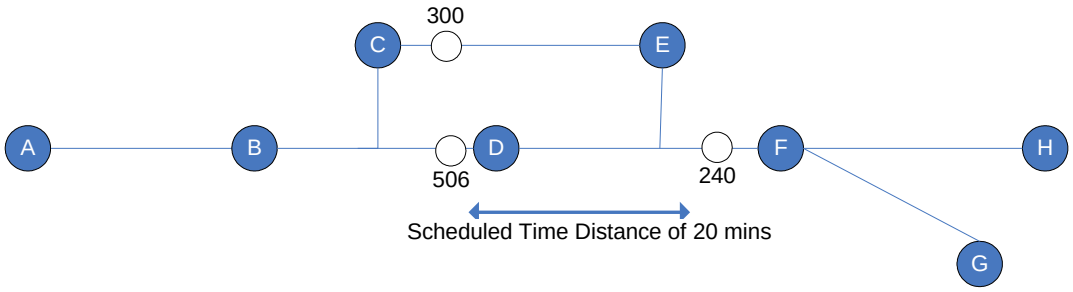

Figure 31

For example, the bunching threshold is 5 minutes. In the above diagram, stars are the timepoints and dots are the vehicles. Let's say 214 is the actual leader and 214 and 506 are 20 minutes apart as per schedule. Server predicts 506 to hit F at 8:15 and 214 to hit F at 8:10, then

Actual Headway = (8:15 – 8:10) = 5 minutes
Scheduled Headway = 20 minutes.
Headway Deviation = 15 minutes, which is greater than the threshold limit of 5 minutes.
So 506 will be charged with Bunching.

Note that:

1. In this design, bunching and gapping will work even if two vehicles switch positions (leapfrog). This is because, headway is determined by calculating headway between the vehicle and it's ACTUAL leader with respect to the vehicle's SCHEDULED headway.

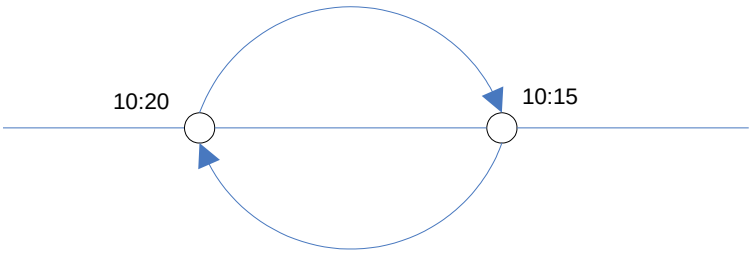

Figure 32

2. At layover points, vehicles will be bunching or gapping based on their arrival time to the layover area. But they will be changing to a LAYOVER status, so headway will not be calculated.
3. No bunching and gapping will be calculated for Deadheads (non-revenue trips).

7.1.2.5 Placement Of Vehicles on the Display

7.1.2.5.1 Calculating Time-Based Vehicle Position

Headway Monitoring calculates the time distance between a vehicle and its predicted next timepoint (as discussed earlier), and uses this time to plot vehicles on the display. Headway Monitoring uses the following factors to calculate a vehicle's position:

- Predicted Next Timepoint
- Predicted Next Timepoint ETA (with respect to the current time)

Headway Monitoring shall place vehicles on the GUI according to the time distance between a vehicle and its predicted next timepoint. Predicted next timepoint ETA is calculated by taking the scheduled ETA at the timepoint and adding the vehicle's Predicted Schedule Adherence (PSA) value. The PSA indicates the number of minutes early or late the vehicle is deviating from the schedule. For example, a vehicle with a scheduled arrival time of 2:00 and a PSA of –2 (2 minutes early) would have an ETA of 1:58.

Once the timepoint and ETA are known, the current location of the vehicle may be calculated by subtracting the current time from the ETA at the timepoint. Now that the time distance is known, it must be converted to units on the display before the vehicle can be rendered.

7.1.2.5.2 Converting Time to Display Units

One the headway monitoring display, time between timepoints is calculated as an average of all route variations in the schedule, so the actual distance between timepoints does not match the distance displayed on the GUI (see the next section for more details). To correct for this, the GUI must map from GUI coordinates to actual time. Headway Monitoring will use the timepoint ETA values from the schedule data to calculate the actual time distance between the timepoints. Then the ratio between the actual and displayed time distances will be used to place the vehicle at the proper percentage of the way between timepoints (in effect, mapping from the time domain to graphical coordinates). A side effect of this change is that vehicles will appear to move slower during peak hours (since there is more time distance between timepoints due to the other traffic on the roads) and faster during non-peak hours (since there is less traffic on the roads and busses may travel faster between timepoints) – however this was not considered a significant issue during development.

$$ScheduledTravelTime = NextTpETA - LastTpETA$$
$$DisplayedTimeDist = NextTpCoordinate - LastTpCoordinate$$
$$MinutesToNextTp = NextTpETA - (CurrentTime + PSA)$$

Equation 1: Travel Time Between Timepoints

$$\frac{MinutesToNextTP}{ScheduledTravelTime} = \frac{VehicleDisplayOffset}{DisplayedTimeDist}$$

Equation 2: Vehicle Display Offset

The series of equations labeled Equation 1 may be used to calculate the scheduled distance, the displayed distance, and the predicted number of minutes until the next timepoint. Equation 2 may then be solved to calculate the Vehicle's Display Offset, which represents the number of Minutes to the Next TP in terms of graphical units. The Headway Monitoring GUI may then place the vehicle this number of units before its Predicted Next TP.

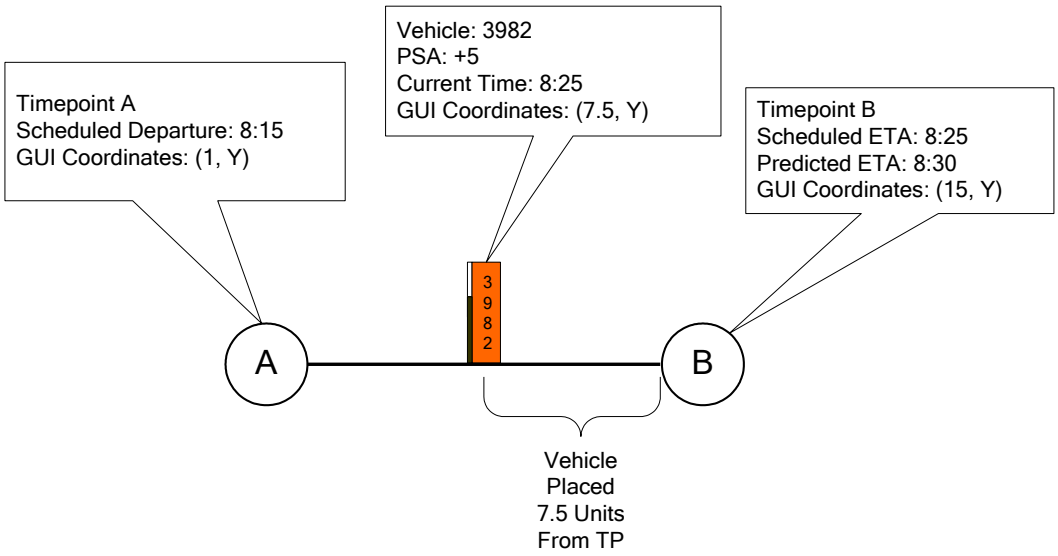

Figure 33: Illustration of Vehicle Placement Algorithm

For example: A vehicle last reported a timepoint encounter at TP A at 8:20 AM, its predicted next timepoint is at TP B, and it has a predicted deviation of 5 minutes late. The vehicle was scheduled to depart A at 8:15 AM and is scheduled to arrive at B at 8:25 AM. A and B are therefore 10 minutes apart, and if it were currently 8:25 AM the vehicle would be halfway between timepoints A and B. Given these conditions, the vehicle can

be projected to arrive at TP B at 8:30 AM. The current time is 8:25 AM so the vehicle still has 5 minutes to go before arriving at TP B. Therefore, the vehicle needs to be placed 5 minutes before B, halfway between timepoints A and B. If the number of Graphical Units between A and B is 15 (as a result of other variations on the same route, multiple route segments, etc), the vehicle will be placed 7.5 units (5/10 = x/15) before timepoint B.

Once the vehicle's position has been mapped from the time domain to graphical units, the vehicle may be placed on the display.

7.1.2.6 Average Time-Distance Between Timepoints

This section describes the algorithms for calculating the average distance between timepoints. These calculations must be made in order to determine the distances to display between timepoints on the GUI. The times from the schedule cannot be used directly because the distances tend to vary depending upon the time of day. For example, during rush hour the time between timepoints increases due to the volume of traffic on the roads, and the fact that more people are riding the bus during this time. But mid-day the time distances decrease because there is less traffic and fewer riders are taking the bus.

The algorithms described here were initially developed for a headway display originally used in the OrbCAD UNIX product developed by TMS. However, in that product the algorithms were run whenever a route was displayed, using the currently running route variations. Thus the algorithms were originally intended to calculate the average times between a set of timepoints for the same timeframe. As a result, the GUI would look different depending upon the time of day (as discussed above). In order to eliminate this effect, this design runs the algorithm a single time during schedule import, which typically is only done once a quarter. Thus all average-time computations are performed at import time, and the algorithm has been extended to look at all route variations for the entire schedule. This has the net effect of creating an overall average time between each timepoint on each route of the schedule.

Since the algorithm is now calculating average times for all route variations for all times of the day, the average time will not necessarily match the actual time distances between routes for any particular time of the day. The Vehicle display algorithm from the previous section gracefully handles the placement of vehicles and mitigates any negative impacts to Headway Monitoring.

7.1.2.6.1 Modified Average Time Distance Algorithm

The time-distance between any two pair of timepoints – the timepoint offset, is determined by using the following algorithm. Let us assume that there are **n** vehicles, B_1, B_2, … B_n, and **m** time points Tp_1, Tp_2, … Tp_m on the route. The schedule matrix for a particular schedule day is shown in the following table:

Vehicle Schedule				
	Tp_1	Tp_2	…	Tp_m
B_1	T_{11}	T_{12}	…	T_{1m}
B_2	T_{21}	T_{22}	…	T_{2m}
…	…	…	…	…
B_n	T_{n1}	T_{n2}	…	T_{nm}

Table 1: Vehicle Schedule Table (Vehicles Arriving at Scheduled Timepoints)

The timepoint offset, **O**, is defined as the difference between the expected arrival times at a pair of timepoints for a single vehicle for a given trip **Vt**. Thus, a vehicle schedule to arrive at Timepoint A at 10:05 a.m. and scheduled to arrive at Timepoint B at 10:15 a.m. defines the offset between Timepoint A and Timepoint B as 10 minutes. Accordingly, the definition of the timepoint offset becomes:

$$O_{i(i+1),j}\Big|_{t=T_{pj}} = T_{i+1,j} - T_{i,j},$$
$$i = 1,2,\ldots,m-1$$
$$j = 1,2,\ldots,n$$
$$t \in [Vt]$$

where:

O = Time Offset

T = Timepoint Arrival Time

Equation 3

Offsets may vary significantly throughout a schedule day and between day types. In order to present a consistent interface to the user, all variations for the booking will be examined and an average time offset will be calculated.

Finally, the average time offset for each pair of timepoints may be determined by calculating each time offset between the pair in the current booking, and taking the mean of the offsets:

$$mean\ O_{r \in R} = \frac{\sum_{b \in B} O_t}{N} \Bigg\} t \in [Tt]$$

O = time offset

R = set of time points

B = set of applicable time point pairs in current booking

N = number of applicable time point pairs

Tt = set of trips in current booking

Equation 4

7.1.2.6.2 Extension to the Algorithm to Handle Route Variations

This section outlines an extension to the algorithm that allows it to handle route variations containing different sets of timepoints. The previous algorithm can get into a bit of trouble depending on the schedule data. As described above, it is possible for the timepoints to get out of order. For example, consider the following schedule and its corresponding single line representation.

A	B	C	D	E	F
8:00	8:05	8:10		8:15	8:20
8:05	8:10	8:15	8:25	8:30	8:35
8:10	8:15	8:20		8:25	8:30

Table 2: Example Schedule Table (Vehicles Arriving at Scheduled Timepoints)

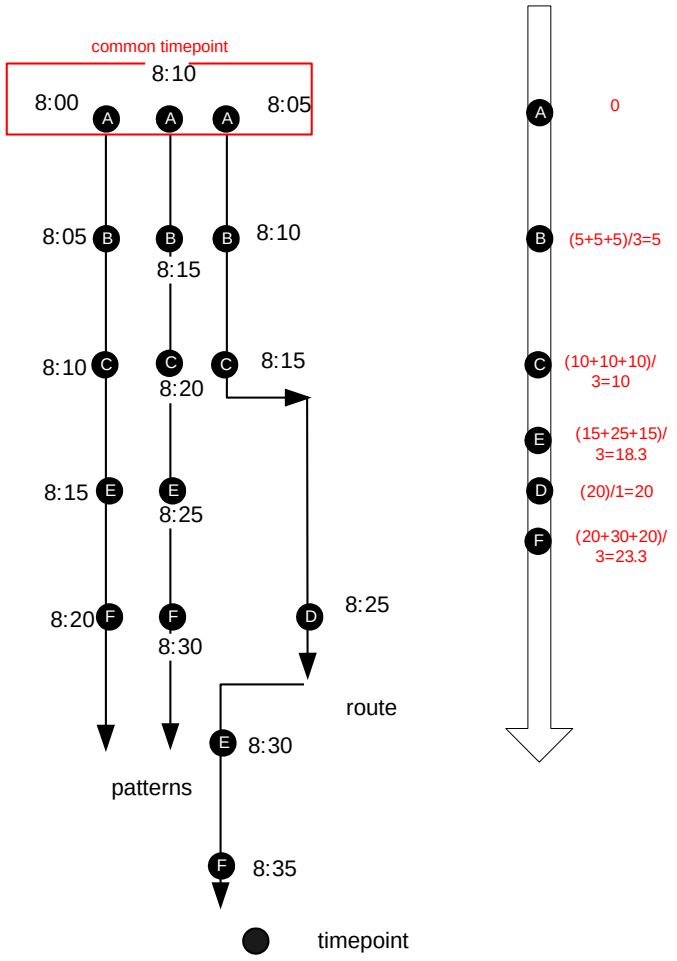

Figure 34: Timepoint Locations for Patterns on a Route (for sample data)

As can be seen in the figure above, a resulting single line representation of this route displays the timepoints in the order of ABCEDF, which is incorrect as timepoint D is displayed further down the line than E. Headway Monitoring needs to account for this situation and correct it. It can be seen that the timepoint D is only in one variation of the route. Performing a couple of calculations can relocate the timepoint.

1) Calculate the average time distance between the timepoint before and after the problem timepoint for all route variations.
2) Calculate the average time distance between the timepoint before and after the problem timepoint for only those route variations that contain the problem time point.
3) Calculate the ratio of the result of #1 over the result of number #2
4) Calculate the average time distance between the timepoint before and the problem timepoint for only those route variations that contains the problem timepoint.
5) Multiply the result of #4 and the result of #3.
6) Calculate the time distance from the start of the route to the timepoint before the problem timepoint for all route variations.
7) Add the result of #5 and #6. This result is the corrected time distance from the start point to where the problem timepoint will be relocated.

Applying these calculations to the example above yields:

1) The average time distance from A to C (for all variations) is 10 and the time distance from A to E (for all variations) is 18.3. 18.3 – 10 = 8.3 which is the average time distance between C and E for all route variations.
2) The average time distance from A to C (for all variations that contain timepoint D) is 10 and the time distance from A to E (for all variations that contain timepoint D) is 25. 25 – 10 = 15 which is the average time distance between C and E for all route variations that contain timepoint D.
3) 8.3 / 15 = .553
4) The average time distance from C to D (for all variations that contain timepoint D) is 10
5) .553 * 10 = 5.53
6) The average time distance from A to C (for all variations) is 10
7) 10 + 5.53 = 15.43

The following figure shows the single route line after these calculations. The timepoints are in displayed in the order ABCDEF, which is the correct sequence.

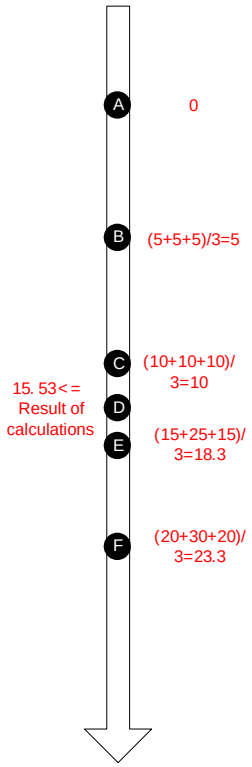

Figure 35: Timepoint Locations for Patterns on a Route

7.2 Layover Management

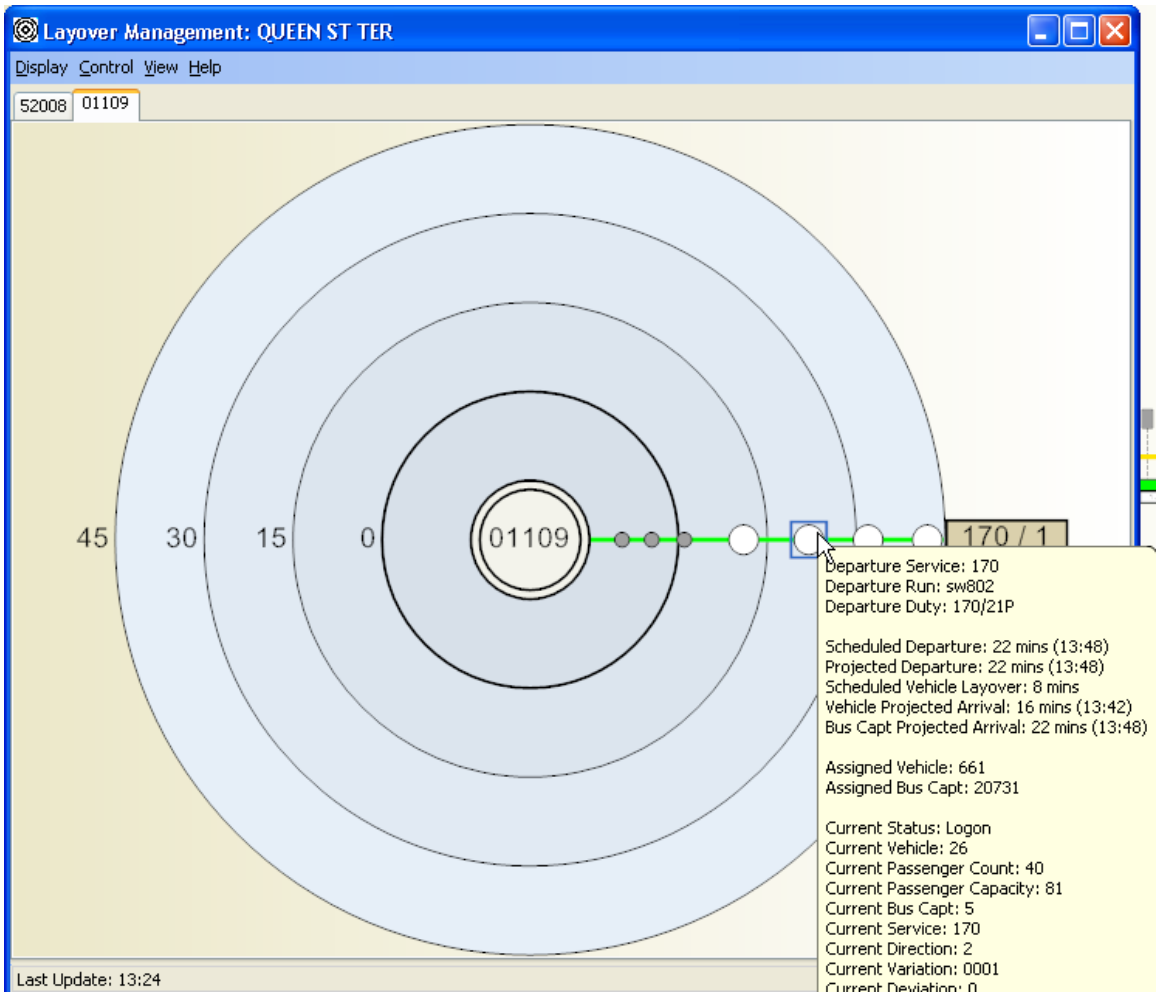

Figure 36: Layover Management Application

SBST's bus routes make large use of interchanges, which are essentially large transit centers placed throughout the city to serve as major hubs for travel across the city. Interchanges are one of the best places for Singapore to make schedule adjustments, because it is difficult for busses on the street to make significant schedule adjustments by speeding up or slowing down, but it is more practical for a bus to leave the interchange earlier or later. But in order to make these schedule adjustments, SBST needs a way to detect headway problems on routes departing from the interchange

The purpose of Layover Management is display the predicted headway of vehicles departing an interchange, in order to detect potential headway problems before they occur.

The layover management application graphically displays the predicted Departure Time for vehicles leaving an interchange on one or more selected services. It detects if operator layovers can be reduced to meet scheduled departure times, warns if trips will not be on time due to minimum required operator layovers, and allows schedule adjustments to vehicles scheduled to depart from an interchange.

Routes originating from an interchange are indicated by lines from the center Interchange to labels at the edge of the GUI. Each vehicle is plotted on one of these lines according to the number of minutes until its departure from the Interchange. As time progresses, the number of minutes until departure decreases and each vehicle on the display will move from the edge of the circle towards the center. Once a vehicle has departed from the center timepoint (IE, it has encountered the start of trip timepoint), it will be removed from the display. Such a vehicle would now be active on a trip, and may be inspected for any current headway problems via the Headway Monitoring GUI.

A dispatcher may detect future headway problems on a trip by observing gaps (or bunches) of the vehicles on the GUI. In such a case a corrective action may be taken; such as to request an operator take a shortened layover break.

For example, assume vehicle is scheduled to depart from Interchange A (center TP on the GUI) to perform work on Route B (at the edge of the circle). Furthermore, the vehicle is currently 10 minutes away from A and has an operator layover requirement of 7 minutes. Given these factors, the vehicle's estimated departure time from A is 17 minutes, and thus the vehicle would be displayed on the line from A to B, 17 minutes away from A. As the vehicle approaches A it will be moved closer to the center of the GUI, until it finally departs on Route B, at which time it is removed from the GUI. The vehicle is now actively performing work on this Route, and a dispatcher using the Headway Monitoring GUI or manually monitoring headway on the Performance Queue could now handle any further headway issues for the vehicle.

7.2.1 User Interface

7.2.1.1 Overview

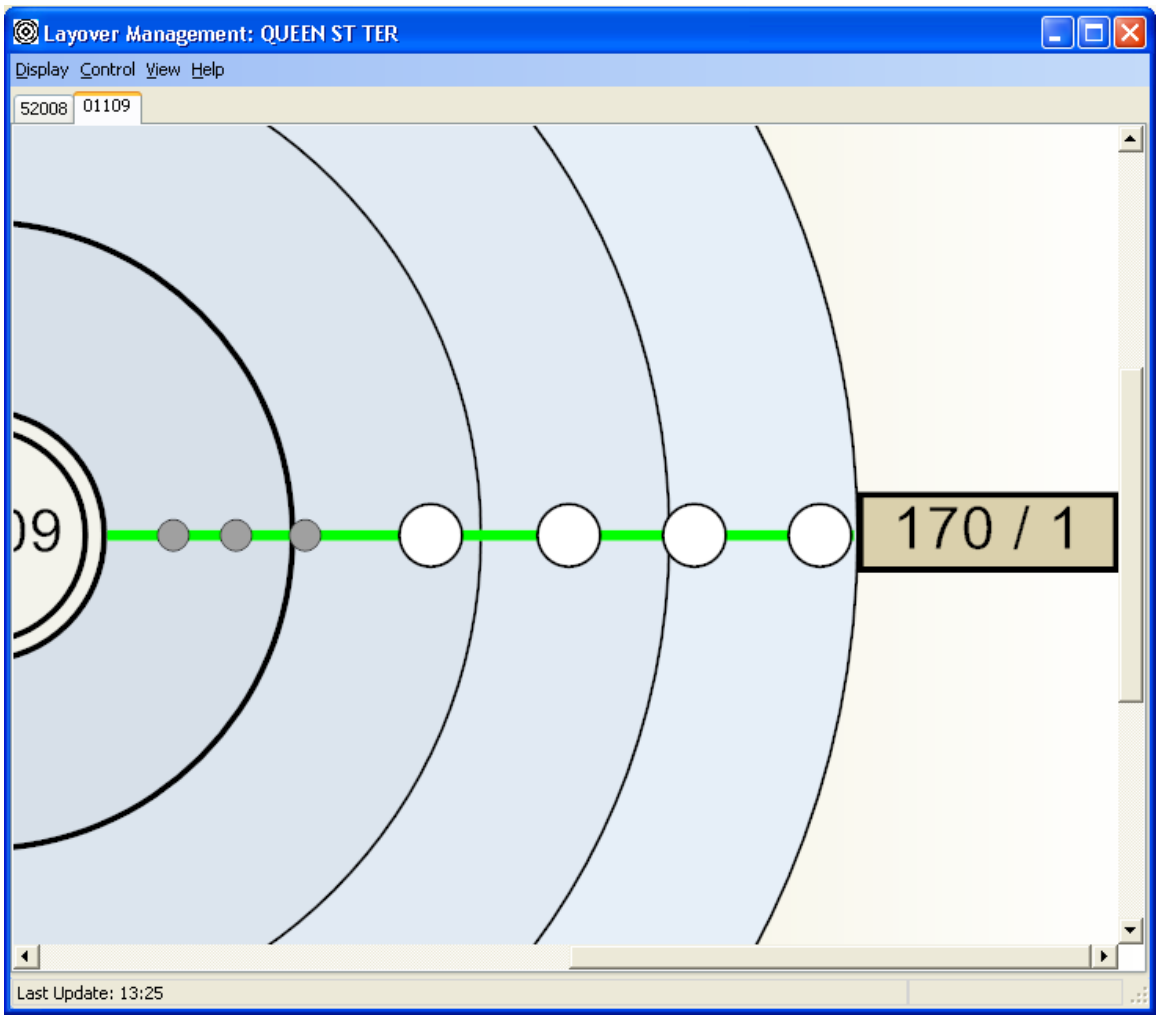

Figure 37: Layover Management Client

As shown in the picture above, the Layover Management application shows a single timepoint in the center of the screen. The GUI then shows lines representing routes that vehicles will depart on from this timepoint. In the example above, there are several evenly spaced departures that are scheduled to depart on Route 170 Direction 1 from Timepoint 01109[3].

Circles are drawn at regular intervals to indicate the estimated amount of time until vehicle departure. Note that the inner-most bolded circle indicates 0 minutes, meaning that assuming normal operations vehicles should have departed prior to reaching this line, although conditions such as traffic jams, an operator showing up late to work, or an operator not logging on to their in-vehicle unit would keep the vehicle on the display past the 0-minute mark.

[3] Singapore uses numbers for their timepoint names. However most other transit customers, such as those in the United States, would use an abbreviated street name here – for example: "5thMain" for the timepoint at 5th and Main.

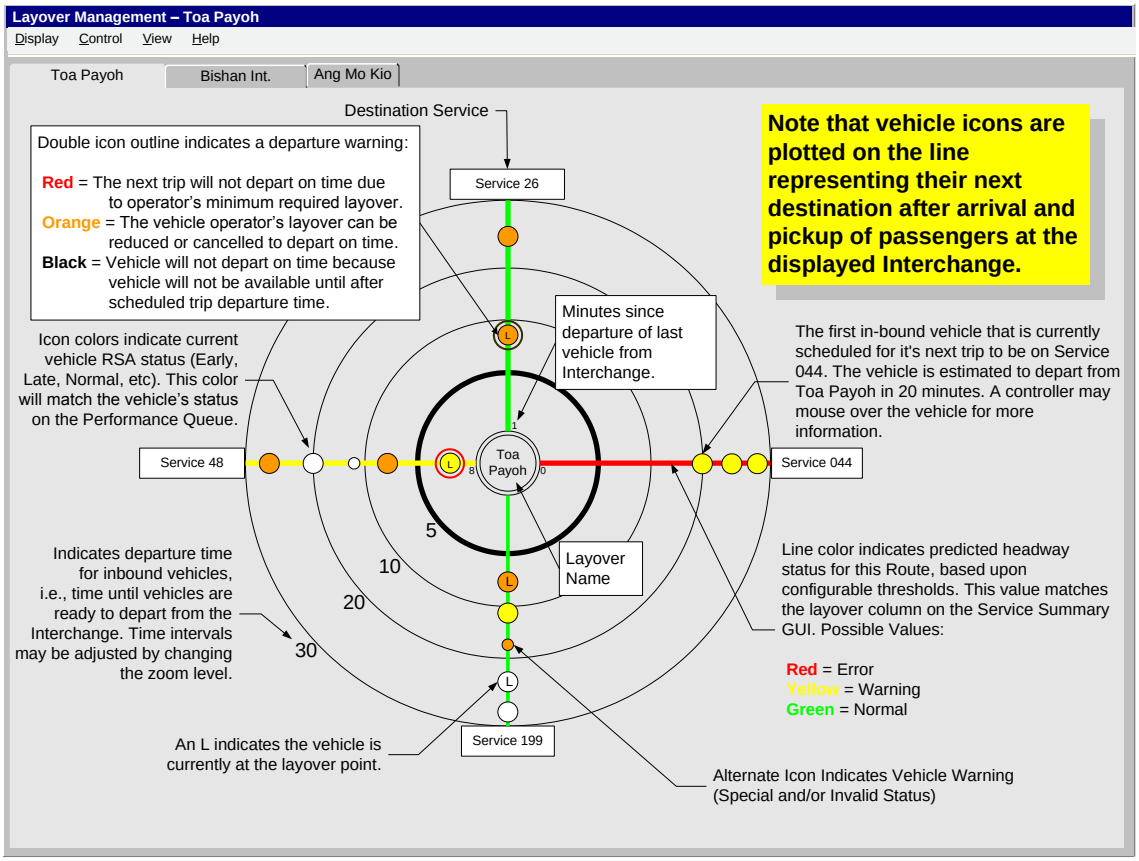

Figure 38: Concept Drawing of the Layover Management Application

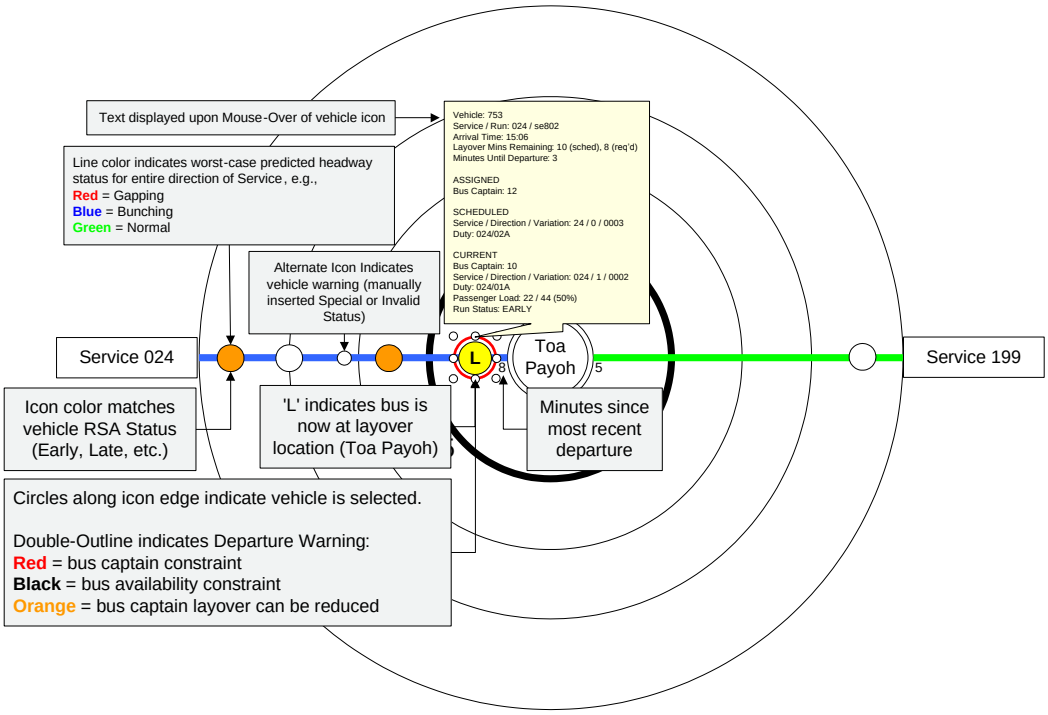

Figure 39: Concept Drawing of Layover Management Bull's-eye

The prototype diagrams above explain aspects of the Layover Management application in detail, including what information is displayed and how the user would interact with the GUI. The following sections explain each element of the UI in more depth, so a detailed description will not be presented in this section.

7.2.1.1.1 Canvas

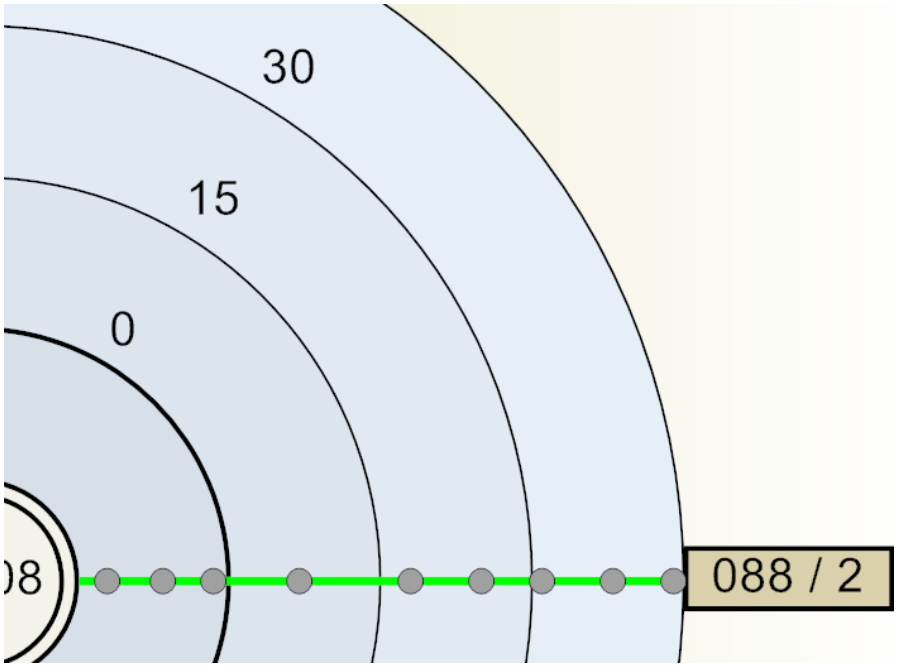

Figure 40: Layover Management Canvas

Layover Management uses a Piccolo canvas object to display a custom UI, per the Headway Monitoring application. Like that application, the layover canvas consists of a PCanvas object, which is a basic primitive in the Piccolo framework required to display objects using Piccolo.

7.2.1.1.2 Graphical Objects

7.2.1.1.3 Interchange

The starting TP – or interchange, for SBS Transit – will be displayed at the center of the Layover Management GUI as a large circular icon with a double outline, and will be labeled using its short timepoint name. All destination routes will be drawn in a circle around this timepoint, and all route lines will intersect it.

The Interchange will provide the following Mouse-Over information:

1. TP Full Name
2. TP Description

7.2.1.1.4 Route

Each user-selected route / direction starting from the interchange will be indicated on the GUI as a line drawn from the interchange to a Destination label. All lines will be spaced an equal number of degrees apart around the interchange circle. At the end of each line a box containing the Route Alpha description and direction text description will be drawn, to clearly label the route / direction. All lines will be drawn of equal length, and all label boxes will be drawn at the outer edge of the same circle. The angle of each route line will be calculated based upon the total number of routes displayed and the total number of degrees in a circle (since routes will be displayed in a circle around the interchange). For example, for eight routes each route will be displayed (360 / 8) = 45 degrees from the last.

Predicted headway status will be calculated for each route as the greatest time distance between vehicles on the route line.

1. System-Configurable, Route-based Headway thresholds will be used to determine if a deviation is out of range (IE, gapping or bunching)

    a. Layover Status from the Service Summary GUI will be used to determine the color of each route line on the Layover GUI. This status will be determined per the Service Summary Design, and will reflect the amount of vehicles that are meeting their scheduled departures, relative to a configurable threshold. The possible colors are Red (Error Threshold Exceeded), Yellow (Warning), and Green (Normal).

2. Lines will be drawn a different color depending on their predicted headway status. Each color and corresponding status is shown below:
    a. Gapping – Uses the Gapping Incident Attribute Color
    b. Bunching – Uses the Bunching Incident Attributed Color
    c. Normal – Green

Each route line will be labeled with its corresponding route alpha and direction code text descriptions. Each label will be drawn inside of a box. Boxes will be placed at the end of each route line, along the outer edge of the Layover Management circle, as follows:

Location	Number of Degrees	Alignment with Circle
Right	0	Center of Left Edge
Upper-Right	0 – 90	Lower-Left Corner

Top	90	Center of Bottom Edge
Upper-Left	90 – 180	Upper-Right Corner
Left	180	Center of Right Edge
Lower-Left	180 – 270	Lower-Right Corner
Bottom	270	Center of Top Edge
Lower-Right	270 – 360	Upper-Left Corner

Figure 41: Orientation of Route Labels

The user may Mouse-Over a Route (line or box) to display a popup with the following information:

1. Route
2. Direction
3. Starting TP (Interchange)
4. Current Status (as text) – Early, Late, etc
5. Minutes Since Last Vehicle Departed
6. Number of Vehicles Displayed

Each route line will display a number next to the interchange indicating the number of minutes since the last vehicle departed on this set of trips.

7.2.1.1.5 Time Intervals

Layover Management will display time intervals on the GUI as a visual aid to indicate the estimate number of minutes until each vehicle's departure. Time Intervals will be drawn on the GUI by a series of concentric circles, each one corresponding to a number of minutes until vehicle departure from the interchange. Vehicles outside a circle are expected to depart in more minutes than the corresponding time interval and likewise, vehicles inside a circle are expected to depart in fewer minutes.

Each circle will be labeled with its corresponding time interval. In addition, the innermost circle will indicate the nearest time interval and will be bolded. Each time interval label will be drawn at the same angle for consistency and as a convenience to the user. To reserve space for these labels, Layover Management will calculate the number of degrees between n route lines (objects) using an input of n + 1 objects. Only n route lines will actually be drawn, but the time interval labels may then be placed at a specific angle without being obstructed by other objects on the GUI.

The same number of time interval circles, 4, will be drawn at any zoom level. However, the number of minutes assigned to each circle will be different depending on the current zoom level.

7.2.1.1.6 Vehicles

Vehicles will be drawn on the GUI at a location corresponding to their estimate departure from the interchange to a given route. Vehicles will be drawn according to the following specifications:

1. The inside of each circle will be blank by default. Vehicle ID could be placed here, but the concern is that the circles would then take up too much space on the GUI.

2. Each vehicle circle will be filled with a color indicating its current RSA status. RSA status will be displayed using the same colors as the CAD Performance Queue.

3. A second circle will be drawn around a vehicle if it is detected that the vehicle may not meet its scheduled departure. The following cases are possible:

Color	Status	Detection Method
Red	The next trip will not depart on time due to minimum required operator layover	Vehicle Departure Time Algorithm
Orange	The vehicle operator's layover can be reduced or cancelled to depart on time	Vehicle Departure Time Algorithm
Black	Vehicle will not depart on time because vehicle will not be available until after scheduled trip departure time	Vehicle Departure Time Algorithm
(No Circle)	Normal	-

4. A letter "L" will be displayed inside a vehicle icon if the vehicle is currently on layover at the interchange.
    a. A vehicle will be determined to be on layover when a timepoint encounter is received for the vehicle at the layover timepoint.

5. Each vehicle will be placed on the route line corresponding to the number of minutes until departure from the interchange on a given route.
    a. Each vehicle will be plotted along the line according to the estimated number of minutes until the vehicle will depart for a trip on the route.
    b. There will be no additional error checking to prevent overlapping of vehicles. It is assumed that two vehicles would not depart simultaneously from the same interchange on a common route. The user may zoom in to see more details if many vehicles are overlapping in close proximity to the interchange.

6. The user may mouse-over a vehicle to display the following information.
    a. Vehicle ID
    b. Logon Route / Block
    c. Estimated Minutes Until Departure (as indicated by position on the GUI)
    d. Layover (If applicable)
        i. Scheduled Length
        ii. Minimum Required Length (only displayed if the feature flag is enabled)
    e. Scheduled / Assigned
        i. Operator Badge
        ii. Route / Direction / Variation
        iii. Run
    f. Current
        i. Operator Badge
        ii. Route / Direction / Variation
        iii. Run
        iv. Deviation
        v. Predicted Arrival Time of Vehicle
        vi. Passenger Count / Capacity

7. The user may click or "rubber band" a vehicle to select it. Multiple vehicles may be selected at any given time. If the user clicks on an empty space on the GUI, all selected vehicles will be deselected.

8. Vehicles scheduled to approach an interchange but that are not active will be displayed using a warning icon similar to the icon for Headway Monitoring. Such vehicles will be displayed at their scheduled location from the interchange, and will be automatically moved forward. These vehicles will display the same mouse-over information as regular vehicles, where applicable.

7.2.1.1.7 Main Menu

Figure 42: Layover Management Main Menu

Layover Management will include a Main Menu at the top of the window. Common functions will be included under each menu item, including opening new tabs, closing tabs, and loading/saving display configurations. A description of each menu is provided below.

7.2.1.1.8 Display Menu

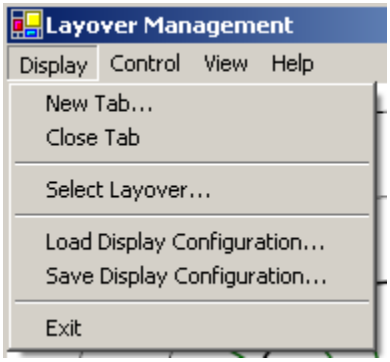

Figure 43: Layover Management Display Menu

The display menu will contain a set of basic operations for the GUI.

New Tab allows the user to open an interchange in a new tab page. When it is selected, the Select Interchange GUI will be opened and the user may select an Interchange and set of Routes. Once an initial configuration has been selected, a new tab will be created and the selected Interchange / Routes will be opened in the new tab page. If the user cancels out of the Select Interchange GUI, a new tab will not be added.

Close Tab will close the current tab page, and unload all data associated with that tab page. If no tab pages are open the menu item will be grayed-out and will not be selectable.

Select Interchange will bring up the Select Interchange GUI and allow the user to adjust the Interchange and/or Routes for the current tab page.

Load Display Configuration will allow the user to open a saved configuration. The configuration will replace all tabs, tab configurations, and display settings for the current instance of the GUI.

Save Display Configuration will allow the user to save the current tab page's configuration. All details for the tab page will be saved, including Interchange, Routes, zoom level, and the automatic refresh setting. Access to this menu item will be restricted via the "Edit Layover Management" classmark. Only users with this classmark will have permission to save configurations. All other users will have the menu item grayed out and non-selectable on their displays.

Exit will terminate the current instance of Layover Management.

7.2.1.1.9 Control Menu

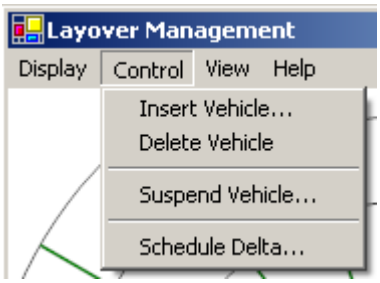

Figure 44: Layover Management Control Menu

The control menu allows the user to perform service adjustments on the GUI. Each of these service adjustment functions is discussed below in the Service Adjustments section of this document.

7.2.1.1.10 View Menu

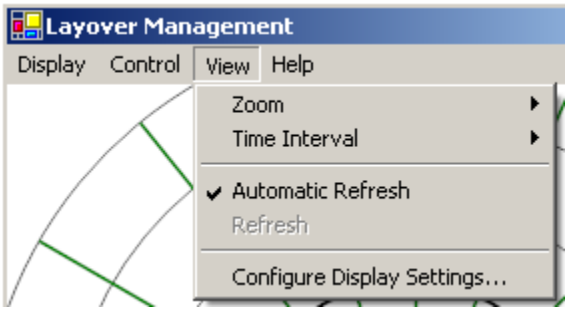

Figure 45: Layover Management View Menu

The view menu allows the user to modify various aspects of the currently displayed Interchange. These menu items are not applicable to a GUI that does not have an interchange displayed – in this case, the View menu itself will be grayed out.

Zoom will allow the user to specify different zoom levels. The following zoom levels will be listed (in order): 400%, 200%, 150%, 100% (default), 75%, 50%. A checkbox will be displayed next to the active zoom level. These values will not be hard coded, but instead will be defined within an XML-based configuration file on the client. If the customer has an urgent need to change these values, the configuration file could be edited as necessary.

Time Interval will allow the user to adjust the maximum time interval displayed on the GUI. The following time intervals will be available for selection, and the active interval will be indicated by a checkbox:

Time Interval	Individual Intervals	Default
90	20, 40, 60, 90	
75	15, 35, 55, 75	
60	15, 30, 45, 60	
45	10, 15, 30, 45	X
30	5, 10, 20, 30	
15	5, 9, 12, 15	

Table 3: Layover Management Time Intervals

The refresh menu items allow the user a degree of control over refreshing of the GUI. Automatic Refresh will toggle automatic refreshing on and off. A checkbox will be displayed next to this menu item to signify the status of automatic refreshing. Only the displayed tab page will be refreshed at regular intervals (although the other tab pages will receive updates in memory). However, the automatic refresh setting will affect all tab pages on the GUI – that is, if the user switches tab pages, the automatic refresh setting will be retained. Finally, access to Automatic Refresh will be controlled to prevent unauthorized access[4].

Clicking the Refresh menu item will trigger a manual refresh of the GUI. This menu item will be grayed out if the automatic refresh menu item is checked.

[4] A customer request, presumably to prevent a lazy dispatcher from "freezing" the display at normal status and ignoring it to ease their workload.

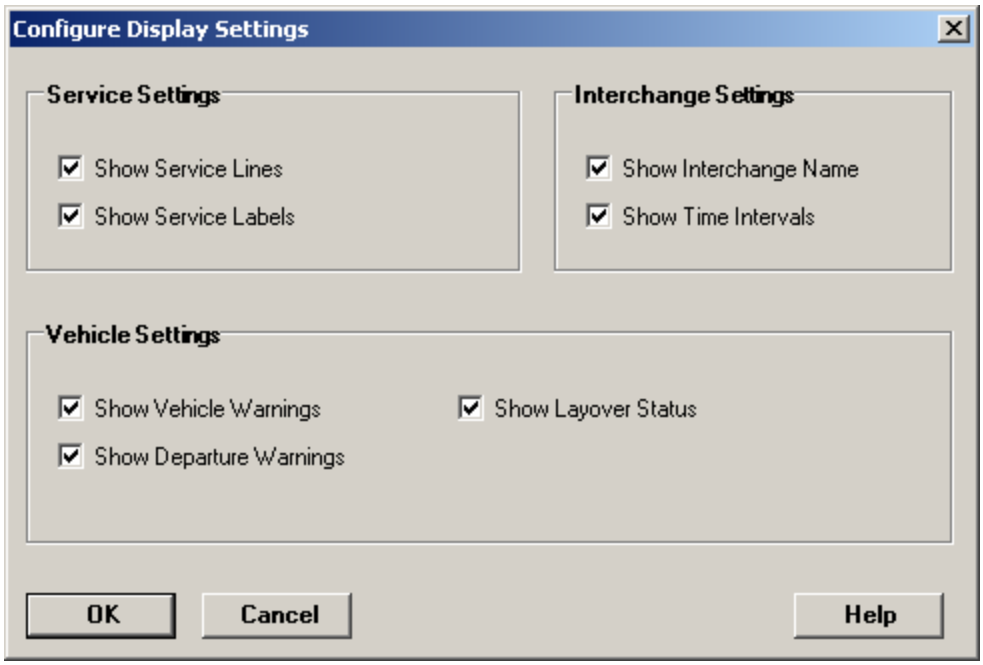

Figure 46: Configure Display Settings GUI

Change Display Settings allows the user to disable various objects on the Layover Display, and will function per the Headway Configure Display Settings GUI. Display options will be available per the picture shown above.

7.2.1.1.11 Help Menu

Figure 47: Layover Management Help Menu

When the user selects the “Layover Management Help” menu item, the Layover Management help file will be displayed.

The “About Layover Management” menu item will open the About dialog box, which will contain copyright and version information, as well as a large application icon.

7.2.1.1.12 Context Menu

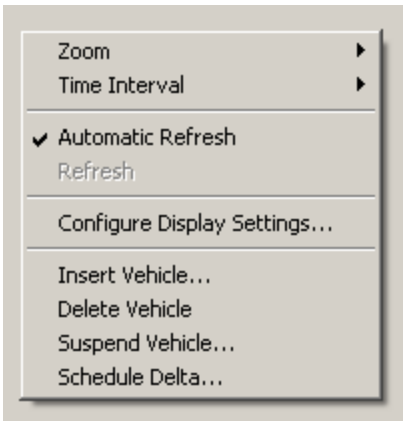

Figure 48: Layover Management Context Menu

Finally, the user may right-click on the GUI to display the context ("popup") menu shown above, which contains a subset of the menu items already discussed.

7.2.2 Business Logic

7.2.2.1 Plotting of Objects Along a Circular Path

The layover display must calculate coordinates along the outer-most circle in order to place destination timepoints across its outer edge and to draw lines between these timepoints and the center. The following trigonometric functions will be used to calculate display coordinates, where r is the radius of the outer circle and θ is the angle at which to draw a line.

$$x = r \cdot \cos(\theta)$$
$$y = r \cdot \sin(\theta)$$

Equation 4: Trigonometric Equations for Coordinates along the Edge of a Circle

These equations are also used when placing vehicles along a path.

7.2.2.2 Vehicle Departure Identification

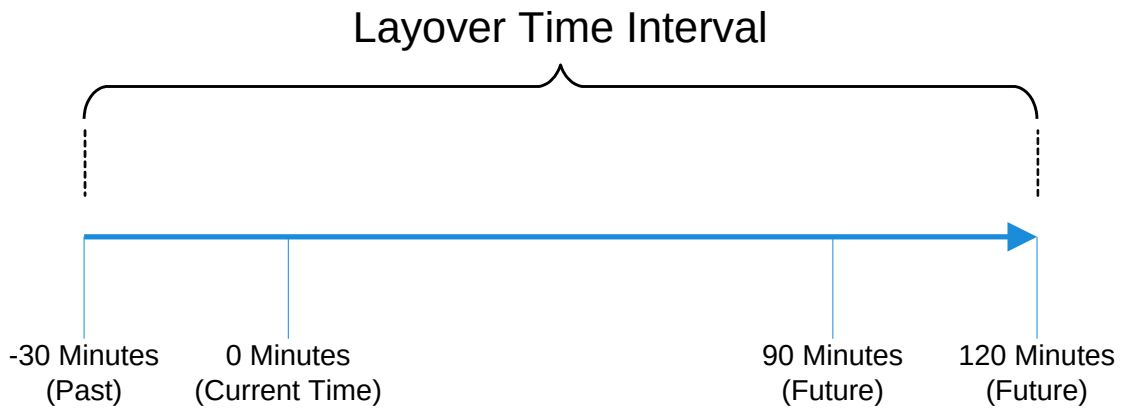

Figure 49: Time Interval used to Identify Vehicle Departures

The layover server calculates vehicle departure times using a sliding time interval. The timeframe used to calculate departures are configurable, although the defaults are listed above. A configurable offset will also be defined to allow a wider time window to account for late/early vehicle departures. The offset is illustrated a both ends of the time interval shown above.

If a departure falls outside this time window, it will no longer be tracked by the Layover server and thus will not show up on the client display. The thinking here is that if a vehicle is too late for its departure (for example, 30 minutes), then that departure will likely be cancelled and trips will simply continue to depart per the schedule. This does not affect early vehicles, as they will have their departures calculated as the times approach the time interval.

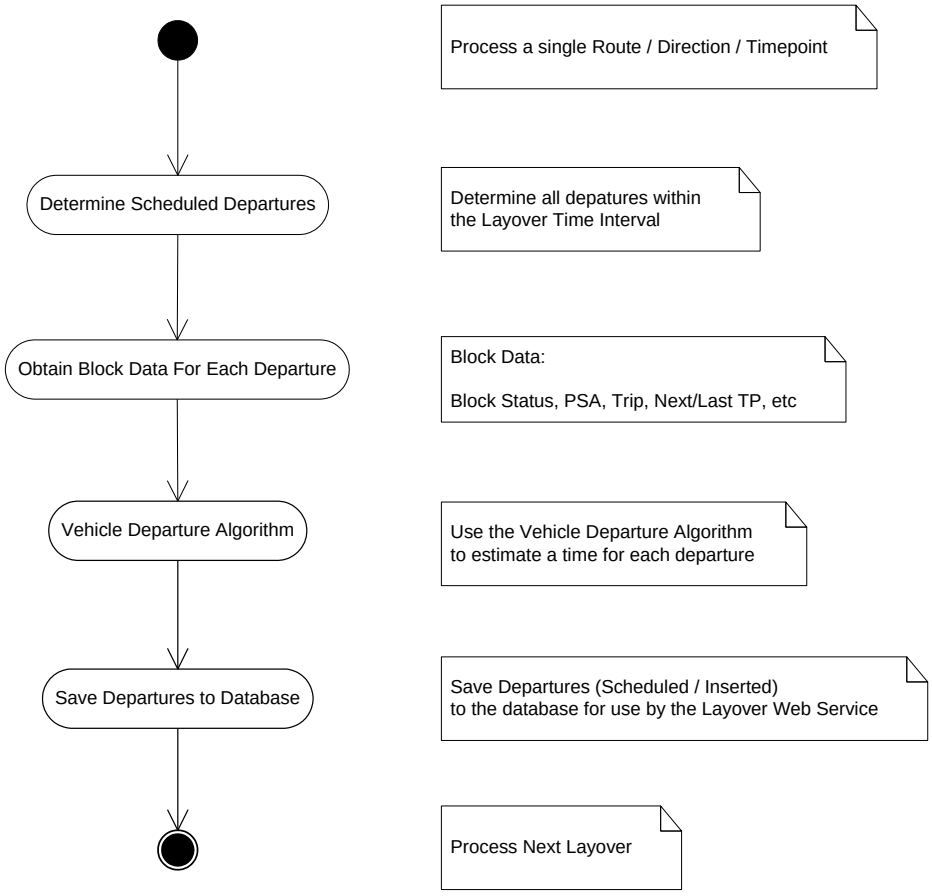

Figure 50: Control Flow of the Layover Management Server

In order to estimate vehicle departure times, the server must obtain a list of scheduled vehicles (blocks) that can be provided as input to the vehicle departure algorithm. Based upon the static data from above, the server will then obtain the current status of each block, using information from shared memory on the server. The database cannot be used because it would be a bottleneck due to the large volume of data required.

Based upon the latest TP encounter and trip information, if a vehicle has already encountered the departure timepoint and has begun work on the departing trip, then a departure time will not be calculated – instead, the data will be used to keep track of the last departure from the layover point. Note that it is important to verify that the vehicle has begun work on the departing trip, instead of an earlier trip that happens to depart from the same layover point.

Manually inserted departures (specials) will also be included, although an estimated departure time will not be calculated for them. Instead, their position will be determined by simply using the departure time entered for them on the Insert Vehicle GUI.

In summary, the vehicle for each departure shall fall into one of the following categories:

- Currently Active
- Currently Active but logged off at Layover Point
- Scheduled to perform a departure but Currently Inactive
- Special Vehicle added via the Insert Vehicle GUI

In addition to vehicle information for the departure algorithm, the departure time of the last vehicle from each interchange on each route/direction must be obtained. This information is displayed next to the interchange for each route displayed, and is required to determine if the next departing bus for each route is gapping or bunching with its leader. Again, information for the last departure will be retrieved from shared memory, based upon the last timepoint encounter from the interchange for each route.

After all calculations have been performed, the departure information will be written to the database as well as shared memory. Data is written to Shared Memory so that it may be modified in near-real time by other server processes, without requiring database polling.

7.2.2.3 Vehicle Departure Time

Layover Management displays each vehicle along a route line based upon that vehicle's estimated departure time from the interchange. To obtain these departure times the vehicle departure algorithm will be executed for each upcoming departure. Since each vehicle's location will change in pseudo-real time, the vehicle departure time must be recalculated before any updates to the vehicle location are made on the Layover Management display.

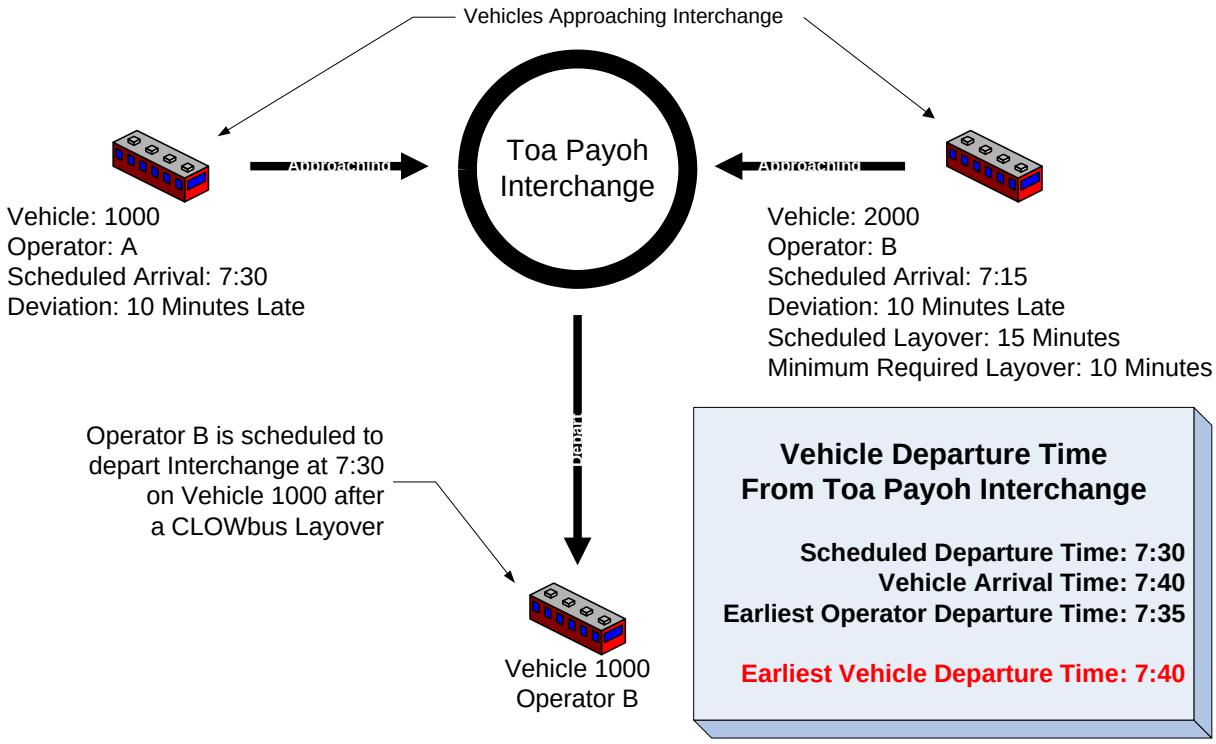

Figure 51: Vehicle Departure Time Considerations

The Scheduled Departure Time, Time of Vehicle Arrival at the Interchange, and Time of Operator Departure from the Interchange must be taken into consideration by the vehicle departure time algorithm – as illustrated by the figure above.

Layover Management will use the following algorithm to calculate departure time from the center TP for a given vehicle.

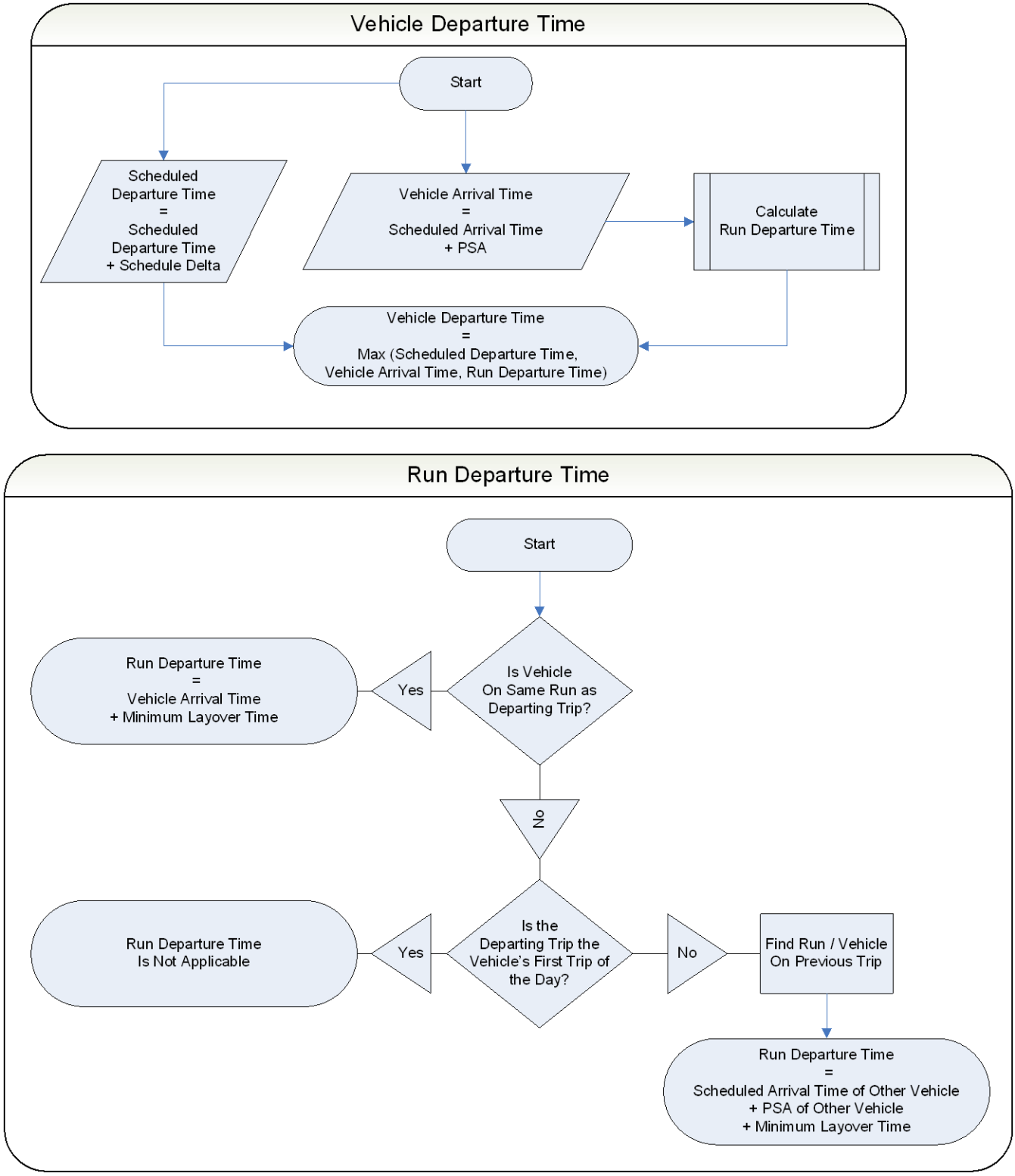

Figure 52: Vehicle Departure Time Algorithm

The algorithm must take three time factors into account before the final vehicle departure time can be determined:

1. Scheduled Departure Time of the displayed trip. If the calculated departure time (with minimum layover) is earlier than the scheduled departure time, the scheduled time should be used because there is no need for the driver to take a shortened break.

2. Vehicle Arrival Time at the end of the previous trip. This time must be taken into account because an operator may switch vehicles between trips, if a CLOWbus layover is performed. If the operator is ready to begin the next trip but the vehicle has not arrived, the operator must wait for that vehicle.

   A PSA value of zero will be used for inactive vehicles that are not currently on a block. It is assumed such a vehicle is parked and awaiting pullout.

3. Run Departure Time: This is the earliest time at which the operator will be ready to begin the displayed trip. To calculate this value, the algorithm must consider when the operator will arrive at the starting timepoint. For SBS, there are two possibilities. If the operator layover previous to the displayed trip is a CLOWbus layover (Crew Layover Without bus) then the operator will use a different vehicle for the next trip – the arrival time of the other vehicle must be considered. Otherwise if the layover is not a CLOWbus the operator will use the same vehicle on the next trip as was used on the previous trip, and that vehicle's arrival time shall be considered.

   Run information for the current trip and previous trip will be obtained from memory (See Data Initialization above).

   Once an arrival time has been determined, the minimum layover time must be added in order to obtain the Run's earliest departure time. For SBS, certain layovers are protected and the vehicle operator is not required to start the next shift until after his/her minimum layover time, even if the operator is running late. Thus, minimum layover time must be considered as opposed to scheduled layover time.

Once each of these components has been found, the algorithm calculates vehicle departure time as the latest (maximum) of the three times. The algorithm must take the maximum time because if the operator or vehicle is late, the next trip cannot start until both arrive. And if both are on time, the operator need not begin the next trip until the scheduled time – unless he is specifically instructed to by dispatch.

Finally, in addition to calculating a departure time, the algorithm returns departure warnings, if any are detected. Only a single warning may be returned for a given vehicle – the algorithm will select the warning with the highest priority. The following warnings are possible:

Warning Description	Priority
None	0
The operator's layover can be reduced or cancelled to depart on time	1
The trip will not depart on time due to the operator's minimum required layover	2
The trip will not depart on time because the vehicle will not be available until after the scheduled departure time	3

Table 4: Vehicle Departure Warnings

During initialization, all required input values for the algorithm are obtained from the Database and cached in the Layover Server's local memory. However, given the large number of inputs required for the algorithm, once initialization is complete the Layover Server obtains all required inputs via IPC Messages or Shared Memory to prevent potentially expensive queries to the Database.

7.2.2.4 Minimum Required Layover

In order to correct potential headway problems at the interchange, dispatchers must be aware of the status of vehicle operators, and whether a particular operator can be requested to start work early. However, SBS has specific union rules guaranteeing operators a minimum break time for certain layovers. Thus it is vital that layover management calculate these minimum layover breaks and display them to dispatchers.

A set of SBS-specific business rules is used to determine minimum required layovers. The rules are as follows (these rules are directly from the customer):

Minimum Layover Considerations

a. A bus driver must be given a minimum layover of 5 minutes (configurable) or 5% of the running time (whichever is higher), on his arrival (even if he arrived after his next scheduled departure time). This applies to the following peak periods:

   i. 0600 - 0900 hrs (Weekday, Saturday)
   ii. 1630 - 1930 hrs (Weekday)
   iii. 1200 – 1400 hrs (Saturday)

   Off peak minimum layover is 5 minutes or 8% of the running time, whichever is the higher.

b. If there is a meal layover allotted at the end of a trip, it has to be at least 20 and 25 minutes for non-Clowbus and Clowbus schedule respectively. Clowbus refers to

Crew layover without bus – in this case, the operator stays at the layover point and another operator drives the bus. In other words, the layover is duty-related not vehicle-related.

c. Similarly, if there is a tea layover allotted at the end of a trip, it has to be at least 10 and 15 minutes for Split Shifts and T-Shift respectively (Note: Tea break is not applicable to A and P Shifts).

d. The above rules are not applicable to some services (trunk or feeder) with 'non-terminating' trips at layover points. There will be a short interval of not more than 4 minutes (i.e. 1, 2, 3, or 4 minutes) before the start of the next trip.

Operation Flexibility

Considerations are to be given to the following variables, wherever applicable, to be set as a parameter entry.

a. Late/early arrival/departure definitions.

b. Minimum layover requirements by break-types (as used in Hastus crew schedule).

c. Peak/non-peak hours definition.

d. 'Non-terminating' service trips exception criteria.

All numerical quantities used in the Business Rules are stored in an XML file to allow them to be user-configurable. In addition, since minimum layover times is are calculated from static schedule data, they are calculated a single time and cached in memory for future accesses.

7.3 Service Restoration

7.3.1 Overview

Service Restoration provides a set of shared functionality enabling dispatcher to perform service adjustments in order to correct headway problems as they are encountered.

Although may service restoration features are already provided in the existing OrbCAD Product, there are specific requirements for both schedule delta's, and in addition a set of other functions are provided to assist dispatchers.

The Service Restoration feature is integrated into the Headway / Layover applications, and these functions may be applied directly from these client applications.

7.3.2 Insert Vehicle

The purpose of Insert Vehicle is to schedule a vehicle for previously unscheduled work on a service. For example, if a service is busy and extra service vehicles must be used to handle the additional demand.

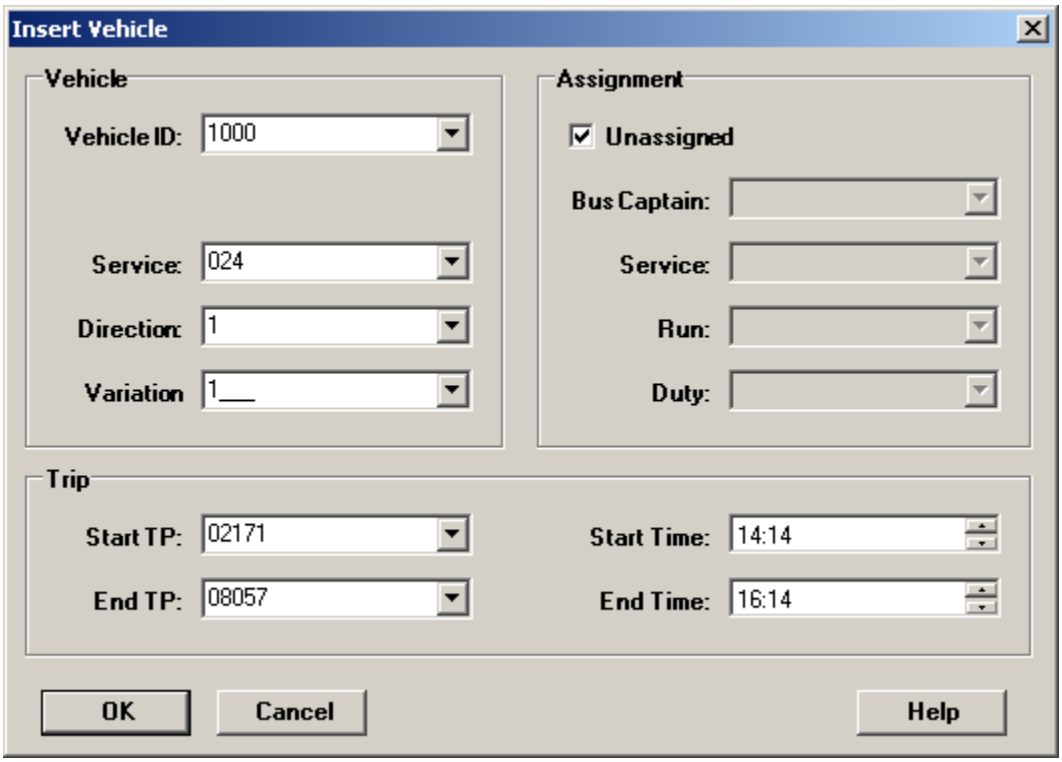

Figure 53: Insert Vehicle GUI

The Insert Vehicle GUI, shown above, allows the user to temporarily display a vehicle on the Headway / Layover GUI's. An inserted vehicle will only be visible for the duration of a single trip – the vehicle would have to be re-inserted to appear for subsequent trips.

The OK button will initially be grayed out. Route and Direction will be auto filled and grayed out if the user has right-clicked on a route immediately before opening the GUI. If the user did not select a route before opening the GUI then the route fields will be editable and will list all routes on the GUI – the user would then have to select a route to continue (direction would be auto filled based upon the route).

The user must select a Route / Direction to place the vehicle upon. These fields will be validated to ensure they are displayed on the calling GUI. If the GUI is opened from Layover Management, the Start TP will be auto-filled to the interchange and grayed-out so the user cannot change it. Each trip field is required for the user to proceed.

The user must then specify the Vehicle ID. Fields from the assignment group box are optional. The fields will only be enabled if the "Unassigned" checkbox is unchecked. Otherwise the fields will be grayed-out and blank. If the user specifies an Operator ID, Route, Block, and Run of the inserted vehicle, then assignment data will be created for the vehicle and sent to SCS. Route will allow selection of all routes in the current booking. Block / Run will only allow selection of Specials, and each field will be auto filled when the other is selected. Validation will be performed for each field to ensure that it is not already displayed.

Once the user clicks OK and closes the GUI, the Insert Vehicle Web Service will be called to apply the change. Headway / Layover displays on other workstations will then be notified of the change upon their next refresh.

The vehicle will initially appear on the Layover Management GUI, until the trip start time is reached, at which point it will be displayed on Headway Monitoring. The vehicle will be moved linearly upon each GUI – however, these calculations will be made by the server and shipped to the client as part of the Headway Update message. Once the entered trip end time has past, the vehicle will disappear from the display. It is expect that work would be completed at this time (Headway Monitoring) or that the operator would log onto the bus at this time and begin work on a special route/block (Layover Management). In either case, the vehicle may then be monitored using the AVL and the CAD Performance Queue.

If edits are required to a vehicle once it has been inserted, the user would have to deleted it (see below) and re-insert the vehicle using the Insert Vehicle GUI.

7.3.3 Delete Vehicle

The user may select an inserted vehicle and click this option to remove the vehicle. The menu item will only be enabled when an inserted vehicle is highlighted – otherwise it will be grayed-out. Once the user selects the menu item, a confirmation message will be displayed. If the user clicks "Yes" to the message, the inserted vehicle will be deleted from all Headway / Layover displays and the database.

7.3.4 Suspend Vehicle

The user may suspend a vehicle's service from the Headway and Layover GUI's. This function may be used to remove a vehicle that has broken down or is known to be unable to perform a departure or remainder of a trip. This function will use existing logic for

Cancel Pullout, and in effect will be an extension of Cancel Pullout for SBS Transit – the "CANCELPO" incident type will be renamed "SUSPEND"

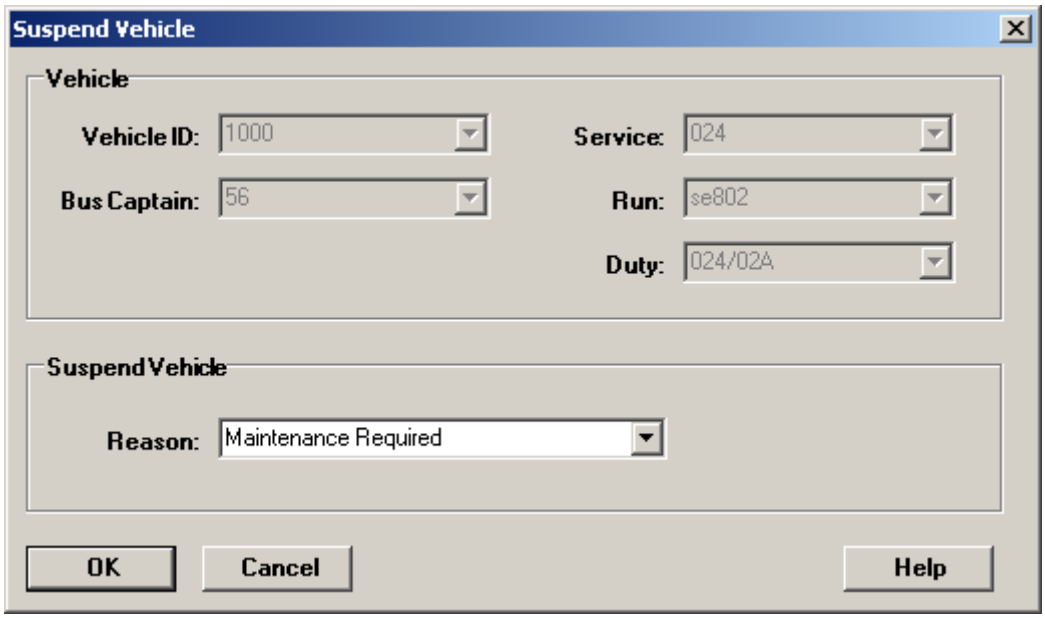

Figure 54: Suspend Vehicle GUI

The user may Suspend a Vehicle by selecting a vehicle on the GUI and clicking the suspend vehicle menu item. The user will be prompted to enter a reason code for suspending the vehicle. The list of reason codes will match the reason codes for Schedule Delta. Once the user supplies a reason and clicks OK to confirm suspension of service, a Cancel Pullout message (see below) will be broadcast to all workstations and the server notifying them of the action. The status will then change to "SUSPEND" on all other GUI's, including the Performance Queue and AVL.

The user may also open the Suspend Vehicle GUI without selecting a vehicle. In this case, all fields will be enabled and the user will have to manually enter vehicle/operator information. Once all information has been entered and the user clicks OK, validation will be performed to ensure that all of the information is correct before the vehicle is suspended.

The cancel pullout message will have to be updated to include the reason code. Once applied, the reason text will replace status in the Current Performance Status table – however, the original status (obtained from the block table) will be restored by the server once the vehicle's status is no longer SUSPEND. In each case, status will be updated on each workstation's Performance Queue upon receipt of the cancel pullout message. Since the text is saved to the Current Performance Status table, it will also be available for any workstations that are logged off and/or restarted.

Finally, a suspended vehicle will remain as SUSPEND in our system until service is resumed for the service – via a RESUME, LOGON, timepoint encounter, etc. No special logic is required for this status change – the existing server RSA logic already handles it.

7.3.5 Schedule Delta

The user may request schedule deltas from the Headway and Layover GUI's. Operationally, the controller would configure the delta from the Headway/Layover GUI and click OK to confirm it. The Schedule Delta GUI will then open to allow the user to review all proposed changes and manually make adjustments as needed. The user may then click OK to confirm and apply the deltas.

7.3.5.1 Automated Adjustment Algorithm

Headway Monitoring and Layover Management will use the following algorithm to automatically calculate Schedule Delta adjustments for a group of vehicles on the GUI. The algorithm will adjust vehicles linearly, giving more weight to those vehicles close to a specific target vehicle, and may be used to correct bunching caused by early or late vehicles, as well as gapping of vehicles. Adjustments will be made using schedule deltas, and each adjustment will affect the arrival time of each vehicle at its next timepoint.

For example, Vehicle 1 is 5 minutes late and gapping with its leader. Those vehicles immediately ahead of Vehicle 1 should be told to slow down, in order to even out the headway between them. A schedule delta is requested for the 3 leaders of vehicle 1. The adjustment algorithm suggests a 4-minute schedule delta for Vehicle 2, a 3-minute delta for Vehicle 3, and a 1-minute delta for Vehicle 4. No adjustment is required for Vehicle 5, so a delta is not applied to this vehicle.

The adjustment algorithm requires as input a target vehicle an action (in this case, adjust late) and the number of vehicles to adjust.

The algorithm then plots all vehicles on the same line, based upon the Scheduled Headway between each vehicle. The steps of the algorithm are as follows:

1. A line slope is calculated, based upon the target vehicle's deviation from its scheduled headway. In this case there is a 5-minute deviation, and a slope of –1/2 is calculated.

2. A single line is plotted using the slope from (1) and an X distance from the origin corresponding to the location of the last vehicle – this line is shown as a dotted line in the diagram. Since the slope and Y-Intercept are known, the equation of the line is also known.

3. An adjustment line (shown in red on the diagram) is plotted from each vehicle's scheduled ETA to the dotted line. The adjustment line is drawn at a configurable

angle from a known point, so the slope and Y-Intercept may be easily calculated, and the equation of the line is known.

4. The X distance between the vehicle's scheduled ETA and the point at which both lines intersect represents the magnitude of the adjustment (IE, the length of the Schedule Delta). The intersection may then be found by setting the Adjustment Line Equation equal to the Dotted Line Equation.

5. Additional adjustment lines are plotted and the Schedule Delta time is computed for each vehicle requiring a schedule adjustment.

The same adjustment algorithm may be used to correct bunching caused by early vehicles. In this cases algorithm components will be *Reflected* – that is, vehicles arriving after the target vehicle will be plotted along the negative x axis, and the dotted slope line will be plotted using a slope of the opposite sign.

In order to correct schedule issues due to insertion (bunching) or deletion (gapping) of vehicles, 2 sets of equivalent adjustments may be performed on either side of the target vehicle. Thus, vehicles ahead of and behind the target vehicle will be evenly spaced with respect to the target vehicle. This technique will be used to create uniform headway for all vehicles close to a target vehicle.

7.3.5.2 Operational Considerations

The user may define schedule deltas from the Layover Management or Headway Monitoring GUI. To define a schedule delta, the user may click on a single target vehicle and select the schedule delta menu item. The Specify Schedule Delta GUI will be displayed to allow the user to select a specific type of Schedule Delta.

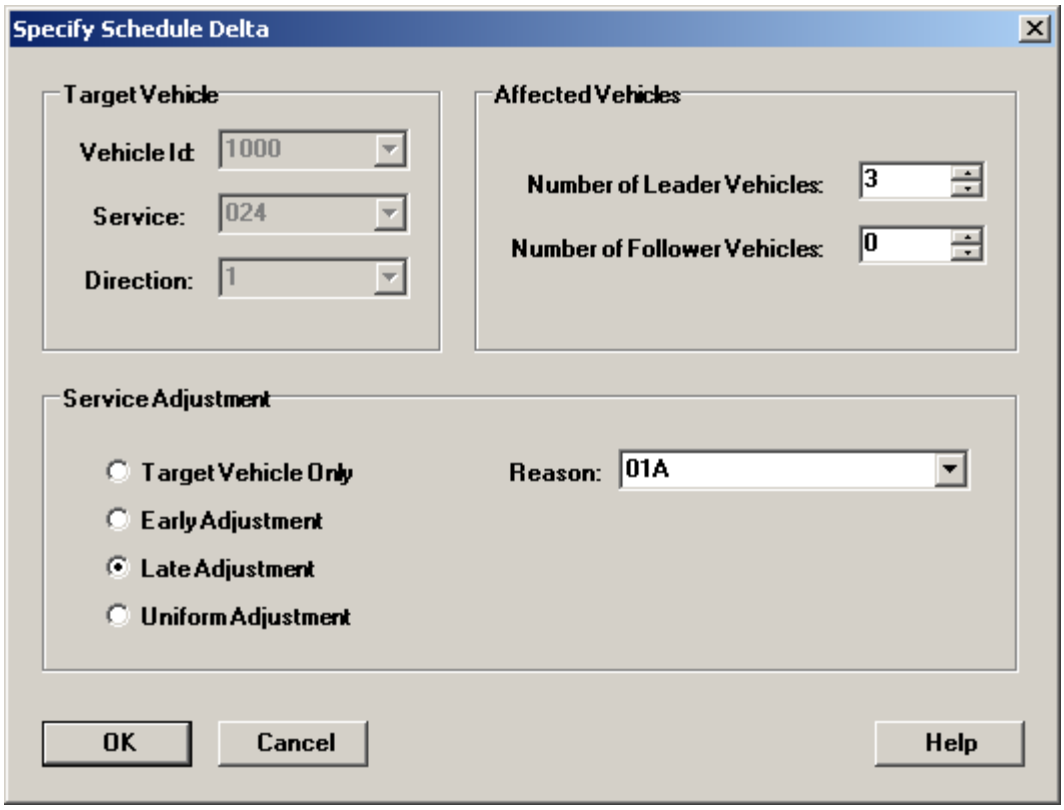

Figure 55: Schedule Delta Selection GUI

The Specify Schedule Delta GUI allows a user to select the type of schedule delta to perform. The Affect Number of Vehicles box refers to the number of vehicles displayed on the same Route / Direction as the target vehicle. Validation will be performed to ensure that this number does not exceed the number of vehicles displayed on the Route / Direction. The options are:

1. Target Vehicle Only
    a. In this case, a schedule delta will only be performed on the single vehicle selected at the top of the GUI. The number of vehicles box will be grayed-out since it does not apply.

2. Early Adjustment
    a. This option allows the user to perform schedule deltas upon the scheduled leaders of a vehicle. Each vehicle will be adjusted early to correct for the target vehicle being early. The user is prompted to select the number of leader vehicles to affect – the minimum is 1. Once a number is specified and the user clicks OK, the automated adjustment algorithm will be called with the specified parameters.

3. Late Adjustment

a. This option allows the user to perform schedule deltas upon the scheduled followers of a vehicle. Each vehicle will be adjusted late to make up for the target vehicle being late. The user is prompted to select the number of leader vehicles to affect – the minimum is 1. Once a number is specified and the user clicks OK, the automated adjustment algorithm will be called with the specified parameters.

4. Uniform Adjustment
   a. This option allows a series of adjustments to be made to leaders and followers of the target Vehicle, in order to space out the headway between all vehicles. For example, if a vehicle is deleted the vehicles following it would need to be sped up and vehicles leading it would need to be delayed, in order to even out the resulting gap. When this radio button is selected the number of vehicles will increment by 2 (an odd number of vehicles does not make sense) and the minimum allowed number would be set to 2.

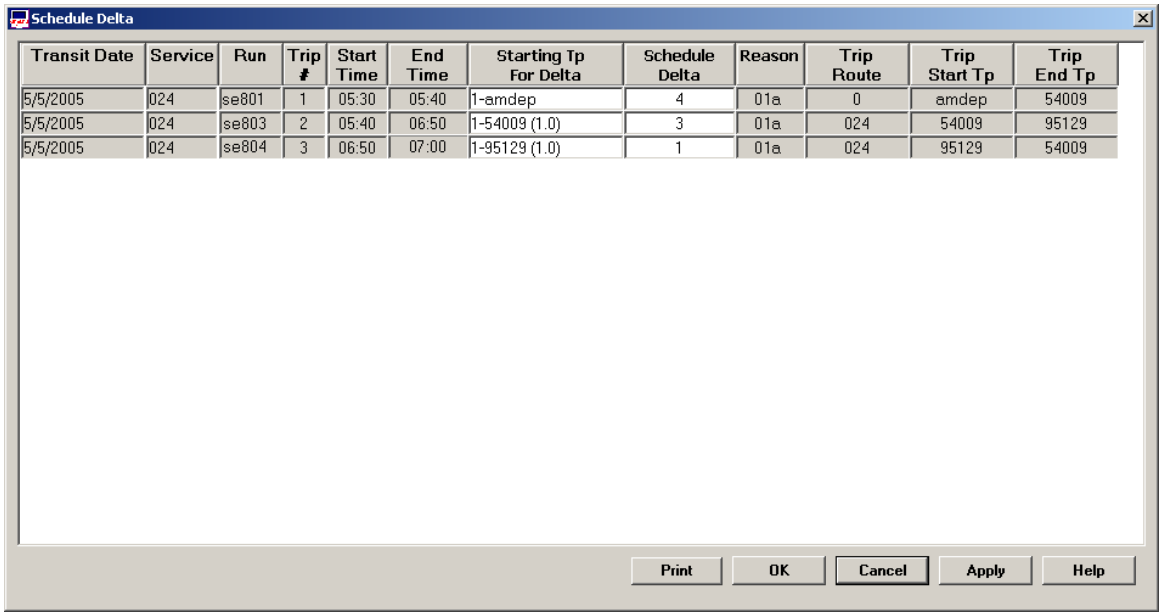

Figure 56: Schedule Delta GUI using the Service Restoration Layout

Once the adjustment has been performed using the automatic adjustment algorithm, the Schedule Delta GUI will be displayed, along with the calculated schedule deltas for each vehicle's next timepoint. When opened in this mode, fields will be rearranged on the GUI in order to allow for processing of multiple vehicles. Route, Block, and Transit Date will be displayed along the row instead of at the top of the GUI. Behind the scenes, this schedule delta GUI will actually be written in .NET instead of PowerBuilder. This decision was made (1) in order to allow for a more robust solution and (2) since a real-

time communication mechanism does not exist between the Headway/Layover processes and the CAD.

The user may then review the proposed adjustments and modify the values accordingly. Finally, the user may select a reason for each schedule adjustment and click OK to apply the adjustments. The Schedule Delta Web Service method will be called to apply schedule delta updates to the system.

Note that both schedule delta GUI's described here will display the Send Message field if it is enabled.

Finally, the user will not have access to the Headway Monitoring / Layover Management Schedule Delta feature unless they have permission to open the GUI from the CAD. This will prevent unauthorized access to the Schedule Delta feature.

8 Modules

This section lists the software modules written for this project. In the actual source code tree, all modules are grouped according to their usage within the solution. For example, all software that was written strictly to run on client workstations is grouped under the Client directory. Modules are presented here within their groupings in the source tree, in order to group common functionality and provide a context for each module.

8.1 Common

Modules under the common directory implement functionality used across all areas of the system. Typically these modules implement generic functionality and thus are candidates for being reused on future projects, or are modules that were originally developed for on projects.

8.1.1 Base Controls

The Base Controls module contains base classes for all major UI classes and components used by the Smart Clients. The intent is to provide wrappers for generic functionality, to extend that provided by standard and third-party controls.

8.1.2 Data Access

This module serves as a Data Access Layer, providing generic database functions abstracting functionality provided by several .NET database classes. The DataAccess class serves as a base class for Microsoft SQL Server, Oracle, ODBC, and OLE. The purpose of this class is to provide a single, unified interface that can be used across the application to provide database access functionality without tying the code to a single database provider.

This class was originally developed by Orbital TMS for the Trimet Annunciator project, and was reworked during this project to be more generic.

8.1.3 Log

The log module provides a suite of classes for logging information. An error logger class logs error messages to a file on disk. An event logger class logs error messages to the windows event log. And a performance log class provides methods to log large amounts of messages to file while minimizing the impact on application performance – this works by leaving the log file open between writes, in order to minimize disk usage. The intent of this class is to provide the ability to instrument code to track the application execution in a production environment.

These modules were also originally developed during the Trimet Annunciator project, and were refactored and extended to include performance logging for this project.

8.1.4 Registry Access

The registry access module provides classes for accessing the windows registry.

8.1.5 Types

The types module provides a set of data types used throughout the OrbCAD-EE code base. For example, data types are provided to represent routes, blocks, vehicles, and other common pieces of data.

8.1.6 Web Service Access

The web service access project provides classes to support web service consumers (clients). Both a base class and an implementation for the utility web service are provided.

Although visual studio automatically generates proxy classes for use on the client side, there are some shortcomings. One potential issue is that custom types are used by the proxy in place of the original types used by the web service provider. A converter base class is provided by this project to allow conversions back to the original base types.

8.2 Client

The client modules contain projects that are used only for client applications. As such, many are GUI-oriented. In addition, many of the modules implement the "controller services" framework shared by both the headway monitoring and layover management Smart Clients.

- Common
  - Controls
  - Data Model
  - Graphics
  - Import
  - Types
  - Utility
- Headway Monitoring
- Layover Management
- Service Restoration

8.3 Interfaces

Projects in the interfaces subdirectory are responsible for proving a means to call existing OrbCAD functions. In general these interfaces consist of function declarations that allow the new .NET code to directly call existing, unmanaged, C/C++ functions in OrbCAD DLL's.

8.4 Server

Projects in this directory implement application code running on the server – which for this project is limited to the Current Status Engine (CSE). An important aspect of the CSE is that it contains pluggable modules (or "engines") that are initialized during startup and run continuously as worker threads of the CSE process.

- Status Engine Base
- Status Engine Headway Monitoring
- Status Engine Layover Management
- Status Engine Service Restoration
- Status Engine Utility

8.5 Web Services

Projects in this directory implement Web Services for consumption by the Smart Clients. These web services abstract the details of the OrbCAD database and provide the means to communicate with the server and other back-end processes. The following web services are implemented:

- Headway
- Layover
- Service Restoration
- Utility – Utility functions used by all requesting applications.

9 Conclusion

Over the course of this project the TMS team and myself successfully designed, implemented, and deployed the Headway and Layover applications described in this paper. Although there were some initial issues during deployment, the application is now stable and a critical part of the customer's day-to-day operations.

This project was quite an undertaking for me, as I had previously never worked on a development task of such a large scale. I have learned a great deal throughout the development process, and will continue to learn more as the OrbCAD-EE framework built here continues to evolve both on Singapore maintenance as well as future development projects.

10 Future Considerations

The OrbCAD-EE code base and frameworks developed for this project were intended to form the basis of new development for TMS. As such, this framework is now being utilized for other projects, for example on a project involving Advanced Traveler

Information System (ATIS). This system contains signs dispersed throughout the city to provide bus arrival time information to patrons waiting at selected bus stops. The work for this task will include the wrapping of DCC components by .NET interfaces, as well as the addition of new, .NET-based processes using the OrbCAD-EE framework.

As for the Singapore project, the headway application continues to play a significant role in their day-to-day operations. It is unclear what role layover is currently serving, although it is expected that this application will play larger role in the future, once SBS personnel become more familiar with the OrbCAD system.

References

1. United States Central Intelligence Agency, The World Factbook – Singapore, https://www.cia.gov/cia/publications/factbook/geos/sn.html, accessed September 23, 2006.
2. SBS Transit, General Information (Operational) , http://www.sbstransit.com.sg/generalinfo/operational.aspx#bus, accessed September 23, 2006.
3. The Mono Project, Main Page, http://www.mono-project.com/Main_Page, accessed September 23, 2006.
4. University of Maryland HCIL, Piccolo Toolkit – A Structured Graphics Framework, http://www.cs.umd.edu/hcil/jazz/, accessed September 23, 2006.
5. University of Maryland HCIL, Piccolo Toolkit – About Piccolo, http://www.cs.umd.edu/hcil/jazz/learn/about.shtml, accessed September 23, 2006.
6. Developer Express, Main Page, http://www.devexpress.com/, accessed September 23, 2006.
7. Damien Foggon, Daniel Maharry, Chris Ullman, and Karli Watson, Programming .NET XML Web Services, Microsoft Press, 2004.
8. Top Downloads from ACM's Digital Library, *Communications of the ACM*, Volume 49 Number 10 (October 2006), Page 19.