TELVENT

Detailed Design Document

TravInfo® 511 System

Transit Agency Application Service

May 5, 2009

Revision History			
Date	**Author**	**QC**	**Notes**
09/05/2007	Karl Barnes		Initial Draft
02/25/2008	Karl Barnes		Adding database changes
04/15/2008	Karl Barnes		Adding additional database changes
08/19/2008	Karl Barnes		Adding additional database changes

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

1.1 Purpose

This document describes the detailed design of the TransitAgencyApplication (TAA) service component of the TravInfo 511 system. The purpose of the TravInfo 511 system is to provide users in the San Francisco, California area with information about transit bus arrivals via telephone and internet web pages displayed on monitors in San Francisco transit centers.

1.2 Objectives

The main objective of this document is to provide software developers with details of the implementation of the software processes involved with the Java Message Service (JMS), file transfer protocol (FTP), and web service design modifications of the TAA system. The deployed version of the TravInfo® 511 system obtains updates from the transit agencies via periodic calls to web services that retrieve agency configuration data including: bus routes, stops along those routes, and predicted arrival times for each bus at each stop. For the next release of the TAA system, bus route and stop information (i.e. agency configuration information), transit vehicle arrival predictions, and vehicle arrival times will be sent from either a central JMS server upon receipt of an extensible markup language (XML) message sent from the TAA, requested from FTP servers, and/or requested from web services provided by transit agencies. Transit vehicle arrival predictions will be sent from the JMS server to systems that subscribe to prediction updates.

1.3 Scope of Design

This design is limited to the 1.0 release.

1.4 Acronyms

The acronyms that appear throughout this document are defined in the table below.

Acronym Definitions

ASCII	American Standard Code for Information Interchange
API	Application Program Interface
CAD	Computer Aided Dispatch

CORBA	Common Object Request Broker Architecture
GUI	Graphical User Interface
ITS	Intelligent Transportation Systems
UML	Unified Modeling Language
SRS	Software Requirements Specification
OI	Operator Interface
IDL	Interface Definition Language
ORB	Object Request Broker
SQL	Structured Query Language
XML	Extensible Markup Language
JMS	Java Message Service
URL	Uniform Resource Locator
FTP	File Transfer Protocol
TAA	Transit Agency Application

1.5 References

- *MTC TravInfo® Contractor, Detailed Design for DYNTRANS, Dynamic Transit Data Interface (NextBus) (Final #3), Deliverable 2-14, Functional Requirement 2.3.2.8, Telvent Farradyne, March 30, 2005*
- *Real-Time Transit Information System Architecture, Regional Real-Time Signs Technical Requirements and Specifications, Version 0.5, Kimley-Horn and Associates Inc., January 23, 2007*
- *Real-Time Transit Information System Architecture, Logical and Physical Architectures DRAFT, Kimley-Horn and Associates Inc., June 19, 2006*
- *Real-Time Transit Information System Architecture, High Level Requirements, FINAL DRAFT, Kimley-Horn and Associates Inc., June 20, 2006*
- *Real-Time Transit Information System Architecture, Concept of Operations, Version 2.1, Kimley-Horn and Associates Inc., October 12, 2006*

1.6 Design Process

Object oriented analysis and design techniques were used in creating this design. The design is documented using diagrams that conform to the Unified Modeling Language (UML) specification. Listed below is the process that was used to create the work products contained in this document:

- Use case diagrams were created to reflect the requirements.
- Class diagrams were created that show possible new classes that will be developed for the system, pre-existing classes that are expected to be used, and the relationships between both new and existing classes.
- Sequence diagrams were created from the use case and class diagrams that show how the classes and their methods will be used to perform each use case.

The design was partitioned into packages that group classes that have a high degree of dependency. This partitioning is documented in a package diagram.

A deployment diagram was created to show which network server(s) would contain each application required by the software module described in this document.

1.7 Design Tools

The work products contained within this document are extracted from the Telelogic Tau UML Suite design tool. This design tool maintains a repository of all project design data stored in a database management system (DBMS). The diagram notation used within this tool is based on the Unified Modeling Language (UML) for Object-Oriented Development, developed by Grady Booch, Ivar Jacobson, and James Rumbaugh. Maintaining the project design data within this tool allows the storage of all documentation within the Models section in a database with the corresponding UML diagram. DocExpress is a tool used to extract the data from the Telelogic Tau system into a Microsoft Word document.

Within the Telelogic Tau tool, the design is contained in the pbfi_repos database in the TransitAgency subsection of the TravInfo®/Data-Collection Basic section.

1.8 Development Tools

The following tools will be used to develop the source code for this module:

- Java Development Environment v1.6.
- Jakarta Ant v1.7.0.
- JacORB v1.4.1.
- OCI Tao ORB v1.3.1.
- Telelogic Synergy/CM Release 6.4 Configuration Management System

- Apache Axis Web Services Framework v1.4
- Apache Tomcat Application Server v6.0 or later
- Apache Web Server v2.2
- SonicMQ JMS Server v7.5 or later
- Microsoft SQL Server Database Management System v2000 or later

1.9 Work Products

This design contains the following work products:

- UML Use Case diagrams that show the operations that are performed by the system.
- UML Class diagrams that show the software objects that will allow the system to be used as described in the Use Case diagrams and implement the system requirements.
- UML Sequence diagrams that show how the classes interact via their methods to accomplish usages of the system.
- A UML Package diagram that shows how the classes are grouped by related functionality into software packages.
- A UML Deployment diagram showing the servers where each software component will be deployed.

2 Software Architecture

This section covers the architecture chosen for the TAA system.

2.1 TAA System

The purpose of the TravInfo® 511 system is to provide users in the San Francisco, California area with information about transit bus arrivals via telephone and internet web pages. The deployed version of the TravInfo® 511 system obtains updates from the transit agencies via periodic calls to web services provided by the NextBus corporation that retrieve agency configuration data including: bus routes, stops along those routes, and predicted arrival times for each bus at each stop. The agency data is accessed via a Java archive (JAR) file provided by NextBus and was designed specifically for use with their services. For the next release of the TAA system, bus route and stop information (i.e. agency configuration information) can be sent from either a central JMS server, FTP server, or web services provided by the transit agencies upon receipt of an extensible markup language (XML) message sent from the TAA. Transit bus arrival predictions can also be sent from the JMS server to systems that subscribe to prediction updates.

Upon startup, the TAA system will read persisted information from a Microsoft SQL Server database. The information read contains the last saved agency configuration information including transit bus routes and stops for each agency. Depending upon the connection information defined in the database for each agency, the system will establish connections to a JMS server, a web server (serving web services), or a NextBus server. Two (2) connection types to the central JMS server may be made. The first connection will handle agency configuration and vehicle arrival time updates. After establishing this connection, the system will issue an XML-formatted configuration update request to the JMS server. The JMS server will send an XML response that contains the requested data for any agencies that provide information to the central server. The second JMS connection will handle transit arrival predictions. In establishing this connection, the TAA system can subscribe to transit arrival prediction updates from the JMS server. The JMS server will send an XML message that contains the current predicted arrival times of all vehicles from agencies that provide information to the central server. The TAA may also request the same information from agency-provided web services. The TAA will examine prediction response data to determine which arrival times have changed. The TAA will examine configuration response data to determine which configurations have changed. For updated predictions, the TAA will publish prediction changes to other subscribing systems via the CORBA Notification service. The TAA will save configuration changes and arrival status information to the system database.

For this release, the capability is added to allow transit hubs to display information on incoming vehicles on video monitors mounted in the transit hubs. These monitors will display a web page that contains vehicle arrival information in a manner similar to that shown in the figure below:

Figure 1. Sample Transit Hub Sign Display

The TAA system will write prediction data to the system database. The transit hub signs will read this data from the database to display on the signs.

2.2 Communications

All communication for this release of the TAA system will either be via XML-formatted message exchange via JMS, files transfered from an FTP server, or marshalling objects via the NextBus-provided classes or the agency web service proxy-stub classes. See the section entitled TAA System Data Specification" for the XML data formats.

2.3 Logging

Error/exception messages, system messages, and debug information will be logged to a file using a previously developed class. The log file is located in the same directory as the properties file.

2.4 Error/Exception Processing

The software handles all expected exceptions. Expected exceptions are problems from which the system can recover from such as problems with input data. The default action when an exception occurs is to either handle the exception where it occurs or to throw a system- specific exception. The system-specific exceptions are thin wrappers for the standard Java Exception, which allow the definition of specific exception messages. Unexpected exceptions or errors will typically cause a stack trace to be written to the log file to aid in analyzing the problem. Errors from which the system cannot recover will cause termination of the system.

2.5 Database Objects

The following sections contain the definitions for the database objects that will be used for this module. Objects that will be new to the system are marked with an asterisk (*).

2.5.1. Database Table: ARRIVALS*

This table contains the actual vehicle arrival times saved for archival purposes.

Table: ARRIVALS*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCY_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for agency
AGENCY_NAME	varchar(255)	NO	YES	(none)	Name of agency
ROUTE_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the route
ROUTE_NAME	varchar(255)	NO	YES	(none)	Name of the route
STOP_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the stop
STOP_NAME	varchar(255)	NO	YES	(none)	Name of the stop
DIRECTION_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the direction
PREDICTION_TIME	Datetime	NO	YES	(none)	Predicted arrival/departure time
PREDICTION_TYPE	Char	NO	YES	(none)	‘A’ for arrival, ‘D’ for departure
VEHICLE_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the vehicle
UPDATETIME	Datetime	NO	YES	(none)	Timestamp for the record
AGENCY_STATUS	varchar(50)	NO	YES	(none)	Status of the agency (e.g. Active, Inactive, Removed, etc.)
PREDICTION_TIME_STATUS	Int	NO	YES	(none)	Prediction status
MESSAGE	varchar(255)	NO	YES	(none)	Optional message

ACTUAL_TIME	Datetime	NO	YES	(none)	Actual arrival/departure time of the vehicle

2.5.2. Database Table: COMPRESSION_TYPES*

The TAA supports data compression when sending/receiving data via JMS. The COMPRESSION_TYPES table lists the supported compression types. Currently, only ZIP ISO-8859-1 is supported.

Table: COMPRESSION_TYPES*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
COMPRESSIONTYPEID	Int	YES	NO	(none)	Unique numeric identifier for each record
COMPRESSIONTYPE	varchar(50)	NO	NO	(none)	Compression Type (e.g. None, ZIP)
ENCODING	varchar(50)	NO	YES	(none)	Encoding used for the compression
SOFTWARETYPEID	Int	NO	NO	(none)	Internal software identifier for the compression type

2.5.3. Database Table: CONNECTION_TYPES*

The COMPRESSION_TYPES table lists the supported agency data connection types. Currently, JMSSubscribe, JMSRequestReply, web service, FTP, and NextBus connection types are supported.

Table: CONNECTION_TYPES*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
CONNECTIONTYPEID	Int	YES	NO	(none)	Unique numeric identifier for each record

CONNECTIONTYPE	varchar(50)	NO	NO	(none)	Connection Type (e.g. JMSRequestReply, JMSSubscribe, web service, FTP, NextBus)
SOFTWARETYPEID	Int	NO	NO	(none)	Internal software identifier for the compression type

2.5.4. Database Table: CONNECTIONS*

This table contains data for the different data connections used by all of the agencies defined in the database.

Table: CONNECTIONS*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
CONNECTIONID	Int	YES	NO	(none)	Unique numeric identifier for each record
CONNECTIONNAME	varchar(255)	NO	YES	(none)	Name of connection
CONNECTIONTYPEID	Int	NO	NO	(none)	Foreign key to ConnectionTypes table denoting type of connection (e.g. JMSRequestReply, web service, etc.)
SERVERSOURCE	varchar(255)	NO	NO	(none)	The URL or connection string for the network location of the server
USERNAME	varchar(255)	NO	YES	(none)	Authenticating username
PASSWORD	varchar(255)	NO	YES	(none)	Authenticating password
RETRIEVALINTERVAL	Int	NO	YES	(none)	Time in seconds between data retrievals
COMPRESSIONTYPEID	Int	NO	NO	(none)	Foreign key to CompressionTypes table denoting what type of data compression is used if any
DATASOURCE	varchar(255)	NO	YES	(none)	Name of the data source if required. For JMS, either topic or queue name; for FTP, the directory containing the file to be downloaded; ignored otherwise

STARTUPCONNECTATTEMPTS	int	NO	YES	1	Number of times to attempt to connect to the data source before reporting an error
RECONNECTATTEMPTS	Int	NO	YES	0	Number of times to attempt to reconnect to the data source after connection is lost before reporting an error.
CONNECTINTERVAL	Int	NO	YES	(none)	Time in seconds to wait before attempting a reconnection to the data source.
CONNECTTIMEOUTSECS	Int	NO	YES	(none)	Time in seconds to wait for a response to a data request
PINGINTERVALSECS	Int	NO	YES	(none)	Time in seconds to periodically check for live connections
DATADESTINATION	varchar(255)	NO	YES	(none)	Name of a data sink/destination – currently unused
RETRIEVALSTART	datetime	NO	YES	(none)	Date/time at which to begin scheduling data retrievals. If date occurs in the past, scheduling will begin at the specified time on the current date.

2.5.5. Database Table: HUB_ROUTE_DISPLAY_MAPPING*

This table maps transit hub routes with sign displays for hubs that contain more data than is desired to be displayed on a single sign display. This table is currently unused.

Table: HUB_ROUTE_DISPLAY_MAPPING*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
HUBROUTEDISPLAYID	Int	YES	NO	(none)	Unique identifier of the record
TRANSITHUBID	Int	NO	NO	(none)	Foreign key to TRANSIT_HUBS table

DISPLAYID	Int	NO	NO	(none)	Determines the display number (in a multi-display transit hub) that will display information about the route specified in this record.

2.5.6. Database Table: NB_AGENCIES

The TAA system persists agency configuration information to a Microsoft SQL Server database. Additional columns will be added to the database table that stores agency configuration information to store connection information. The new structure of the table is shown below.

Table: NB_AGENCIES					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCY_ID	Int	YES	NO	(none)	Unique numeric identifier for the agency
NAME	varchar(64)	NO	NO	(none)	Name of the agency
URL	varchar(255)	NO	NO	(none)	URL of the NextBus server that publishes updates for the agency (currently unused)
ACTIVATIONDATE	Datetime	NO	NO	(none)	Date/time of agency activation. If the activation date is in advance of the current date, the agency will not be processed.
CONFIGURATIONVERSION	varchar(32)	NO	NO	(none)	Version number of the current configuration, used to differentiate from previous versions
NAME511	varchar(64)	NO	YES	(none)	Agency name used by the 511 system
CONFIGURATIONCONNECTIONID*	Int	NO	YES	(none)	Foreign key for establishing a connection to an agency configuration data source
PREDICTIONCONNECTIONID*	Int	NO	YES	(none)	Foreign key for establishing a connection to an agency prediction data source

ARRIVALSTATUSCONNECTIONID*	Int	NO	YES	(none)	Foreign key for establishing a connection to an agency arrival status data source
ICONNAME*	varchar(255)	NO	YES	(none)	Filename of the image file that contains the icon of the agency
GRAMMAR511*	varchar(255)	NO	YES	(none)	VoiceXml grammar used to recognize the agency name spoken by callers to the 511 system.
STOPPREFIX*	varchar(10)	NO	YES	(none)	Text that is added to stop codes to differentiate stops in different agencies

2.5.7. Database Table: NB_ROUTES

This table contains route definitions for an agency.

Table: NB_ROUTES					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
ROUTEID	int	YES	NO	(none)	Primary key
AGENCYID	int	NO	NO	(none)	Foreign key to agency table
ROUTECODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the route
DIRECTIONCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the direction
NAME	varchar(255)	NO	NO	(none)	511 name of the route
DIRECTION	varchar(255)	NO	NO	(none)	Name of the direction
ROUTENAME511	varchar(255)	NO	YES	(none)	Name of the route
GRAMMAR	varchar(255)	NO	YES	(none)	VoiceXML recognition grammar
WAVFILE	varchar(255)	NO	YES	(none)	Sound filename of voice prompt
STATUS	char(1)	NO	NO	(none)	‘A’ for active, ‘R’ for removed
SCHEDULETYPE	char(1)	NO	YES	(none)	
USER	varchar(50)	NO	YES	(none)	‘my511’ for 511 system only or ‘All’

2.5.8. Database Table: NB_STOPS

This table contains stop definitions for an agency.

Table: NB_STOPS					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
STOPID	int	YES	NO	(none)	Primary key
ROUTEID	int	NO	NO	(none)	Foreign key to route table
STOPCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the stop
AGENCYSTOPNAME	varchar(255)	NO	NO	(none)	Name of the stop
NAME	varchar(255)	NO	YES	(none)	511 name of the stop
REGION	varchar(255)	NO	YES	(none)	Name of the region of the stop
REPORTARRIVALTIME	int	NO	YES	(none)	
STATUS	char(1)	NO	NO	(none)	'A' for active, 'R' for removed/inactive
STOPORDER	int	NO	YES	(none)	Order of the stop along the route
ASSOCIATEDCITY	varchar(255)	NO	YES	(none)	City of the stop
ATSTREET	varchar(255)	NO	YES	(none)	Street intersection of the stop
ONSTREET	varchar(255)	NO	YES	(none)	Street along which stop resides
POSITIONONROUTE	varchar(255)	NO	YES	(none)	Order of the stop along the route

2.5.9. Database Table: NB_TRIP_PATTERN_MAPPING*

This table maps NextBus trip mapping IDs to direction codes to allow NextBus data to be processed identically to data from other sources.

Table: NB_TRIP_PATTERN_MAPPING*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
ROUTECODE	varchar(255)	NO	NO	(none)	Alpha-numeric identifier of the route
TRIPPATTERNID	Int	YES	NO	(none)	Unique identifier of the record
DIRECTIONCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the direction
DIRECTION	varchar(255)	NO	YES	(none)	Direction text – unused
TITLE_TEXT	varchar(255)	NO	YES	(none)	Direction title – unused
MUNI_DESCRIPTION	varchar(255)	NO	YES	(none)	Description of trip – unused
AGENCYNAME	varchar(255)	NO	NO	(none)	Name of the agency

2.5.10. Database Table: PREDICTIONS*

This table contains the predicted vehicle arrival times saved for archival purposes.

Table: PREDICTIONS*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCY_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for agency
AGENCY_NAME	varchar(255)	NO	YES	(none)	Name of agency
ROUTE_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the route
ROUTE_NAME	varchar(255)	NO	YES	(none)	Name of the route
STOP_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the stop
STOP_NAME	varchar(255)	NO	YES	(none)	Name of the stop
DIRECTION	varchar(50)	NO	YES	(none)	Direction name
PREDICTION_TIME	Datetime	NO	YES	(none)	Arrival/departure time of the vehicle
PREDICTION_TYPE	Char	NO	YES	(none)	‘A’ for arrival, ‘D’ for departure
VEHICLE_ID	varchar(50)	NO	YES	(none)	Alpha-numeric identifier for the vehicle

UPDATETIME	Datetime	NO	YES	(none)	Timestamp for the record
AGENCY_STATUS	varchar(50)	NO	YES	(none)	Status of the agency (e.g. Active, Inactive, Removed, etc.)
PREDICTION_TIME_STATUS	Int	NO	YES	(none)	Prediction status
MESSAGE	varchar(255)	NO	YES	(none)	Optional message
ACTUAL_TIME	Datetime	NO	YES	(none)	Actual arrival/departure time of the vehicle

2.5.11. Database Table: TRANSIT_HUB_ROUTES*

This table contains the list of routes defined for each transit hub.

Table: TRANSIT_HUB_ROUTES*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
TRANSITHUBROUTEID	Int	YES	NO	(none)	Primary key
AGENCYID	Int	NO	NO	(none)	Foreign key to NB_AGENCIES table
ROUTE_ID	Int	NO	NO	(none)	Foreign key to NB_ROUTES table
STOP_ID	Int	NO	NO	(none)	Foreign key to NB_STOPS table
MESSAGE	varchar(255)	NO	YES	(none)	Optional message to be displayed on the sign instead of arrival time(s)
DISPLAYID	Int	NO	NO	(none)	Display that will display data for the route specified in this record
TRANSITHUBID	Int	NO	NO	(none)	Foreign key to TRANSIT_HUBS table

2.5.12. Database Table: TRANSIT_HUBS*

This table contains the list of routes defined for each transit hub.

Table: TRANSIT_HUBS*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
TRANSITHUBID	Int	YES	NO	(none)	Primary key
TRANSITHUBNAME	varchar(255)	NO	NO	(none)	Name of the transit hub

2.5.13. Database Table: ROUTE_MESSAGES*

This table contains optional messages that are displayed on transit hub signs instead of arrival times.

Table: ROUTE_MESSAGES*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
ROUTEMESSAGEID	Int	YES	NO	(none)	Primary key
ROUTE_ID	Int	NO	NO	(none)	Foreign key to NB_ROUTES table
MESSAGE	varchar(255)	NO	YES	(none)	Optional message to be displayed on the sign instead of arrival time(s)
TRANSITHUBID	Int	NO	NO	(none)	Foreign key to TRANSIT_HUBS table

2.5.14. Database Table: NB_TRIP_PATTERN_MAPPING_STAGING*

This staging table maps NextBus trip mapping IDs to direction codes to allow NextBus data to be processed identically to data from other sources.

Table: NB_TRIP_PATTERN_MAPPING_STAGING*

Column Name	Data Type	Primary Key	Nullable	Default	Description
ROUTECODE	varchar(255)	NO	NO	(none)	Alpha-numeric identifier of the route
TRIPPATTERNID	Int	YES	NO	(none)	Unique identifier of the record
DIRECTIONCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the direction
DIRECTION	varchar(255)	NO	YES	(none)	Direction text – unused
TITLE_TEXT	varchar(255)	NO	YES	(none)	Direction title – unused
MUNI_DESCRIPTION	varchar(255)	NO	YES	(none)	Description of trip – unused
AGENCYNAME	varchar(255)	NO	NO	(none)	Name of the agency

2.5.15. Database Table: NB_ROUTES_STAGING*

This staging table contains route definitions for an agency.

Table: NB_ROUTES_STAGING*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
ROUTEID	int	YES	NO	(none)	Primary key
AGENCYID	int	NO	NO	(none)	Foreign key to agency table
ROUTECODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the route
DIRECTIONCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the direction
NAME	varchar(255)	NO	NO	(none)	Name of the route
DIRECTION	varchar(255)	NO	NO	(none)	Name of the direction
ROUTENAME511	varchar(255)	NO	YES	(none)	VoiceXML recognition grammar
WAVFILE	varchar(255)	NO	YES	(none)	Sound filename of voice prompt
STATUS	char(1)	NO	NO	(none)	‘A’ for active, ‘R’ for removed
SCHEDULETYPE	char(1)	NO	YES	(none)	
USER	varchar(50)	NO	YES	(none)	‘my511’ for 511 system only or ‘All’

2.5.16. Database Table: NB_STOPS_STAGING*

This staging table contains stop definitions for an agency.

Table: NB_STOPS_STAGING*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
STOPID	int	YES	NO	(none)	Primary key
ROUTEID	int	NO	NO	(none)	Foreign key to route table
STOPCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the stop
AGENCYSTOPNAME	varchar(255)	NO	NO	(none)	Name of the stop
NAME	varchar(255)	NO	NO	(none)	511 name of the stop
REGION	varchar(255)	NO	YES	(none)	Name of the region of the stop
REPORTARRIVALTIME	int	NO	YES	(none)	
STATUS	char(1)	NO	NO	(none)	'A' for active, 'R' for removed
STOPORDER	int	NO	YES	(none)	Order of the stop along the route
ASSOCIATEDCITY	varchar(255)	NO	YES	(none)	City of the stop
ATSTREET	varchar(255)	NO	YES	(none)	Street intersection of the stop
ONSTREET	varchar(255)	NO	YES	(none)	Street along which stop resides
POSITIONONROUTE	varchar(255)	NO	YES	(none)	Order of the stop along the route

2.5.17. Database Table: ARRIVAL_STATUS_STOPS*

This table contains a list of stops for an agency for which arrival status data is to be retrieved.

Table: ARRIVAL_STATUS_STOPS*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**

AGENCYID	Int	YES	NO	(none)	Foreign key to NB_AGENCIES table
STOPID	Int	YES	NO	(none)	Foreign key to NB_STOPS table

2.5.18. Database Table: RTT_ARRIVALS_LOG*

This table contains arrival status data imported from the TransitAgencyApplication log files.

Table: RTT_ARRIVALS_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCYNAME	nvarchar(40)	NO	YES	(none)	Agency name
ROUTECODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the route
DIRECTIONCODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the direction
STOPCODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the stop
PREDICTIONTYPE	nvarchar(40)	NO	YES	(none)	'A' for arrival, 'D' for departure
VEHICLEID	nvarchar(40)	NO	YES	(none)	Identifier of the vehicle en-route
TIMESTAMP	nvarchar(40)	NO	YES	(none)	Date/time of the receipt of the information
VEHICLETIME	nvarchar(40)	NO	YES	(none)	Arrival/departure time of the vehicle

2.5.19. Database Table: RTT_CONFIGURATIONS_LOG*

This table contains configuration data imported from the TransitAgencyApplication log files.

Table: RTT_CONFIGURATIONS_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCYNAME	nvarchar(40)	NO	YES	(none)	Agency name

ACTIVATIONDATE	nvarchar(40)	NO	YES	(none)	Date/time of agency activation. If the activation date is in advance of the current date, the agency will not be processed.
CONFIGURATIONVERSION	nvarchar(40)	NO	YES	(none)	Version number of the current configuration, used to differentiate from previous versions
MESSAGE	nvarchar(40)	NO	YES	(none)	Optional message

2.5.20. Database Table: RTT_DIRECTIONS_LOG*

This table contains direction data imported from the TransitAgencyApplication log files.

Table: RTT_DIRECTIONS_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCYNAME	nvarchar(40)	NO	YES	(none)	Agency name
ROUTECODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the route
DIRECTIONCODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the direction
DIRECTIONNAME	nvarchar(40)	NO	YES	(none)	Direction of the route
STATUS	nvarchar(40)	NO	YES	(none)	‘A’ for active, ‘R’ for removed

2.5.21. Database Table: RTT_PREDICTIONS_LOG*

This table contains prediction data imported from the TransitAgencyApplication log files.

Table: RTT_PREDICTIONS_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**

AGENCYNAME	nvarchar(40)	NO	YES	(none)	Agency name
ROUTECODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the route
DIRECTIONCODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the direction
STOPCODE	nvarchar(40)	NO	YES	(none)	Alpha-numeric identifier of the stop
TIMESTAMP	nvarchar(40)	NO	YES	(none)	Date/time of the receipt of the information
VEHICLETIME	nvarchar(40)	NO	YES	(none)	Arrival/departure time of the vehicle

2.5.22. Database Table: RTT_ROUTES_LOG*

This table contains route data imported from the TransitAgencyApplication log files.

Table: RTT_ROUTES_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCYNAME	Nvarchar(40)	NO	YES	(none)	Agency name
ROUTECODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the route
NAME	varchar(255)	NO	NO	(none)	511 name of the route
STATUS	char(1)	NO	NO	(none)	‘A’ for active, ‘R’ for removed

2.5.23. Database Table: RTT_STOPS_LOG*

This table contains stop data imported from the TransitAgencyApplication log files.

Table: RTT_STOPS_LOG*					
Column Name	**Data Type**	**Primary Key**	**Nullable**	**Default**	**Description**
AGENCYNAME	Nvarchar(40)	NO	YES	(none)	Agency name
STOPCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the stop

STOPNAME	varchar(255)	NO	NO	(none)	Name of the stop
DIRECTIONCODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the direction
ROUTECODE	varchar(50)	NO	NO	(none)	Alpha-numeric identifier of the route
STATUS	char(1)	NO	NO	(none)	‘A’ for active, ‘R’ for removed/inactive

2.6 Interface Definition Language (IDL)

The system uses classes defined in the main system IDL file. The version of the IDL file used for this design is shown in Appendix D.

3 Models

The following sections provide models and diagrams that show the detailed design of the TAA system software. These sections contains one or more use case diagrams, class diagrams, and sequence diagrams. The use case diagrams serve to show the types of ways the users and external systems may interact with the TravInfo 511 system. The class diagrams show software objects that comprise the system and their relationships to each other. The sequence diagrams show how the objects defined in the class diagram(s) interact to accomplish specific uses of the system.

3.1 Use Cases

This use case diagram describes the ways in which the TransitAgency subsystem may interact with the operating system, Travinfo administrator, transit agency Java Message Server system, NextBus system, and agency web services.

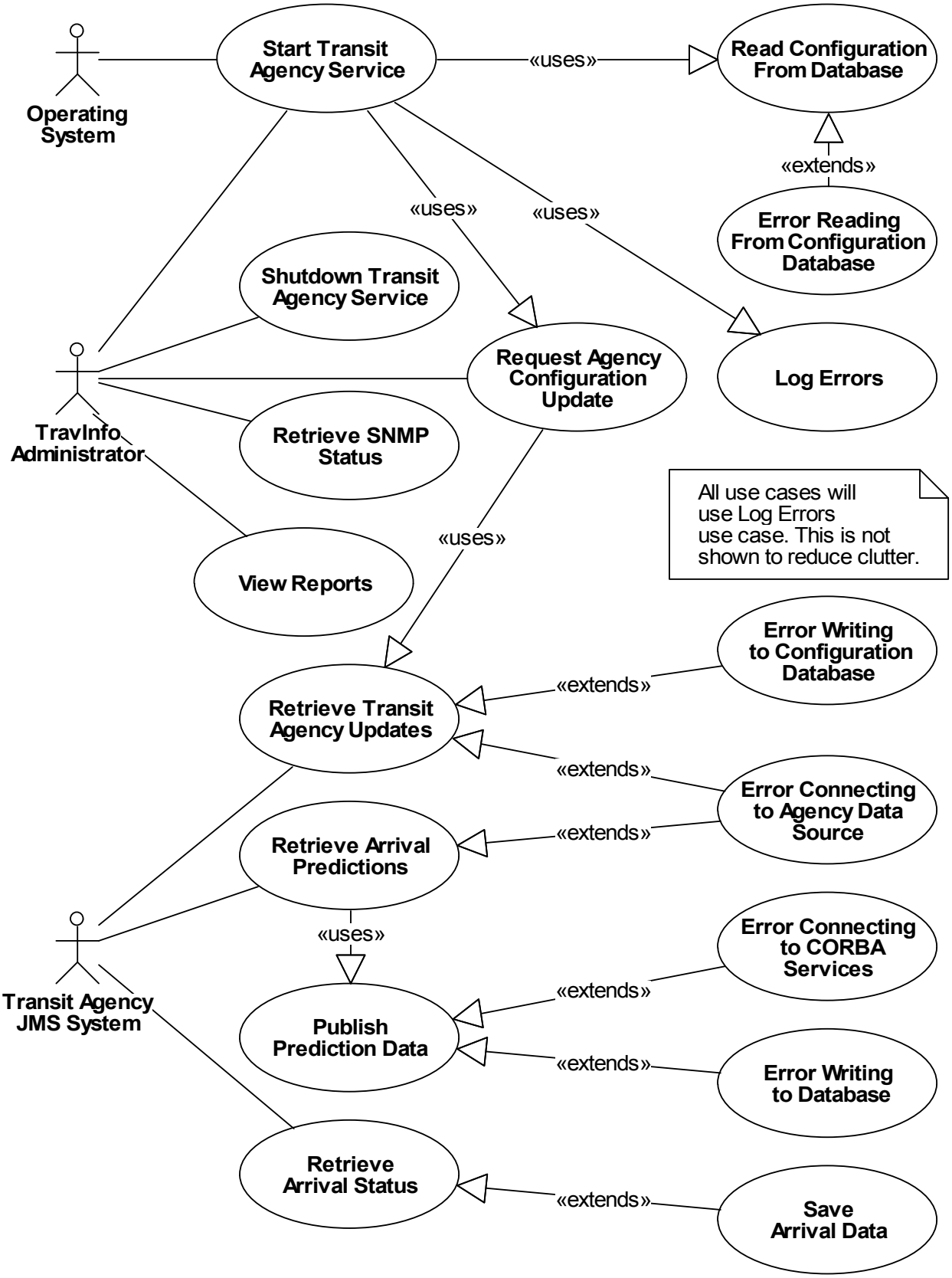

Figure 2. TransitAgencyUseCases (Use Case Diagram)

3.1.1. Error Connecting to Agency Data Source (Use Case)

Connectivity must be established and maintained with the JMS service. If not, the error is logged.

3.1.2. Error Connecting to CORBA Services (Use Case)

Connectivity must be established with the CORBA Naming and Notification services.

3.1.3. Error Reading From Configuration Database (Use Case)

If the system is repeatedly unable to read agency configuration information from the database, the error is logged and the system is halted.

3.1.4. Error Writing to Configuration Database (Use Case)

If the system is repeatedly unable to write agency configuration information to the database, the error is logged.

3.1.5. Log Errors (Use Case)

Writes status and error information to a log file.

3.1.6. Error Writing to Database (Use Case)

If the system is repeatedly unable to write prediction information to the database, the error is logged.

3.1.7. Operating System (Actor)

The operating system starts the service at system startup.

3.1.8. Publish Prediction Data (Use Case)

Publishes prediction data to the CORBA notification service in order to be used by external systems. Also writes prediction data to the system database.

3.1.9. Request Agency Configuration Update (Use Case)

Initiates transit agency service to begin receiving and processing transit agency updates either from the Java Message Server, NextBus, or agency web services.

3.1.10. Read Configuration From Database (Use Case)

The transit agency service will read persisted agency configuration information from the database.

3.1.11. Retrieve Arrival Predictions (Use Case)

Receives and processes transit agency arrival predictions from either the Java Message Server, NextBus, or agency web services.

3.1.12. Transit Agency JMS System (Actor)

The transit agency will maintain a Java Message Server (JMS) that will contain updates to the transit agency configuration (e.g. routes), and arrival prediction updates. The transit agency service will subscribe to updates to the JMS.

3.1.13. TravInfo Administrator (Actor)

The TravInfo administrator is responsible for the operation of the service. The administrator may re-start the service after a shutdown, perform a forced shutdown on the system, manually request an agency configuration update, view reports prepared by the system, or view the status of the service via SNMP.

3.1.14. Retrieve Arrival Status (Use Case)

Receives and processes transit agency arrival status data from either the Java Message Server, NextBus, or agency web services.

3.1.15. Save Arrival Data (Use Case)

Saves arrival status data to the system database.

3.1.16. Retrieve SNMP Status (Use Case)

All changes in the status of the application including errors are sent as alerts to an SNMP manager application or the network to respond to queries for program status.

3.1.17. Shutdown Transit Agency Service (Use Case)

When the transit agency service is shutdown, the service will un-subscribe to transit agency JMS updates and shutdown all internal processes.

3.1.18. Start Transit Agency Service (Use Case)

When the transit agency service is started, it will read persisted agency configuration data from the database and subscribe to receive transit agency updates from their Java Message Server.

3.1.19. Retrieve Transit Agency Updates (Use Case)

Receives and processes transit agency updates from either the Java Message Server, NextBus, or agency web services.

3.1.20. View Reports (Use Case)

Reports, including a transit agency configuration report, will be generated.

3.2 Activity Diagram

The activity diagram shows threads are used to process agency configuration changes, arrival status, and prediction updates. Each of the swimlanes represents a thread of execution.

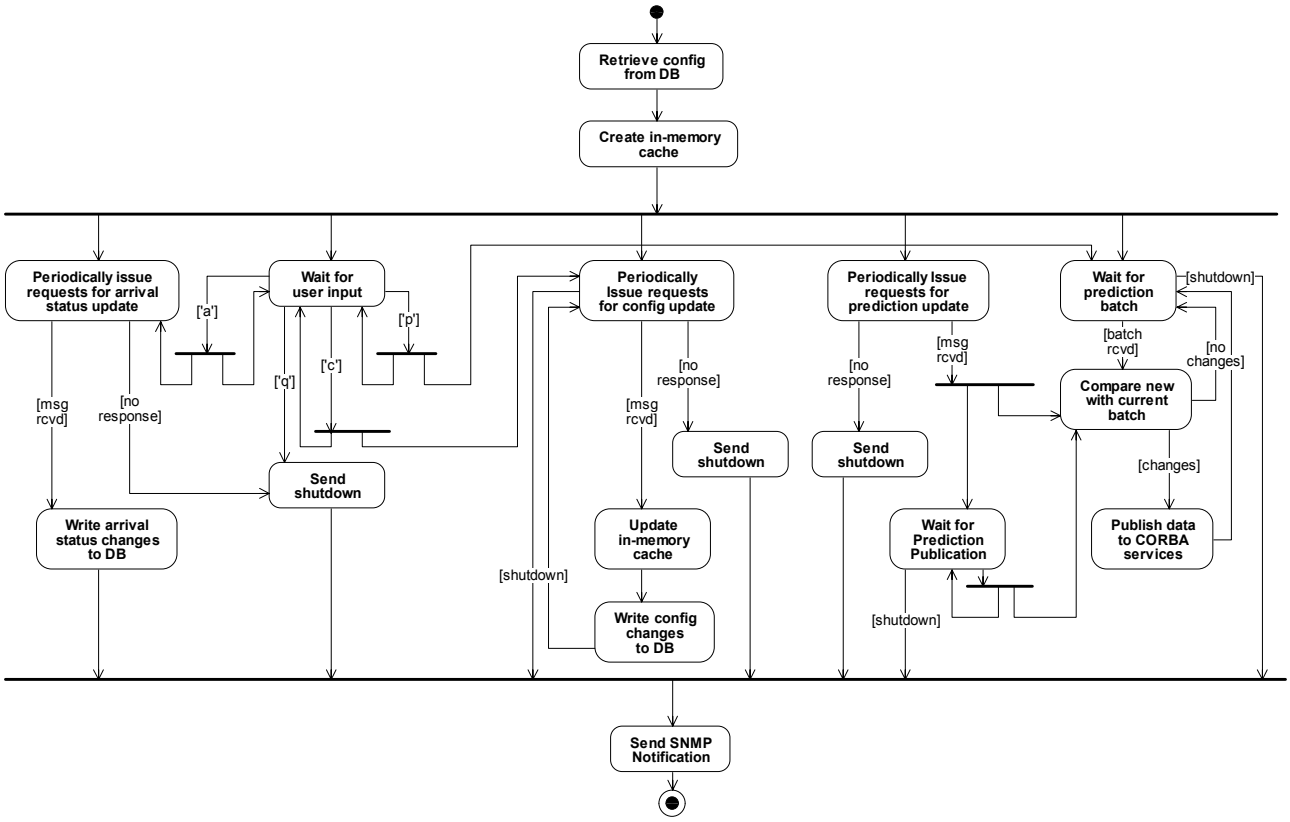

Figure 3. TransitAgencyApplication:Activity (Activity Diagram)

3.2.1. Compare new with current batch (Action State)

This action compares newly-received arrival time prediction updates with the current predictions to determine if there are any changes that require publication to the CORBA Notification service.

3.2.2. Create in-memory cache (Action State)

Upon system startup, this action uses the agency configuration data read from the database to create an in-memory cache containing the configuration information. The cache contains the configuration information used to retrieve predictions for routes/stops.

3.2.3. Periodically Issue requests for prediction update (Action State)

During this state, the system periodically issues a request for prediction data. If data is retrieved, it is published to the CORBA notification server and optionally saved to the database.

3.2.4. Retrieve config from DB (Action State)

Upon system startup, this action retrieves the previously saved agency configurations from the system database.

3.2.5. Wait for user input (Action State)

This action waits for and processes user commands entered via the console (i.e. keyboard). If the user types 'c', the agency configurations are updated for all request/reply data sources. If the user types 'a', the arrival status is retrieved for all request/reply data sources. If the user types 'p', vehicle predictions are retrieved for all request/reply data sources. If the user types 'q', the system shuts down.

3.2.6. Periodically issue requests for arrival status update (Action State)

During this state, the system periodically issues a request for arrival status data. If data is retrieved, it is written to the database.

3.2.7. Periodically Issue requests for config update (Action State)

During this state, the system periodically issues a request for configuration data. If data is retrieved, it is optionally used to update the in-memory configuration cache and changes are written to the database.

3.2.8. Publish data to CORBA services (Action State)

This action sends newly-received arrival time prediction updates for publication to the CORBA Notification service.

3.2.9. Send shutdown (Action State)

This action sends shutdown signals to all running threads.

3.2.10. Send SNMP Notification (Action State)

This action handles notifying the operator of an event causing a system shutdown.

3.2.11. Update in-memory cache (Action State)

This action uses the agency configuration data sent from the Java Message Service to update the in-memory cache containing the configuration information. The cache contains the configuration information used to retrieve predictions for routes/stops.

3.2.12. Write arrival status changes to DB (Action State)

This action writes arrival status data to the database.

3.2.13. Wait for prediction batch (Action State)

This action waits for arrival time prediction updates the require publication to the CORBA Notification service.

3.2.14. Wait for Prediction Publication (Action State)

In this state, the system waits for predictions to be received from any publish-subscribe-based data sources.

3.2.15. Write config changes to DB (Action State)

This action writes configuration data to the database.

3.3 Class Diagrams

3.3.1. ArrivalStatusCD (Class Diagram)

This class diagram shows the relationship between the classes comprising agency vehicle arrival status data. This diagram shows the hierarchical and containment/composition relationships between the classes.

ArrivalStatus

m_routeID: int
m_agencyID: int
m_agencyName: String
m_routeCode: String
m_directionCode: String
m_routeName: String
m_direction: String
m_agencyStatus: String
m_stopID: int
m_stopCode: String
m_stopName: String
m_predictionTimeStatus: String
m_message: String
m_timeStamp: Date
m_actualTime: Date
m_predictionType: String
m_vehicleId: String

ArrivalStatus(int agencyId, String routeCode, String directionCode, int stopId, Date actualTime, String vehicleId)
ArrivalStatus(String agency, String routeCode, String directionCode, String stop, Date actualTime, String vehicleId)
getRouteID(): int
getAgencyID(): int
getTime(): Date
getRouteKey(): String
getStopKey(): String
getDirectionKey(): String
getAgency(): String
getRoute(): String
getDirection(): String
getMessage(): String
getAgencyStatus(): String
getPredictionTimeStatus(): String
getStop(): String
getVehicleID(): String
getPredictionType(): String
getStopID(): int
setRouteID(int routeID): void
setAgencyID(int agencyID): void
setRouteCode(String routeCode): void
setDirectionCode(String directionCode): void
setDirection(String direction): void
setAgencyStatus(String status): void
setActualTime(Date actualTime): void
setVehicleID(String vehicleId): void
setStopID(int stopId): void
setPredictionType(String predictionType): void
setAgencyName(String name): void
setRouteName(String name): void
setStopName(String name): void
setStopCode(String code): void
toString(): String
toXML(): String

Figure 4. ArrivalStatusDataCD (Class Diagram)

3.3.2. ArrivalStatus (Class)

This class represents a ArrivalStatus data structure and contains transit arrival/departure time information.

3.3.3. ConfigurationDataCD (Class Diagram)

This class diagram shows the relationship between the classes comprising agency configuration data. This diagram shows the hierarchical and containment/composition relationships between the classes.

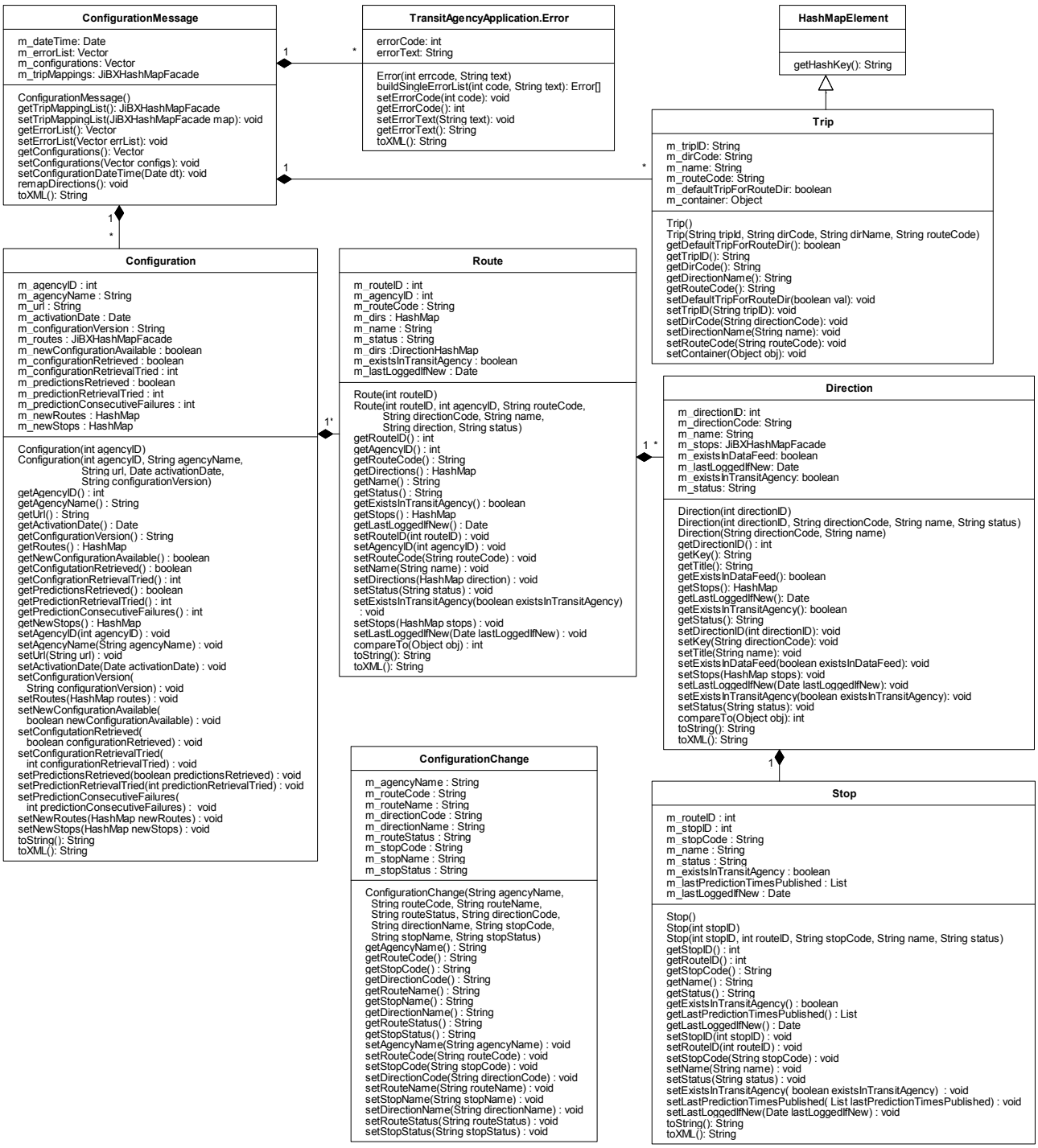

Figure 5. ConfigurationDataCD (Class Diagram)

3.3.4. Configuration (Class)

This class is used to create the Configuration Cache that is used throughout the TransitAgency application to store the current route/stop configuration data for each transit agency. It contains the list of currently active transit agencies that is used to determine which transit

agencies to retrieve predictions for.

3.3.5. ConfigurationChange (Class)

This class represents a ConfigurationChange data structure and contains added or removed route and stop information for a transit agency.

3.3.6. ConfigurationMessage (Class)

This class is used to hold the data used to create the Configuration Cache that is used throughout the TransitAgency application to store the current route/stop configuration data for each transit agency. It contains the list of currently active transit agencies that is used to determine which transit agencies to retrieve predictions for.

3.3.7. Direction (Class)

This class represents a Direction data structure and contains transit agency direction information.

3.3.8. Stop (Class)

This class represents a Stop data structure and contains transit agency stop information.

3.3.9. TransitAgencyApplication.Error (Class)

This class represents an Error data structure and contains information about any errors encountered during data retrieval.

3.3.10. Trip (Class)

This class represents a Trip data structure and contains information that maps agency trips with direction codes.

3.3.11. HashMapElement (Class)

This interface defines a method for obtaining a key from an object that will be stored in a Map object.

3.3.12. Route (Class)

This class represents a Route data structure and contains transit agency route information.

3.3.13. DataConnectionCD (Class Diagram)

This class diagram shows the relationship between the classes comprising connections to agency data sources. This diagram shows the hierarchical and containment/composition relationships between the classes.

Figure 6. DataConnectionCD (Class Diagram)

3.3.14. DataConnection (Class)

3.3.15. DataConnectionFactory (Class)

The DataConnectionFactory class creates DataConnection objects.

3.3.16. FTPDataConnection (Class)

This class provides information about the configuration properties needed to establish a connection to a FTP request/reply data source.

3.3.17. JMSPubSubscribeDataConnection (Class)

This class provides information about the configuration properties needed to establish a connection to a JMS publish/subscribe data source.

3.3.18. JMSRequestReplyDataConnection (Class)

This class provides information about the configuration properties needed to establish a connection to a JMS request/reply data source.

3.3.19. NextBusWSDataConnection (Class)

This class provides information about the configuration properties needed to establish a connection to a NextBus web service data source.

3.3.20. WSDataConnection (Class)

This class provides information about the configuration properties needed to establish a connection to a web service data source.

3.3.21. DataProcessorCD (Class Diagram)

This class diagram shows the relationship between the classes that receive data from agency data sources. This diagram shows the hierarchical and containment/composition relationships between the classes.

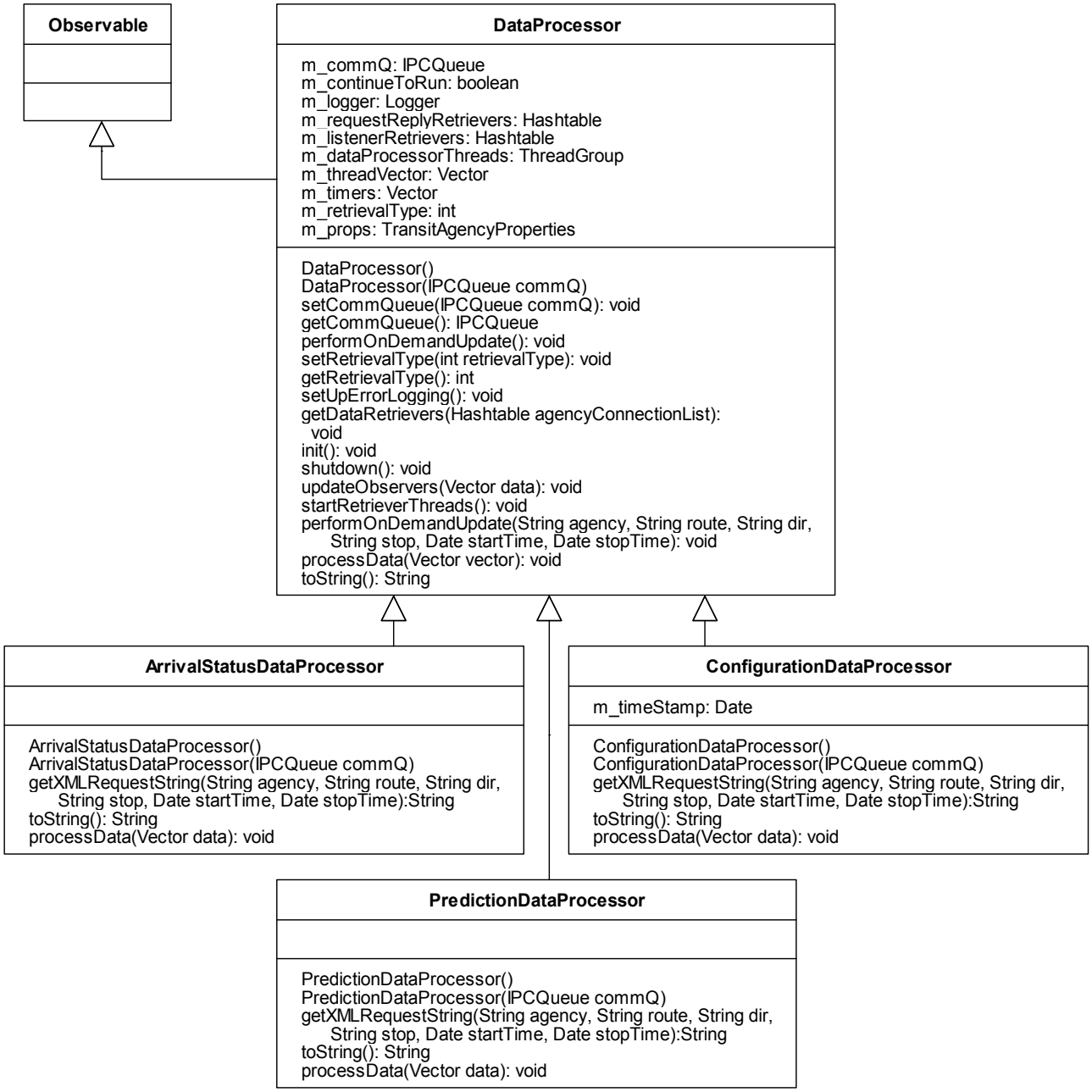

Figure 7. DataProcessorCD (Class Diagram)

3.3.22. ArrivalStatusDataProcessor (Class)

This class receives arrival status data sent by and/or requested from one or more data retrievers.

3.3.23. ConfigurationDataProcessor (Class)

This class receives configuration data sent by and/or requested from one or more data retrievers.

3.3.24. DataProcessor (Class)

This class is an abstract class that processes data received from one or more data retrievers.

3.3.25. Observable (Class)

This is a Java library class.

3.3.26. PredictionDataProcessor (Class)

This class receives prediction data sent by and/or requested from one or more data retrievers.

3.3.27. DataProcessorTaskCD (Class Diagram)

This class diagram shows the relationship between the classes that process data from agency data sources. This diagram shows the hierarchical and containment/composition relationships between the classes.

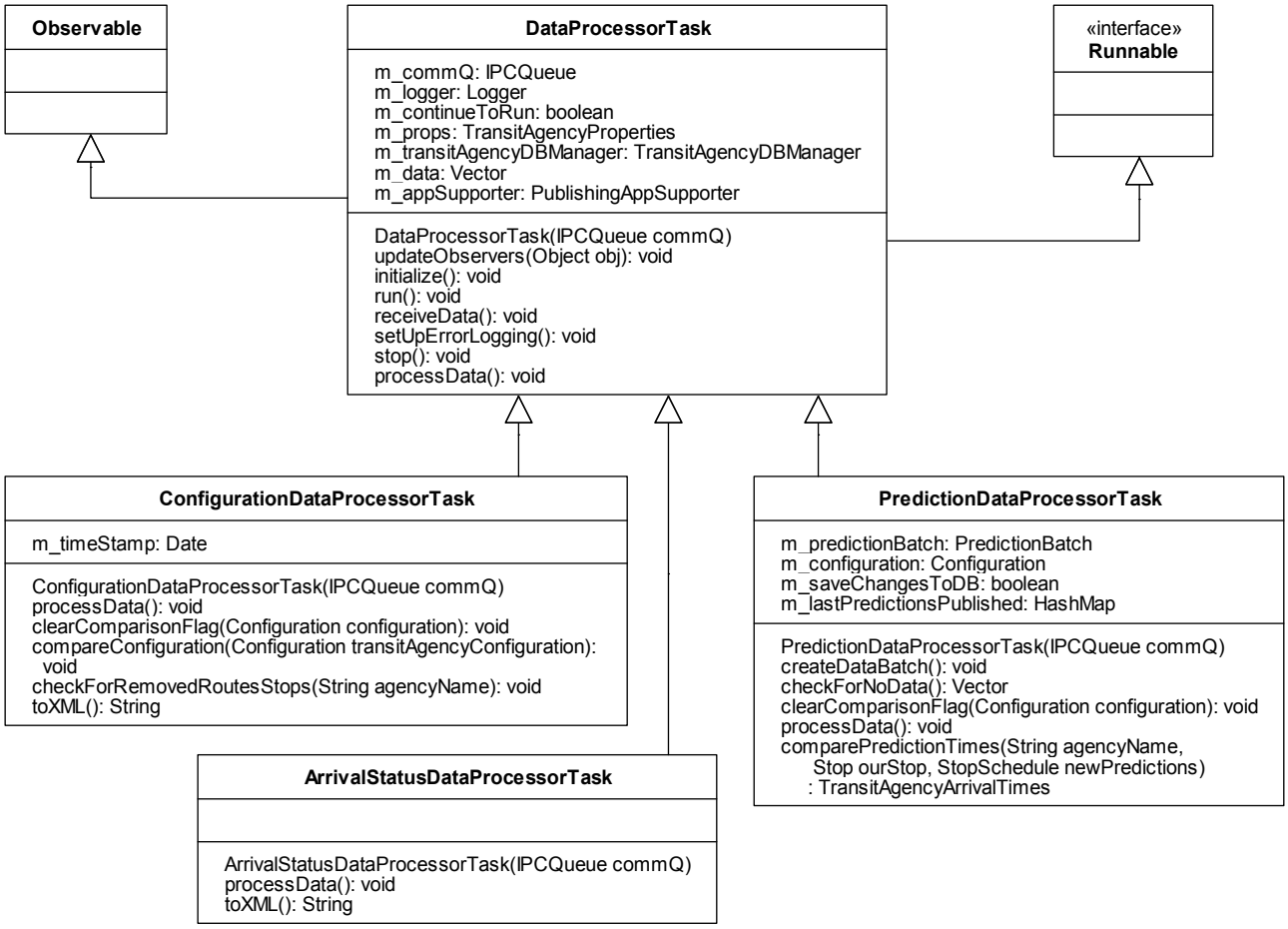

Figure 8. DataProcessorTaskCD (Class Diagram)

3.3.28. ArrivalStatusDataProcessorTask (Class)

This class runs in a separate thread and is responsible for processing agency vehicle arrival time data received from a ArrivalStatusDataProcessor object.

3.3.29. ConfigurationDataProcessorTask (Class)

This class runs in a separate thread and is responsible for processing agency configuration data received from a ConfigurationDataProcessor object.

3.3.30. DataProcessorTask (Class)

This class is responsible for processing data received from a DataProcessor object.

3.3.31. Observable (Class)

This is a Java library class.

3.3.32. PredictionDataProcessorTask (Class)

This class runs in a separate thread and is responsible for processing agency arrival time prediction data received from a PredictionDataProcessor object.

3.3.33. Runnable (Class)

This is a Java library class.

3.3.34. DataRetrieverCD (Class Diagram)

This class diagram shows the relationship between the classes that retrieve data from agency data sources. This diagram shows the hierarchical and containment/composition relationships between the classes.

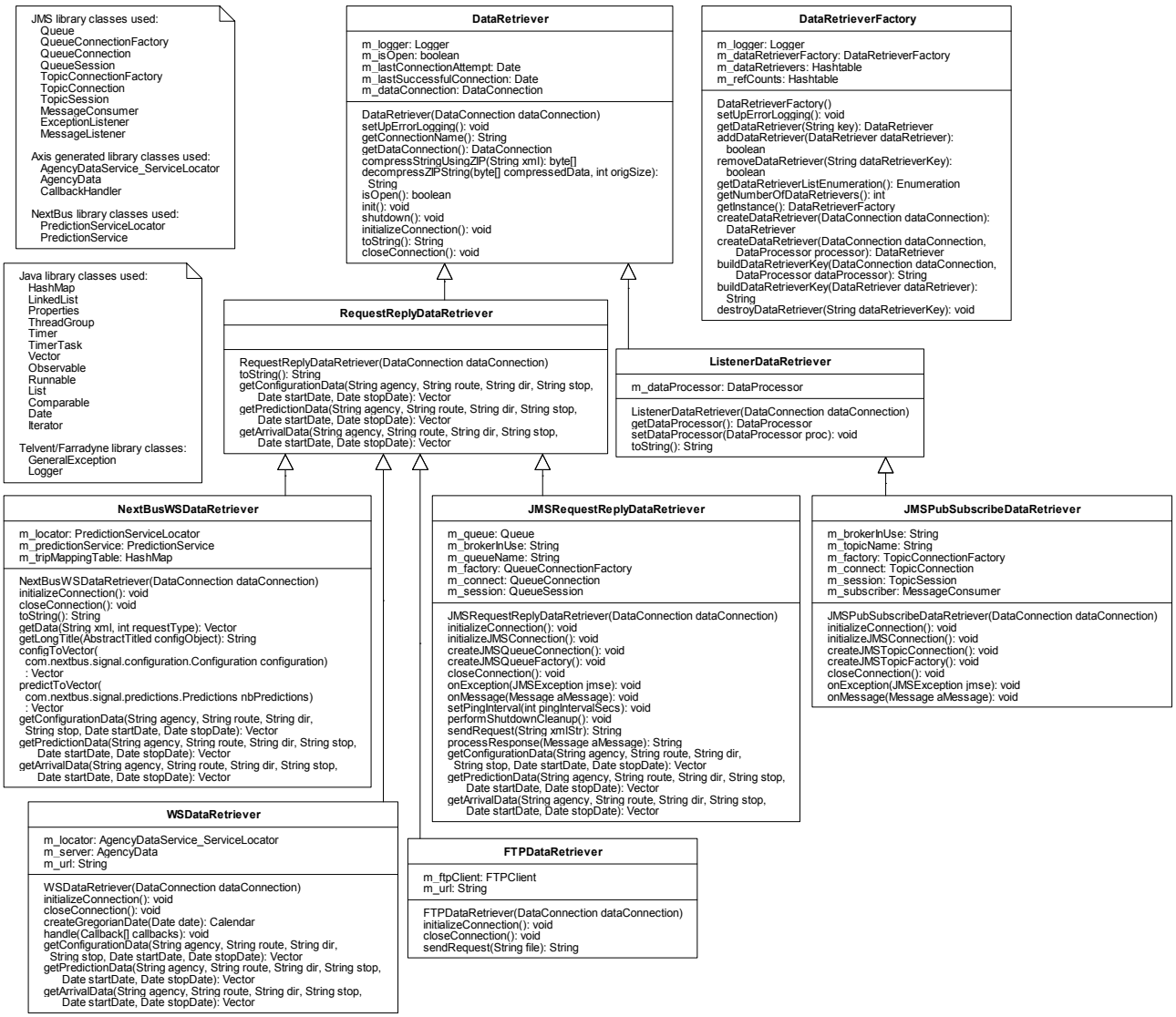

Figure 9. DataRetrieverCD (Class Diagram)

3.3.35. DataRetriever (Class)

The DataRetriever class is responsible for retrieving agency data from TransitAgencyApplication. and then passing it on to any and all Observers that have been registered.

3.3.36. DataRetrieverFactory (Class)

The DataRetrieverFactory class contains the list of data retrievers that are currently in use. Transit agencies that use the same connection parameters for retrieving data will use the same data retriever object.

3.3.37. FTPDataRetriever (Class)

The FTPDataRetriever class is responsible for retrieving agency data from an FTP server.

3.3.38. JMSPubSubscribeDataRetriever (Class)

This file contains the JMSPubSubscribeDataRetriever class which is responsible for retrieving agency configuration, arrival status, and prediction data from a JMS server using the publish/subscribe paradigm.

3.3.39. JMSRequestReplyDataRetriever (Class)

This file contains the JMSRequestReplyDataRetriever class which is responsible for retrieving agency configuration, arrival status, and prediction data from a JMS server using the request/reply paradigm.

3.3.40. ListenerDataRetriever (Class)

The ListenerDataRetriever class defines methods for subscription-based retrieval of data from transit agencies.

3.3.41. NextBusWSDataRetriever (Class)

The NextBusWSDataRetriever class is responsible for retrieving agency data from a NextBus web server using NextBus' defined interface.

3.3.42. RequestReplyDataRetriever (Class)

The RequestReplyDataRetriever class defines methods for on-demand retrieval of data from transit agencies.

3.3.43. WSDataRetriever (Class)

The WSDataRetriever class is responsible for retrieving agency data from a web server using web service calls.

3.3.44. PredictionDataCD (Class Diagram)

This class diagram shows the relationship between the classes comprising agency prediction data. This diagram shows the hierarchical and containment/composition relationships between the classes.

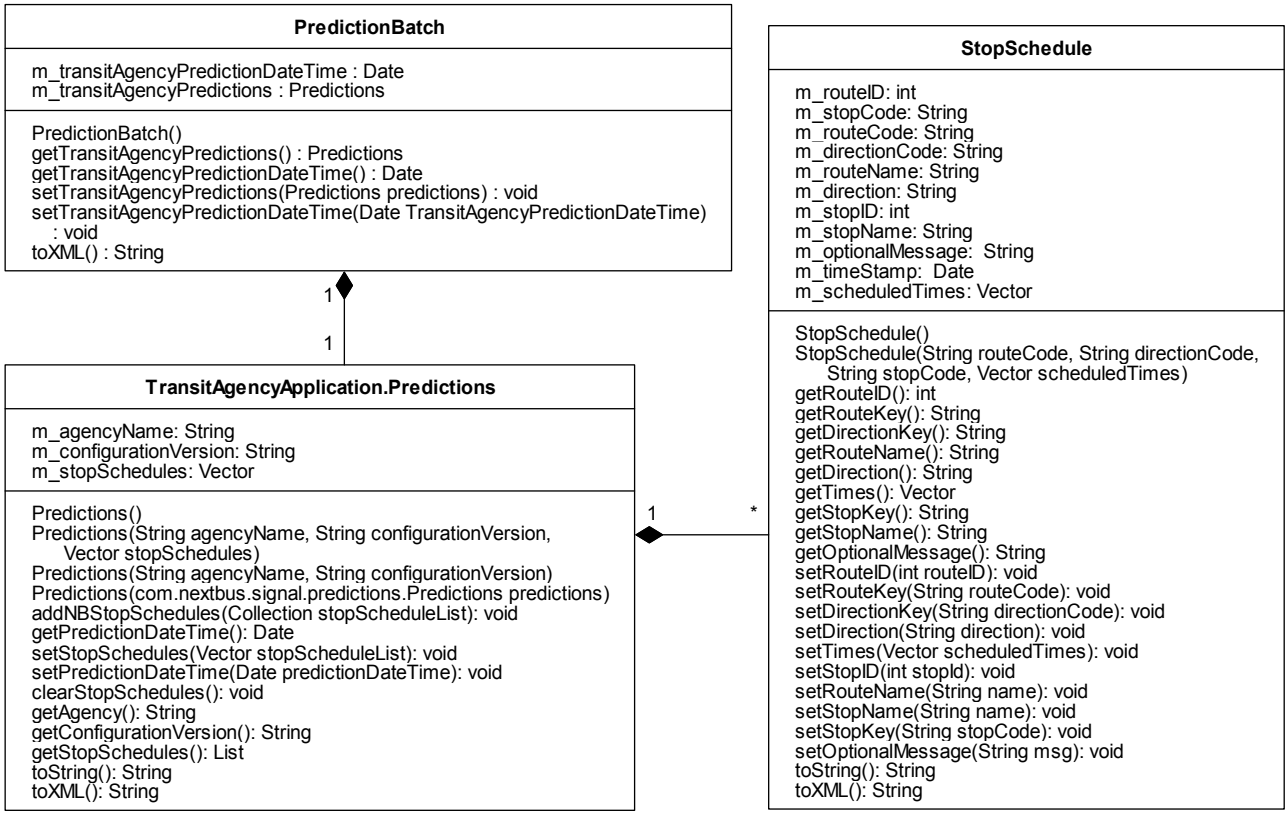

Figure 10. PredictionDataCD (Class Diagram)

3.3.45. PredictionBatch (Class)

This class is used to store a batch of predictions received from TransitAgencyApplication. This includes a timestamp, and a TransitAgencyApplication-defined Prediction object that contains the next arrival times.

3.3.46. StopSchedule (Class)

This class represents a StopSchedule data structure and contains transit arrival, and optionally departure time information.

3.3.47. TransitAgencyApplication.Predictions (Class)

This class is used to store a batch of predictions received. This includes a timestamp, and a Vector that contains StopSchedule objects containing the next arrival times.

3.3.48. TransitAgencyCD (Class Diagram)

This class diagram shows the relationship between the classes comprising the main class of the TransitAgencyApplication. This diagram shows the hierarchical and containment/composition relationships between the classes.

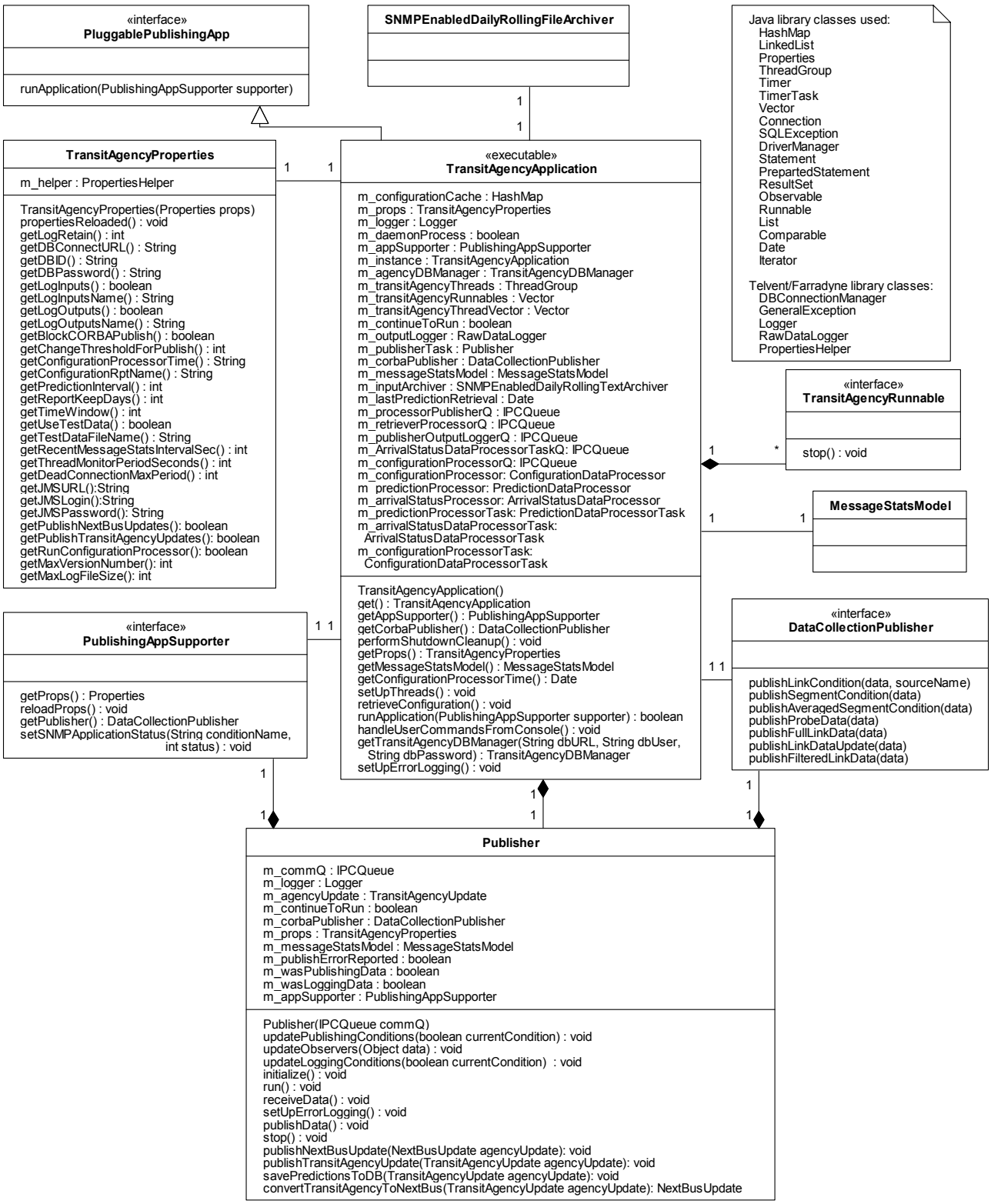

Figure 11. TransitAgencyCD (Class Diagram)

3.3.49. DataCollectionPublisher (Class)

This interface supports the publishing of the data that is output from the Data Collection subsystem.

3.3.50. MessageStatsModel (Class)

This class is used to keep track of statistics for all messages that are published and/or received.

3.3.51. PluggablePublishingApp (Class)

This interface exposes any necessary functionality from the DataCollection publishing application framework to support the needs of classes implementing the PluggablePublishingApp interface without exposing the specific implementation of the application framework.

3.3.52. PublishingAppSupporter (Class)

3.3.53. SNMPEnabledDailyRollingFileArchiver (Class)

This class logs entries in time-stamped files.

3.3.54. TransitAgencyApplication (Class)

The main class for the TransitAgencyApplication performs program setup and starts elements used to read and process incoming configuration, arrival status, and arrival prediction updates.

3.3.55. TransitAgencyRunnable (Class)

This interface defines the methods that are implemented by Runnable objects used within the TransitAgencyApplication Data Interface.

3.3.56. Publisher (Class)

The Publisher class is responsible for publication of processed prediction data to the CORBA server.

3.3.57. TransitAgencyProperties (Class)

This class loads application properties specific to the TransitAgencyApplication application.

3.3.58. UtilitiesCD (Class Diagram)

This class diagram shows classes that are not part of the main class hierarchy that will be used by the application.

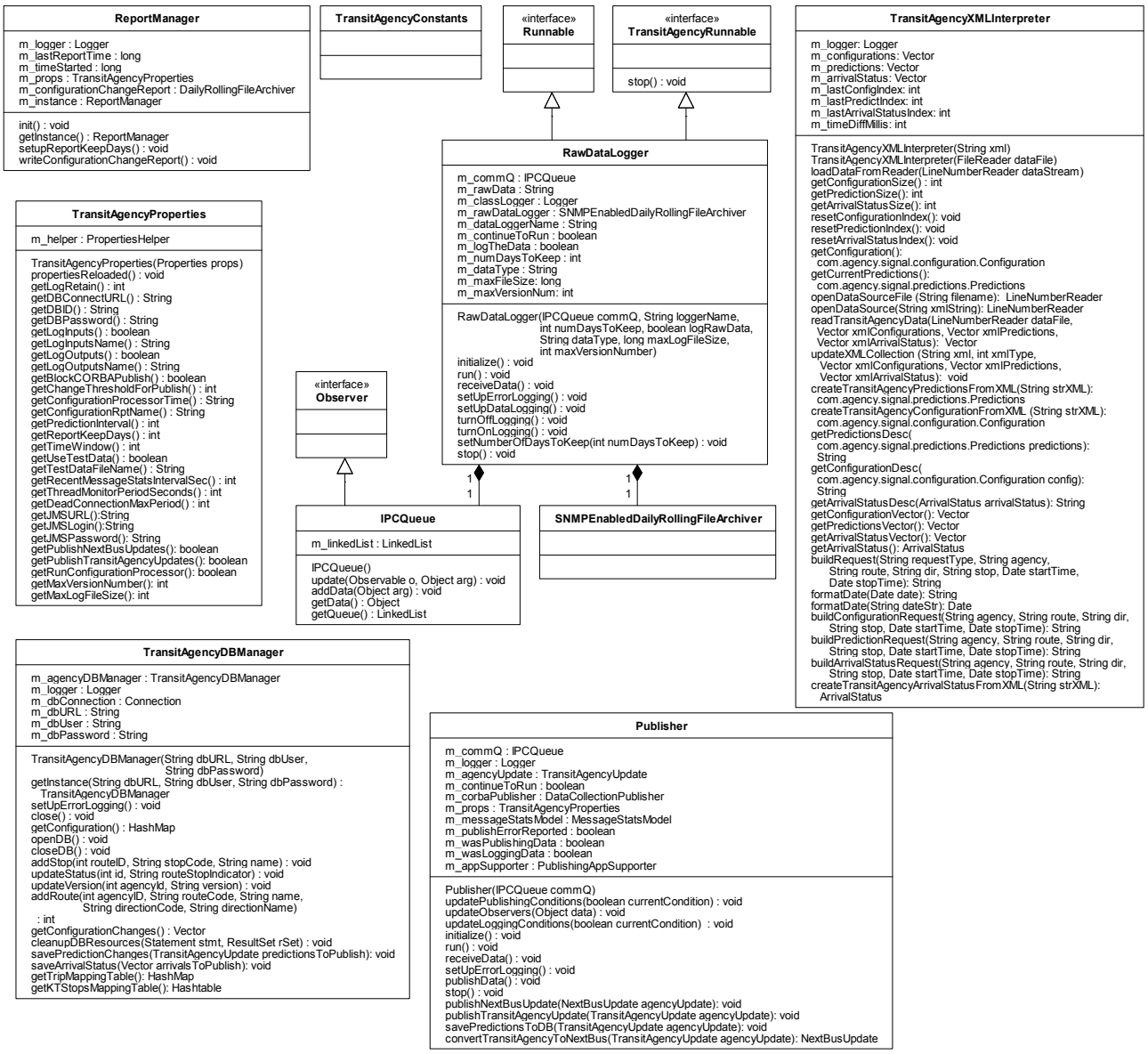

Figure 12. UtilitiesCD (Class Diagram)

3.3.59. IPCQueue (Class)

3.3.60. Observer (Class)

3.3.61. RawDataLogger (Class)

RawDataLogger objects are runnable objects that can be placed in their own thread of execution. Upon creation, the intraprocess communicaton queue (what is observed) and the name of the log file are specified. The RawDataLogger object is added as an observer to the IPCQueue and logs the data received upon update notifications.

3.3.62. ReportManager (Class)

This class is responsible for managing the reports generated by TransitAgencyApplication.

3.3.63. Runnable (Class)

This is a Java library class.

3.3.64. SNMPEnabledDailyRollingFileArchiver (Class)

This class logs entries in time-stamped files.

3.3.65. Publisher (Class)

The Publisher class is responsible for publication of processed prediction data to the CORBA server.

3.3.66. TransitAgencyConstants (Class)

This class defines the constants associated with the TransitAgencyApplication Data Interface.

3.3.67. TransitAgencyDBManager (Class)

The TransitAgencyDBManager is a singleton class responsible for handling the interface with the TravInfo DCOL database for the TransitAgencyApplication Data Interface. Responsibilities of this class include setting up and maintaining the connection to the TravInfo DCOL database, reading configuration data, and providing support for updating the TransitAgencyApplication database tables as needed. Methods in this class are specific to the needs of the TransitAgencyApplication Data Interface.

3.3.68. TransitAgencyRunnable (Class)

This interface defines the methods that are implemented by Runnable objects used within the TransitAgencyApplication Data Interface.

3.3.69. TransitAgencyProperties (Class)

This class loads application properties specific to the TransitAgencyApplication application.

3.3.70. TransitAgencyXMLInterpreter (Class)

This class contains the TransitAgencyXMLInterpreter that converts XML string data into data structures used by the rest of the TransitAgencyApplication service.

3.4 Sequence Diagrams

3.4.1. ArrivalStatusDataProcessorTask:processData (Sequence Diagram)

This sequence diagram shows the steps necessary to process received vehicle arrival time data.

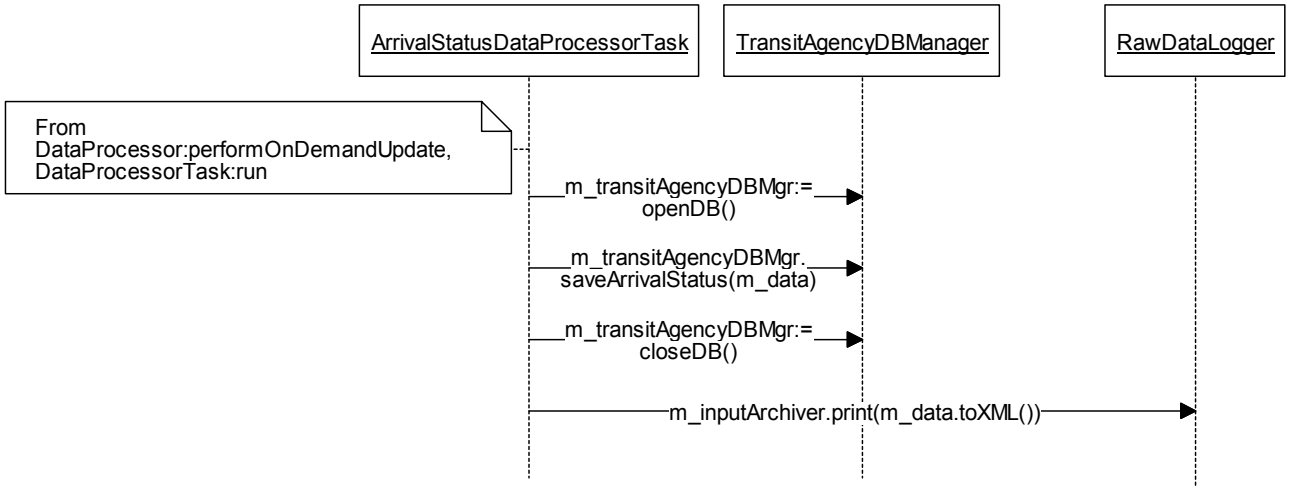

Figure 13. ArrivalStatusDataProcessorTask:processData (Sequence Diagram)

3.4.2. DataProcessor:init (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of a data processor task. Data processor tasks retrieve data from various data sources as defined for each agency in the system database, and if necessary, converts the data from the different sources into a common format for processing.

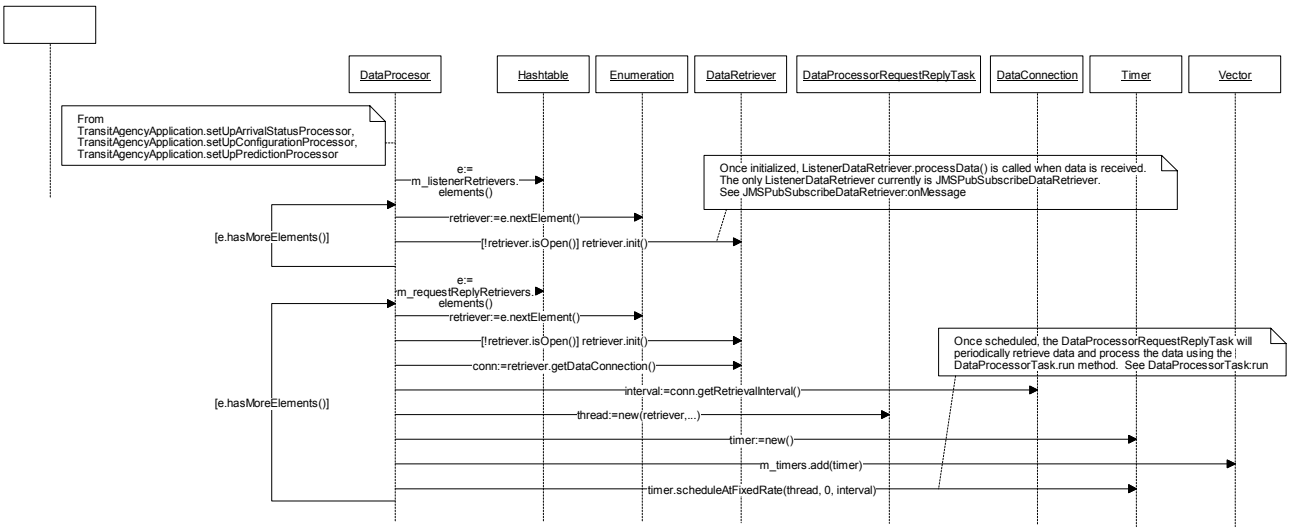

Figure 14. DataProcessor:init (Sequence Diagram)

3.4.3. DataProcessor:performOnDemandUpdate (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated configuration, prediction, or arrival status data for each agency that supports on-demand updates.

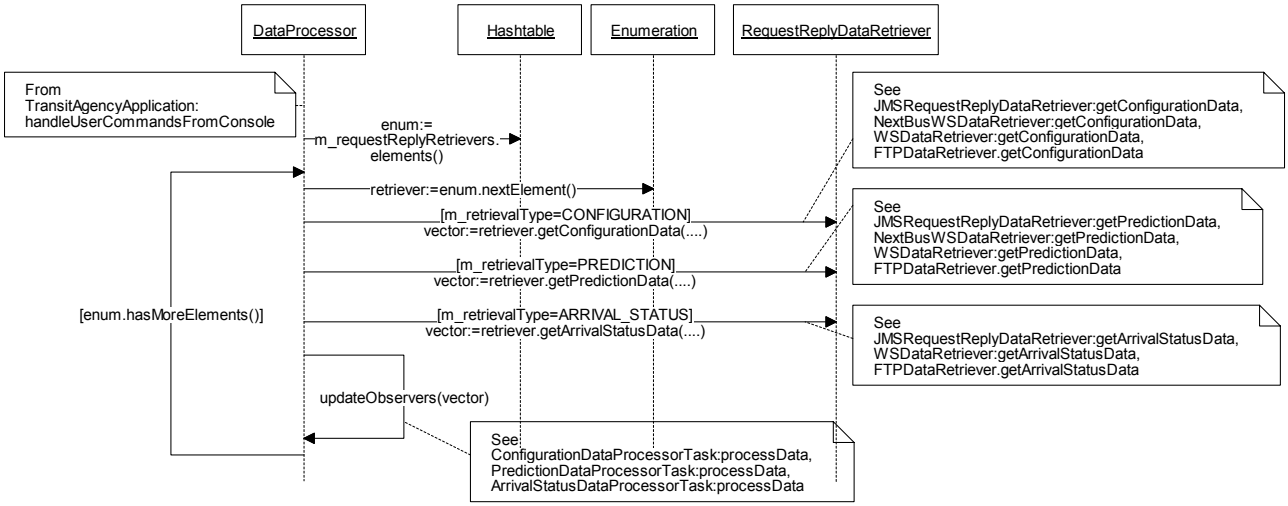

Figure 15. DataProcessor:performOnDemandUpdate (Sequence Diagram)

3.4.4. ConfigurationDataProcessorTask:processData (Sequence Diagram)

This sequence diagram shows the steps necessary to process received agency configuration data.

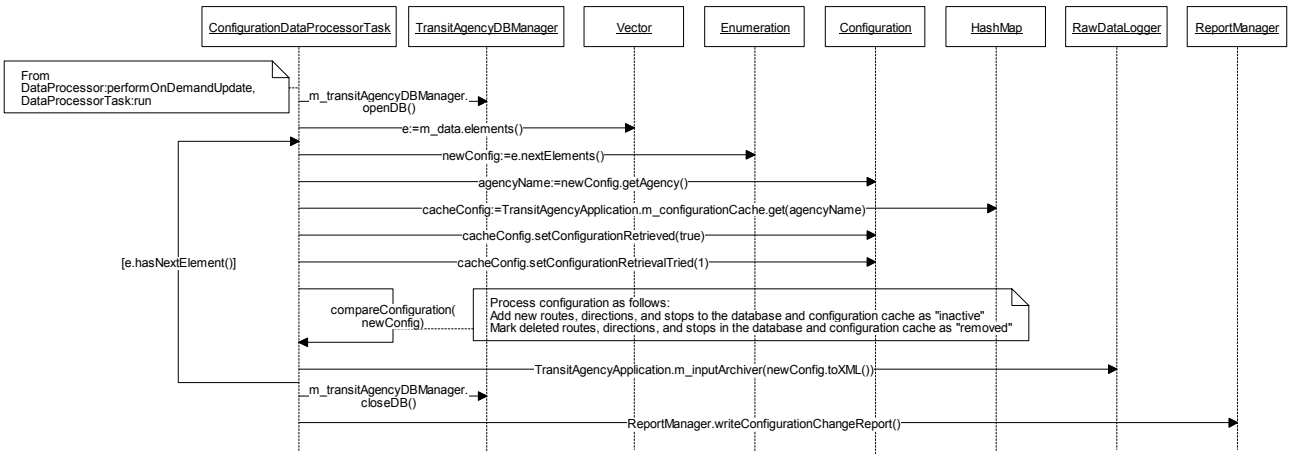

Figure 16. ConfigurationDataProcessorTask:processData (Sequence Diagram)

3.4.5. DataProcessorTask:run (Sequence Diagram)

This sequence diagram shows the steps necessary to process received data.

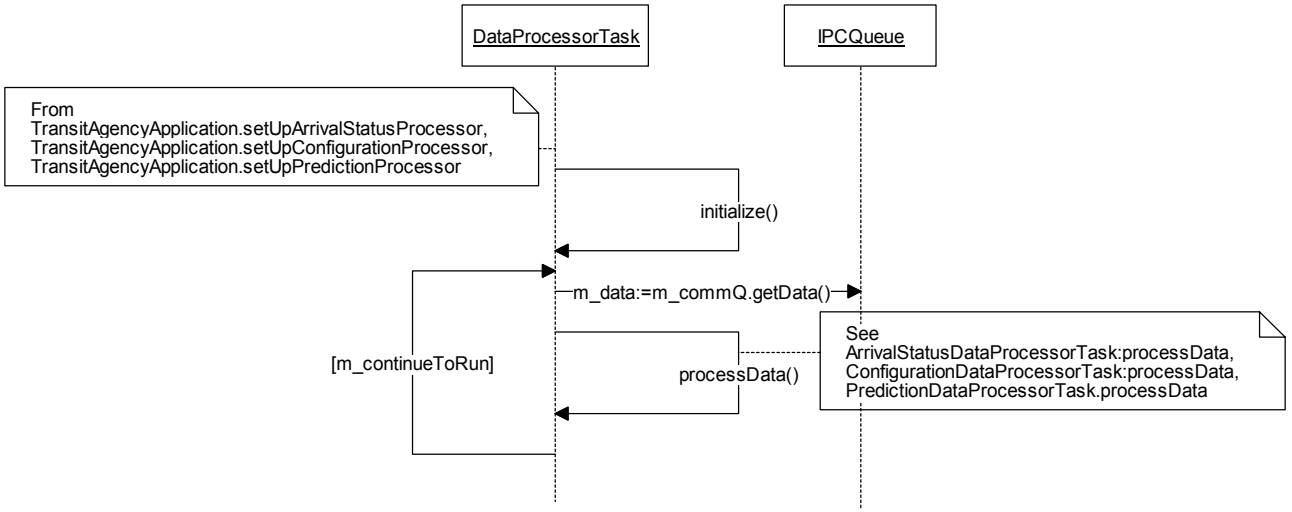

Figure 17. DataProcessorTask:run (Sequence Diagram)

3.4.6. FTPDataRetriever:getConfigurationData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated agency configuration data via an FTP server.

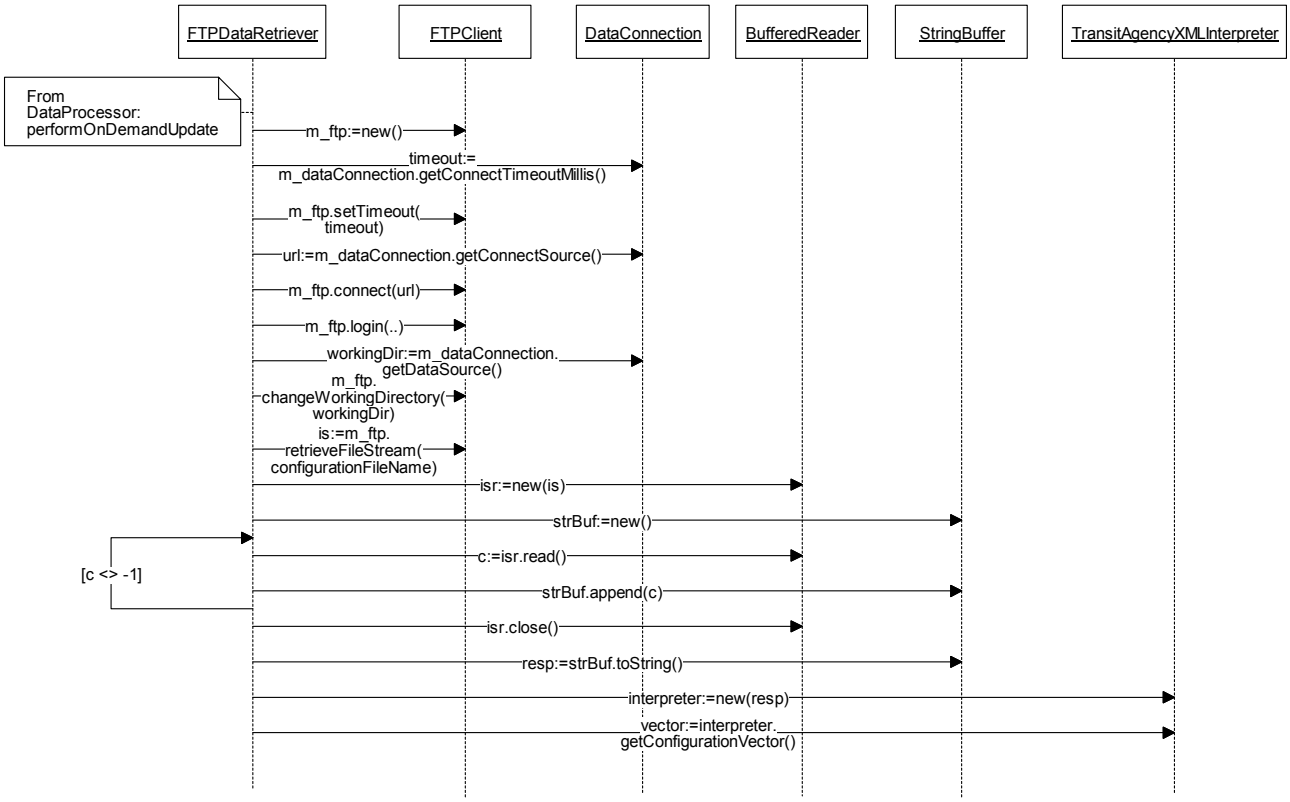

Figure 18. FTPDataRetriever:getConfigurationData (Sequence Diagram)

3.4.7. TransitAgencyApplication:handleUserCommandsFromConsole (Sequence Diagram)

This sequence diagram shows the steps necessary to handle user commands from the console. The user may enter two commands: 'c' to retrieve updated configuration data for each agency, 'q' to shutdown the system.

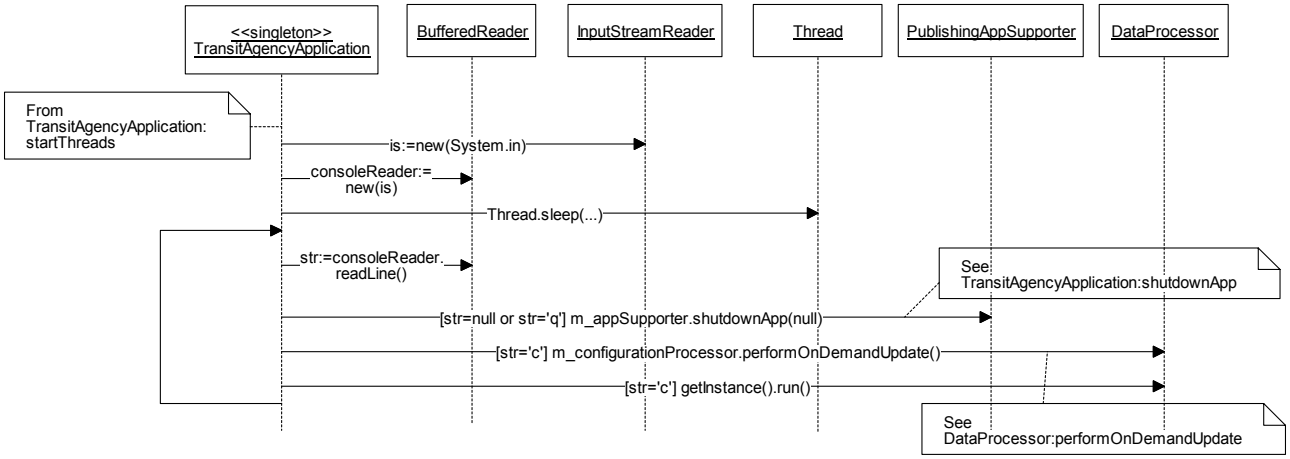

Figure 19. TransitAgencyApplication:handleUserCommandsFromConsole (Sequence Diagram)

3.4.8. TransitAgencyApplication:retrieveConfiguration (Sequence Diagram)

This sequence diagram shows the process for retrieving configuration information from the database upon system startup.

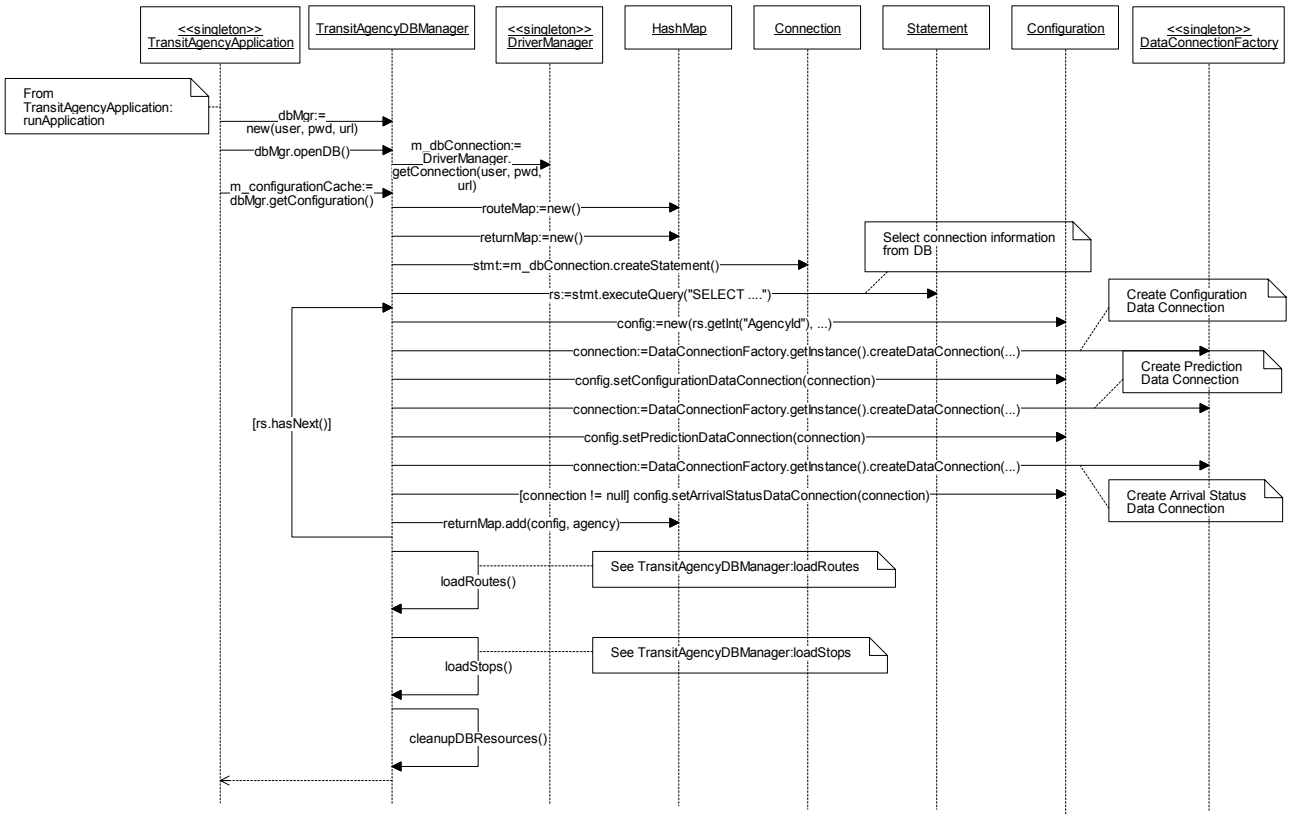

Figure 20. TransitAgencyApplication:retrieveConfiguration (Sequence Diagram)

3.4.9. TransitAgencyApplication:runApplication (Sequence Diagram)

This sequence diagram shows the initialization of the TransitAgencyApplication: retrieving the initial set of configurations from the database and starting the main threads of the application that handle reading console input, logging, and processing configuration, arrival status, and prediction updates.

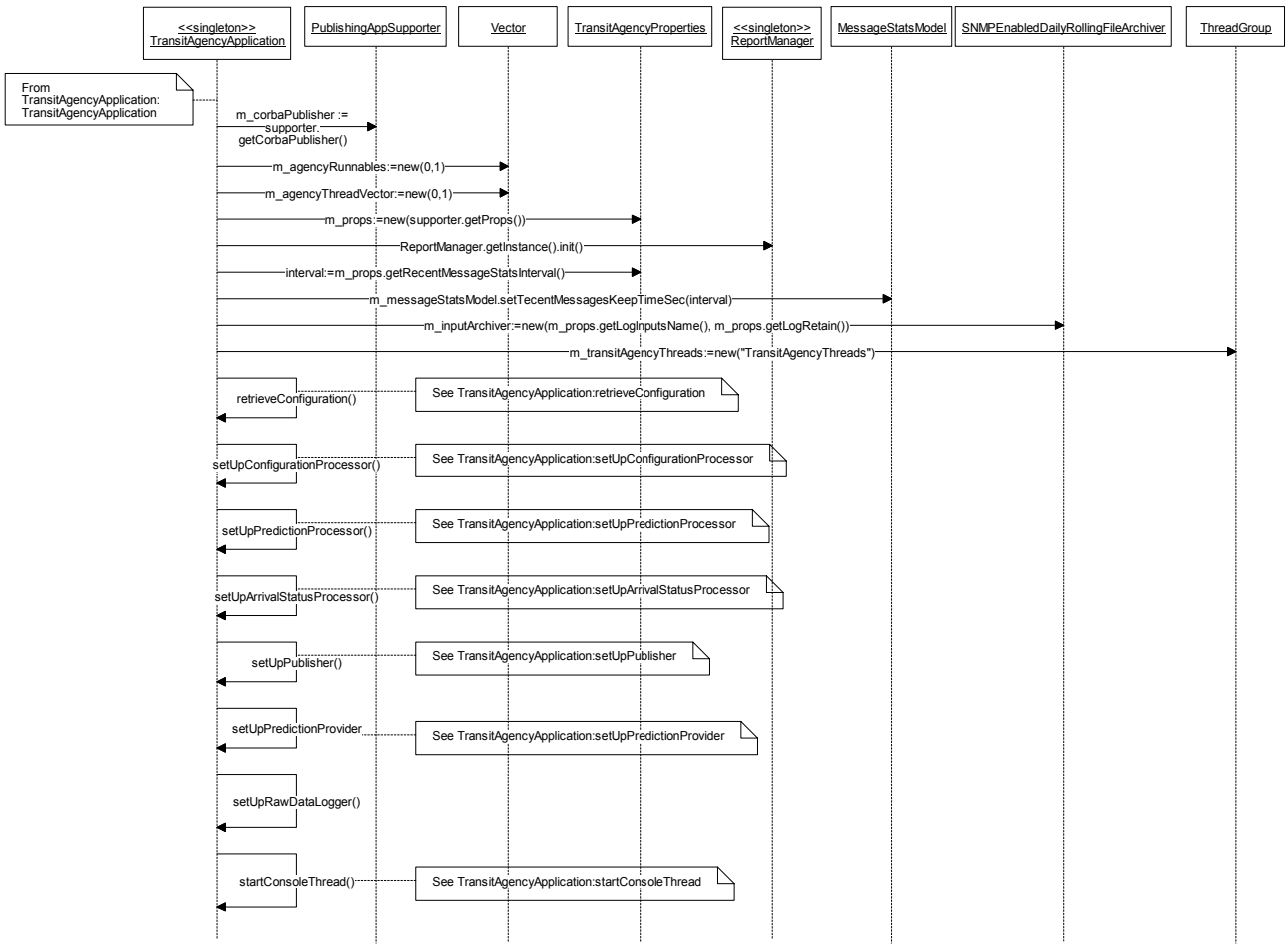

Figure 21. TransitAgencyApplication:runApplication (Sequence Diagram)

3.4.10. TransitAgencyApplication:setUpArrivalStatusProcessor (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of the arrival status data processor task.

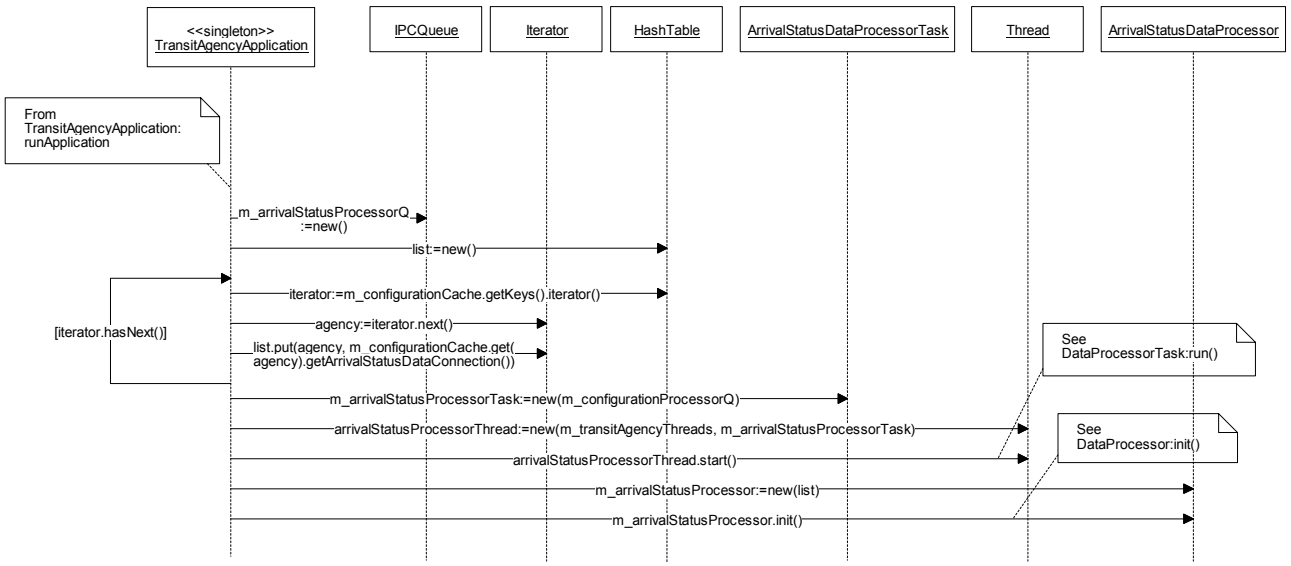

Figure 22. TransitAgencyApplication:setUpArrivalStatusProcessor (Sequence Diagram)

3.4.11. TransitAgencyApplication:setUpConfigurationProcessor (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of the agency configuration data processor task.

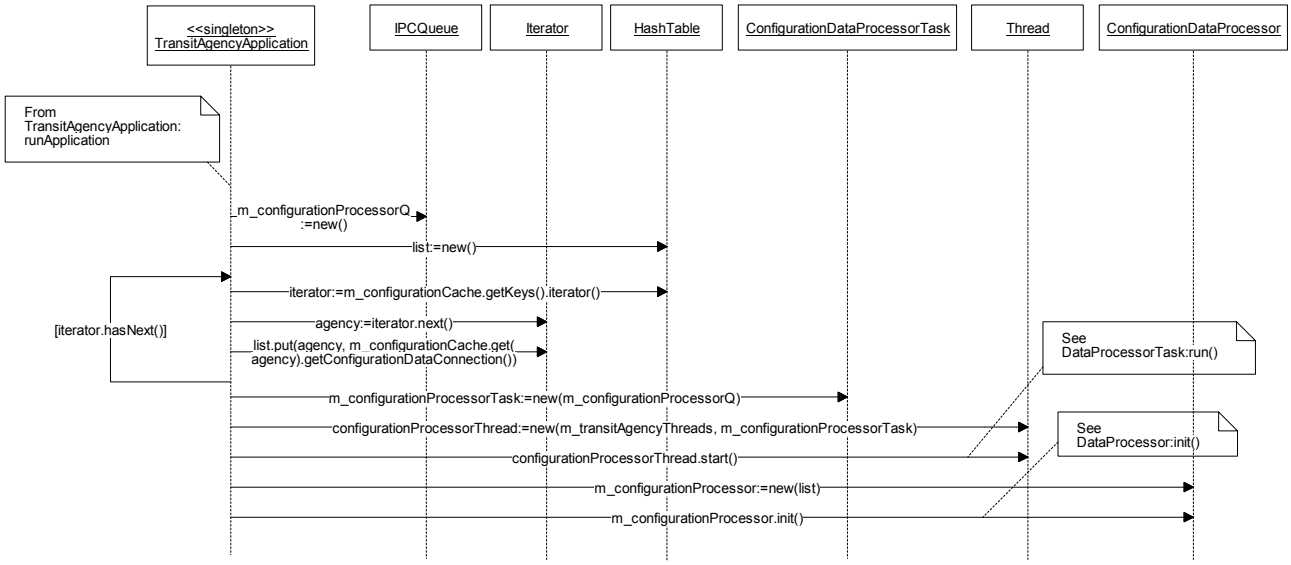

Figure 23. TransitAgencyApplication:setUpConfigurationProcessor (Sequence Diagram)

3.4.12. TransitAgencyApplication:setUpPredictionProcessor (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of the vehicle arrival time prediction data processor task.

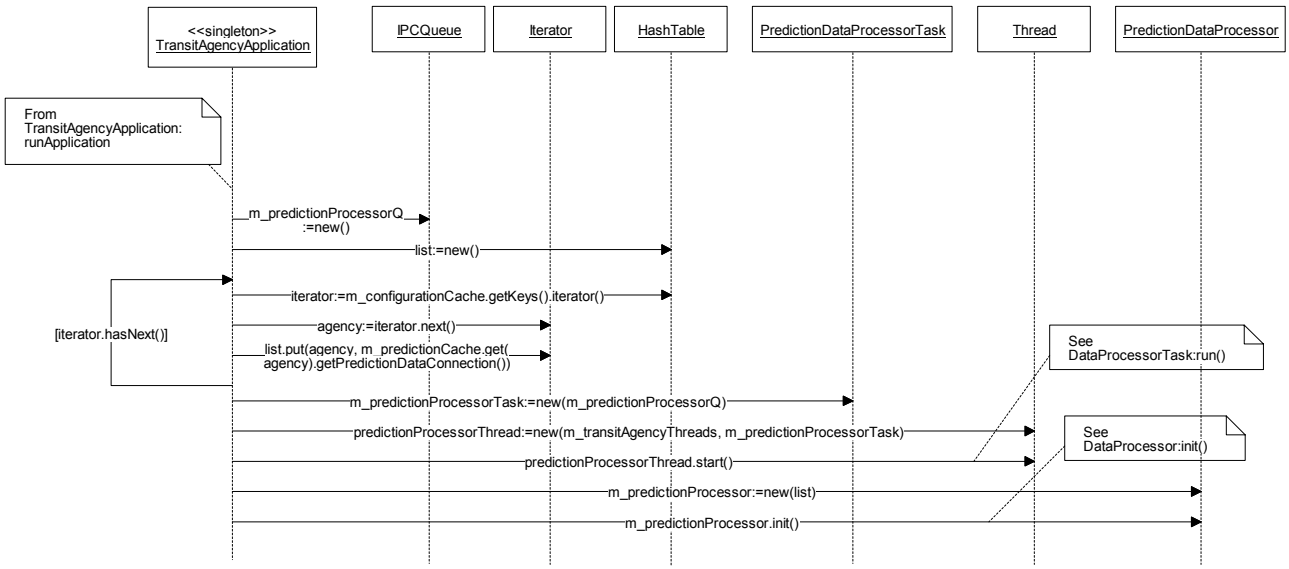

Figure 24. TransitAgencyApplication:setUpPredictionProcessor (Sequence Diagram)

3.4.13. TransitAgencyApplication:shutdown (Sequence Diagram)

This sequence diagram shows the methods that are called when a shutdown notification is received. When the system is shutdown, all running threads must be gracefully stopped and an SNMP message is sent indicating that the application is shutting down.

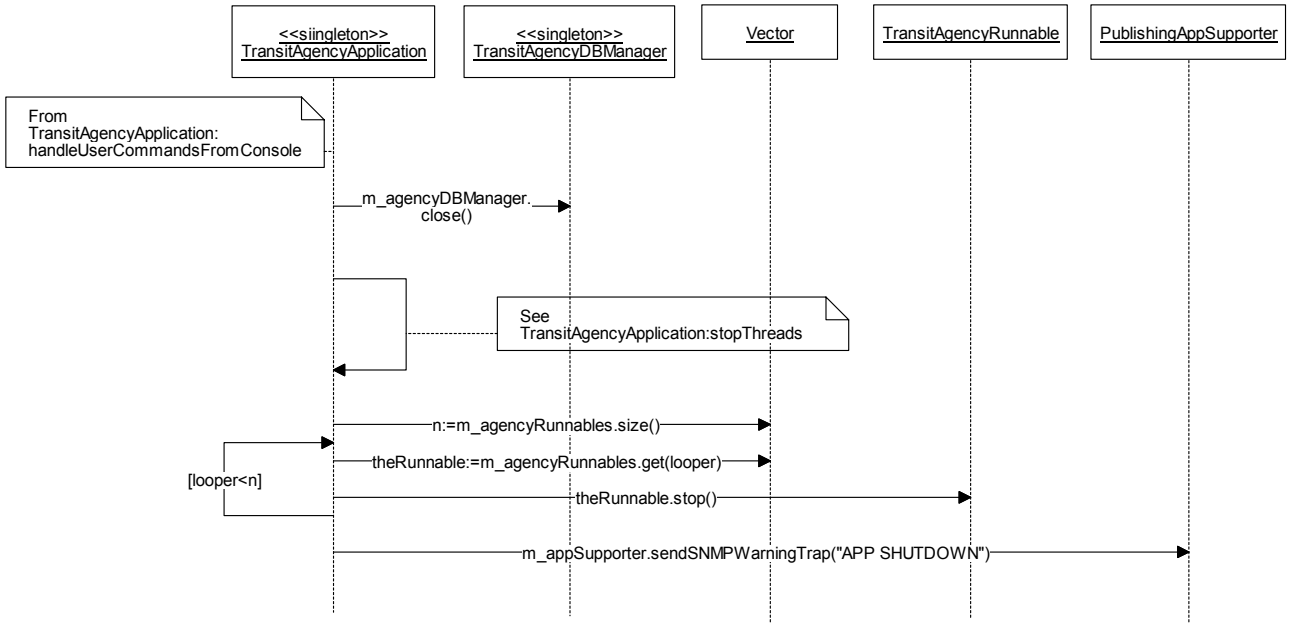

Figure 25. TransitAgencyApplication:shutdown (Sequence Diagram)

3.4.14. TransitAgencyApplication:startConsoleThread (Sequence Diagram)

This sequence diagram shows the methods that are called to start console thread. The console thread handles user requests to update the agency configuratons on-demand, and requests from the user to shutdown the application.

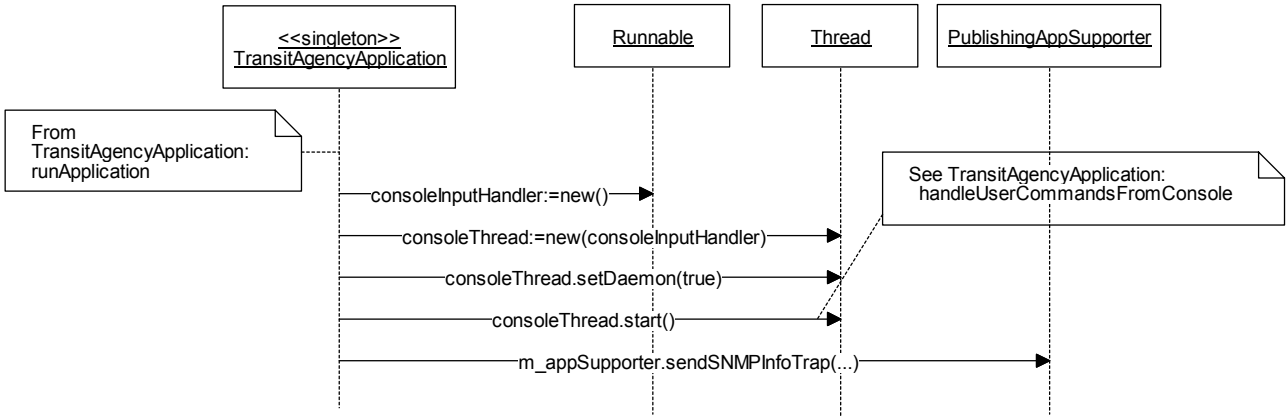

Figure 26. TransitAgencyApplication:startConsoleThread (Sequence Diagram)

3.4.15. TransitAgencyApplication:stopThreads (Sequence Diagram)

This sequence diagram shows the methods that are called to shutdown the DataProcessor and DataProcessorTask threads.

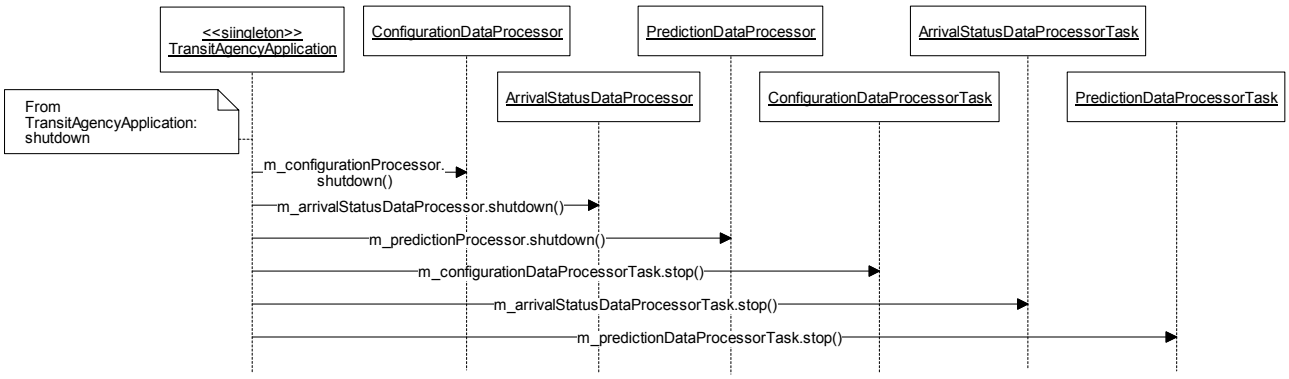

Figure 27. TransitAgencyApplication:stopThreads (Sequence Diagram)

3.4.16. TransitAgencyApplication:TransitAgencyApplication (Sequence Diagram)

This sequence diagram shows the methods that are called when a TransitAgencyApplication is created.

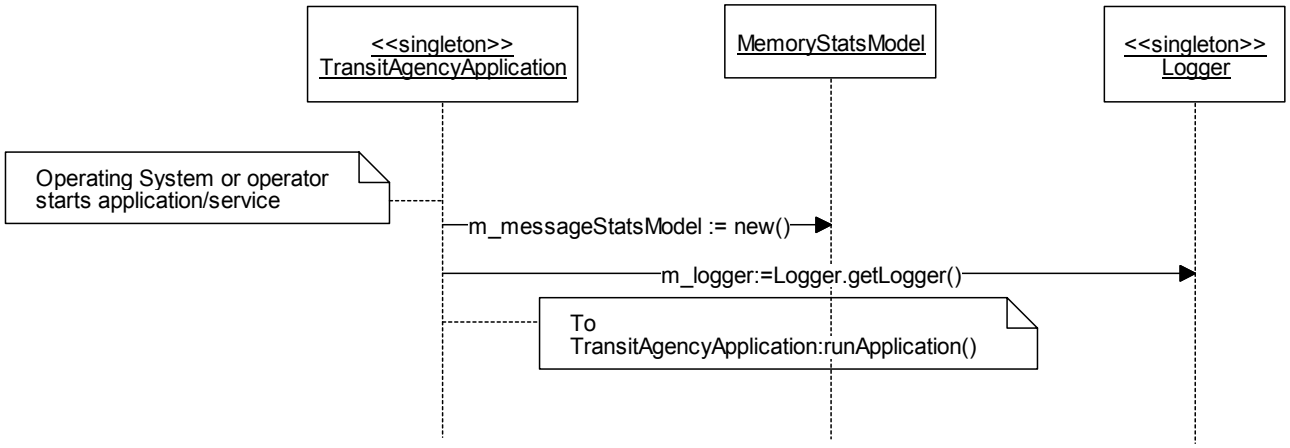

Figure 28. TransitAgencyApplication:TransitAgencyApplication (Sequence Diagram)

3.4.17. TransitAgencyDBManager:LoadRoutes (Sequence Diagram)

This sequence diagram shows the process for retrieving route configuration information from the database upon system startup.

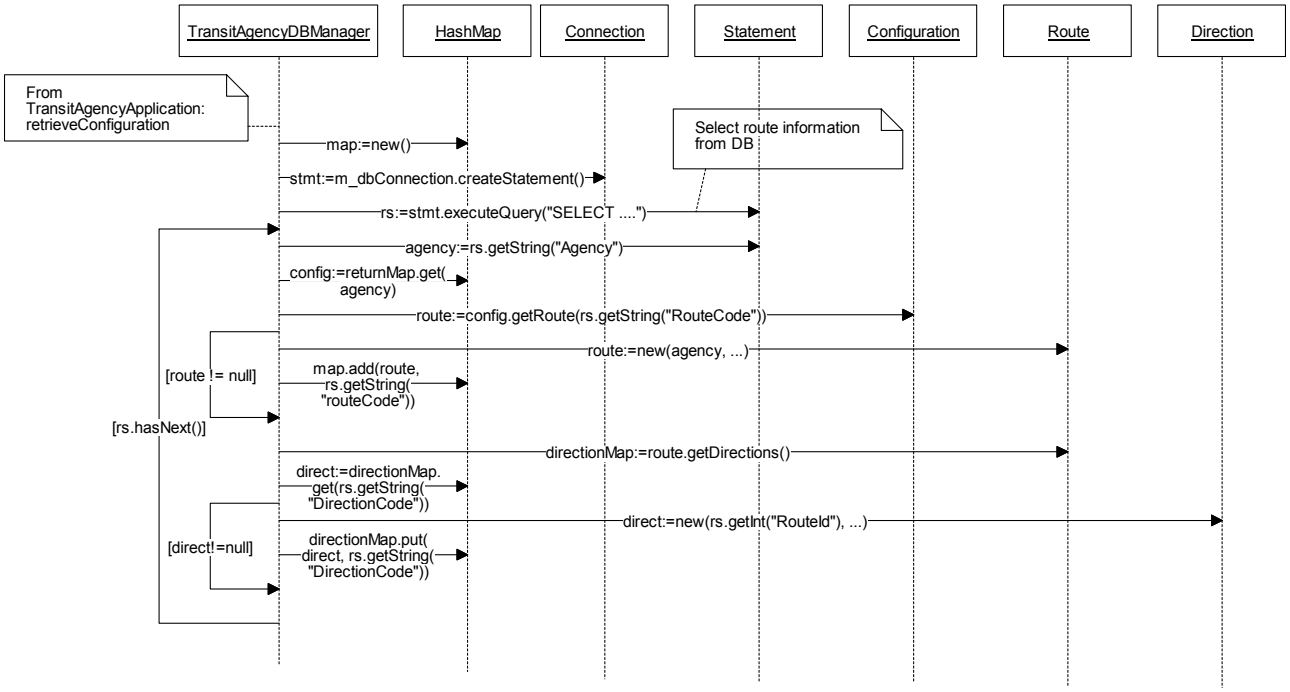

Figure 29. TransitAgencyDBManager:LoadRoutes (Sequence Diagram)

3.4.18. TransitAgencyDBManager:LoadStops (Sequence Diagram)

This sequence diagram shows the process for retrieving stop configuration information from the database upon system startup.

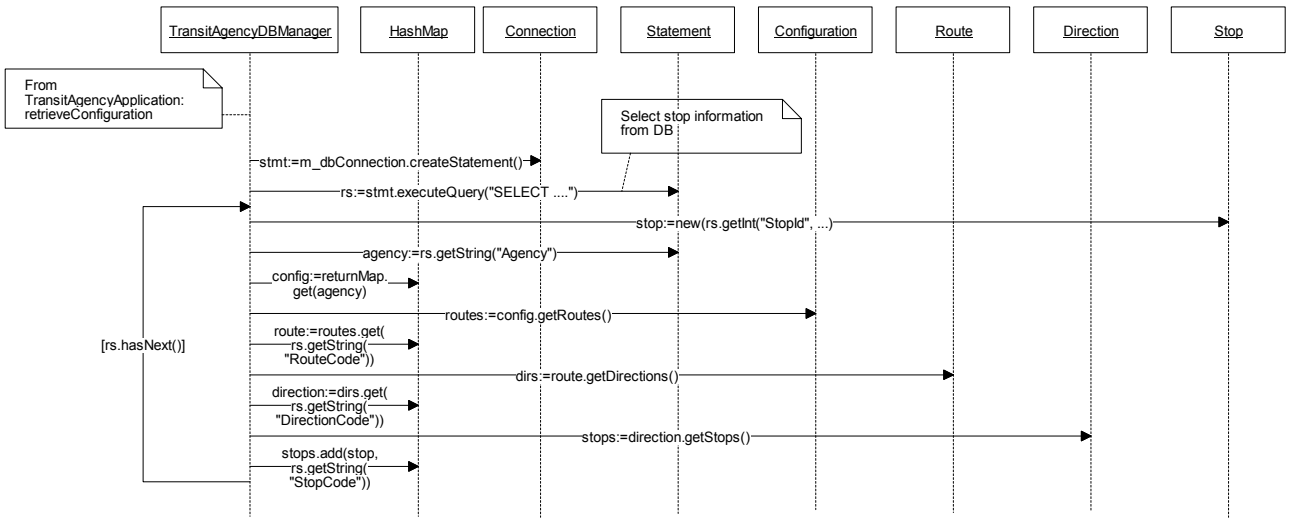

Figure 30. TransitAgencyDBManager:LoadStops (Sequence Diagram)

3.4.19. FTPDataRetriever:getPredictionData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated prediction data via an FTP server.

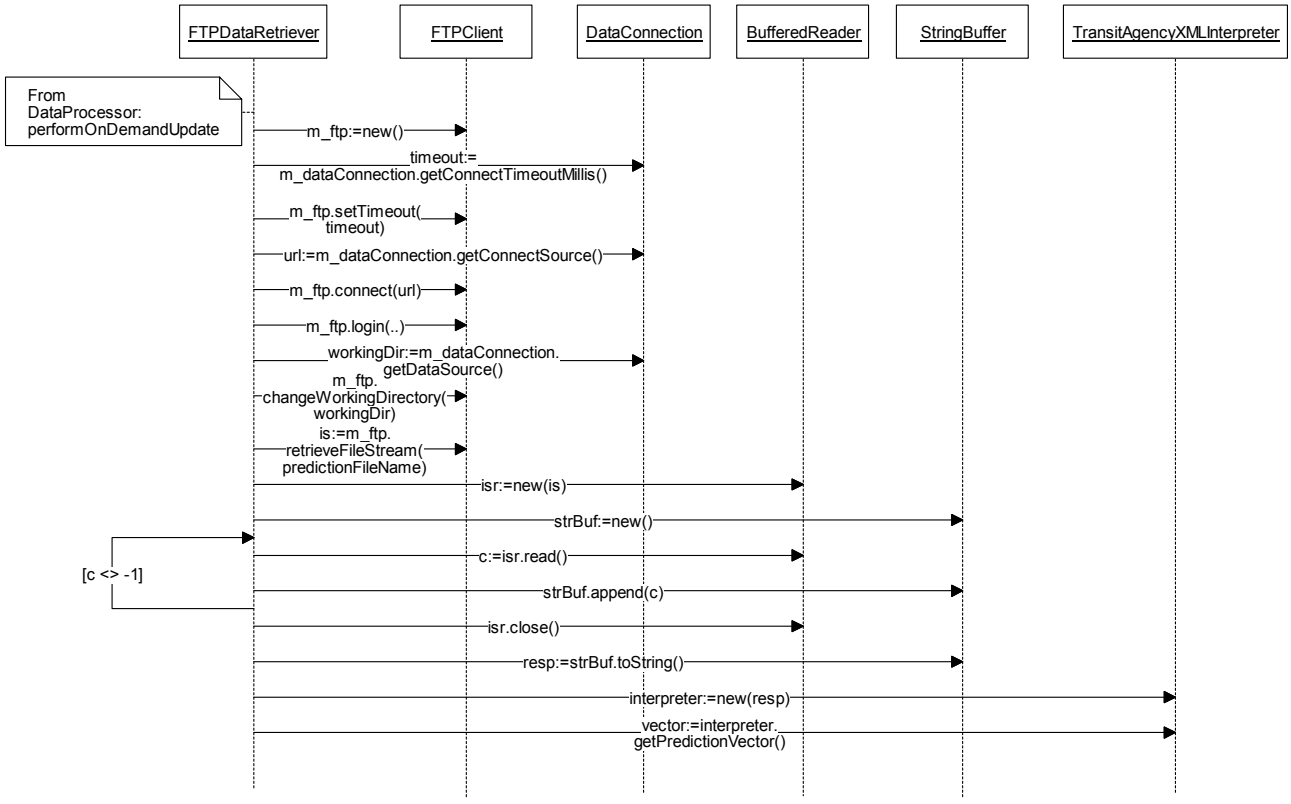

Figure 31. FTPDataRetriever:getPredictionData (Sequence Diagram)

3.4.20. JMSRequestReplyDataRetriever:getConfigurationData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated configuration data via the Java Message Service.

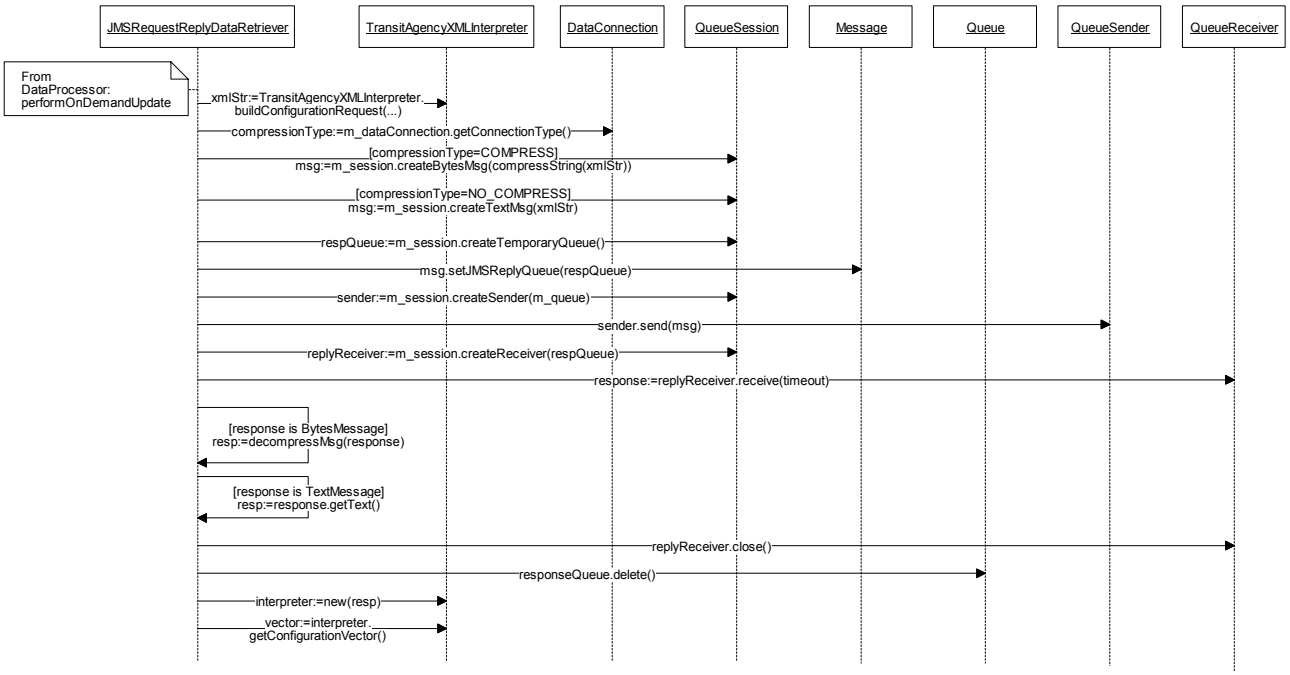

Figure 32. JMSRequestReplyDataRetriever:getConfigurationData (Sequence Diagram)

3.4.21. JMSRequestReplyDataRetriever:getPredictionData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated prediction data via the Java Message Service.

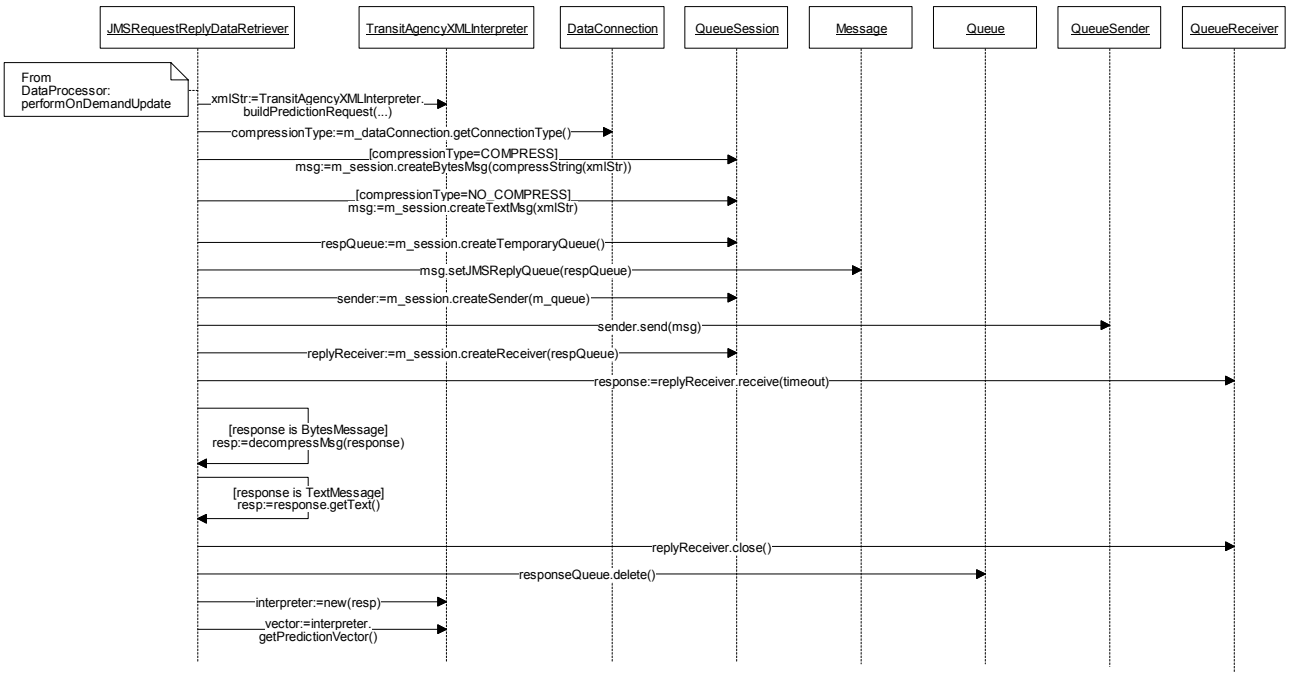

Figure 33. JMSRequestReplyDataRetriever:getPredictionData (Sequence Diagram)

3.4.22. NextBusWSDataRetriever:getConfigurationData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated configuration data via the objects provided by NextBus.

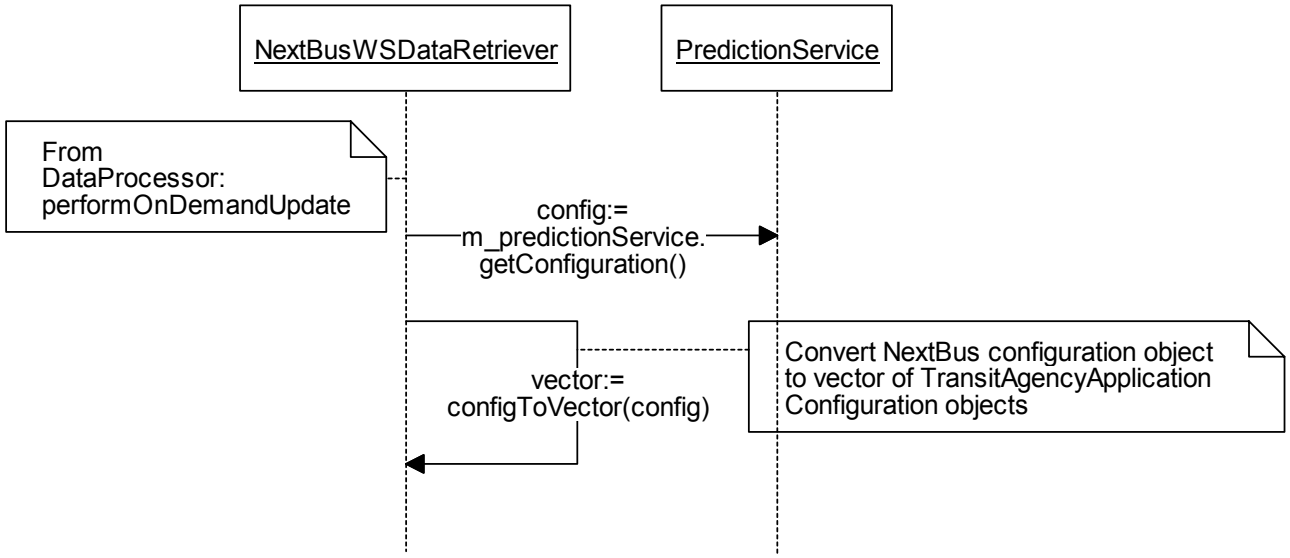

Figure 34. NextBusWSDataRetriever:getConfigurationData (Sequence Diagram)

3.4.23. NextBusWSDataRetriever:getPredictionData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated prediction data via the objects provided by NextBus.

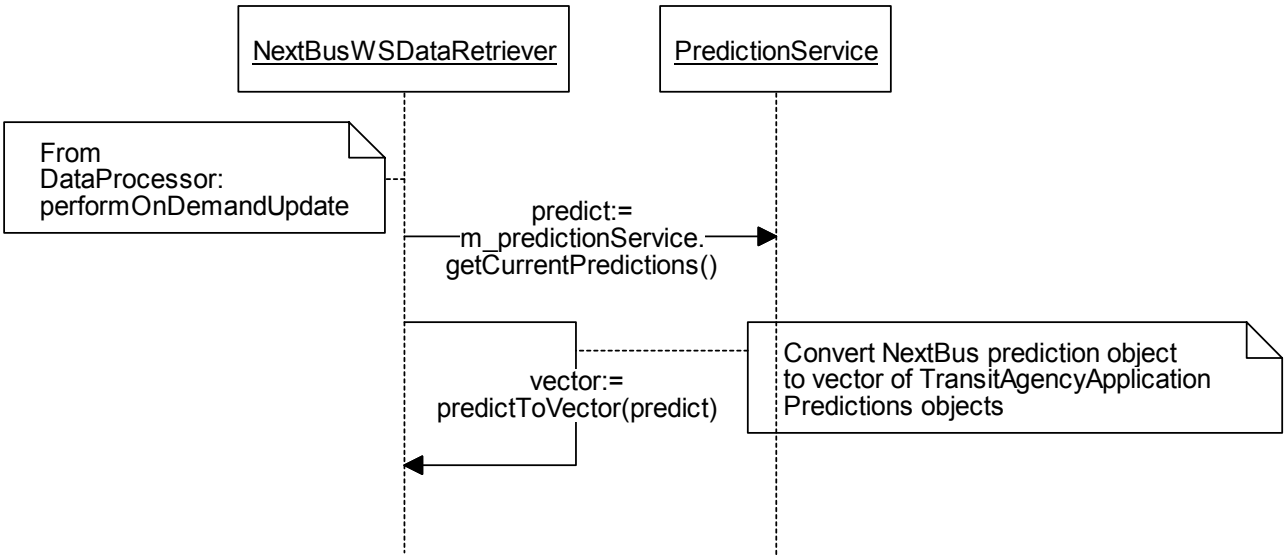

Figure 35. NextBusWSDataRetriever:getPredictionData (Sequence Diagram)

3.4.24. PredictionDataProcessorTask:processData (Sequence Diagram)

This sequence diagram shows the steps necessary to process received agency arrival time prediction data.

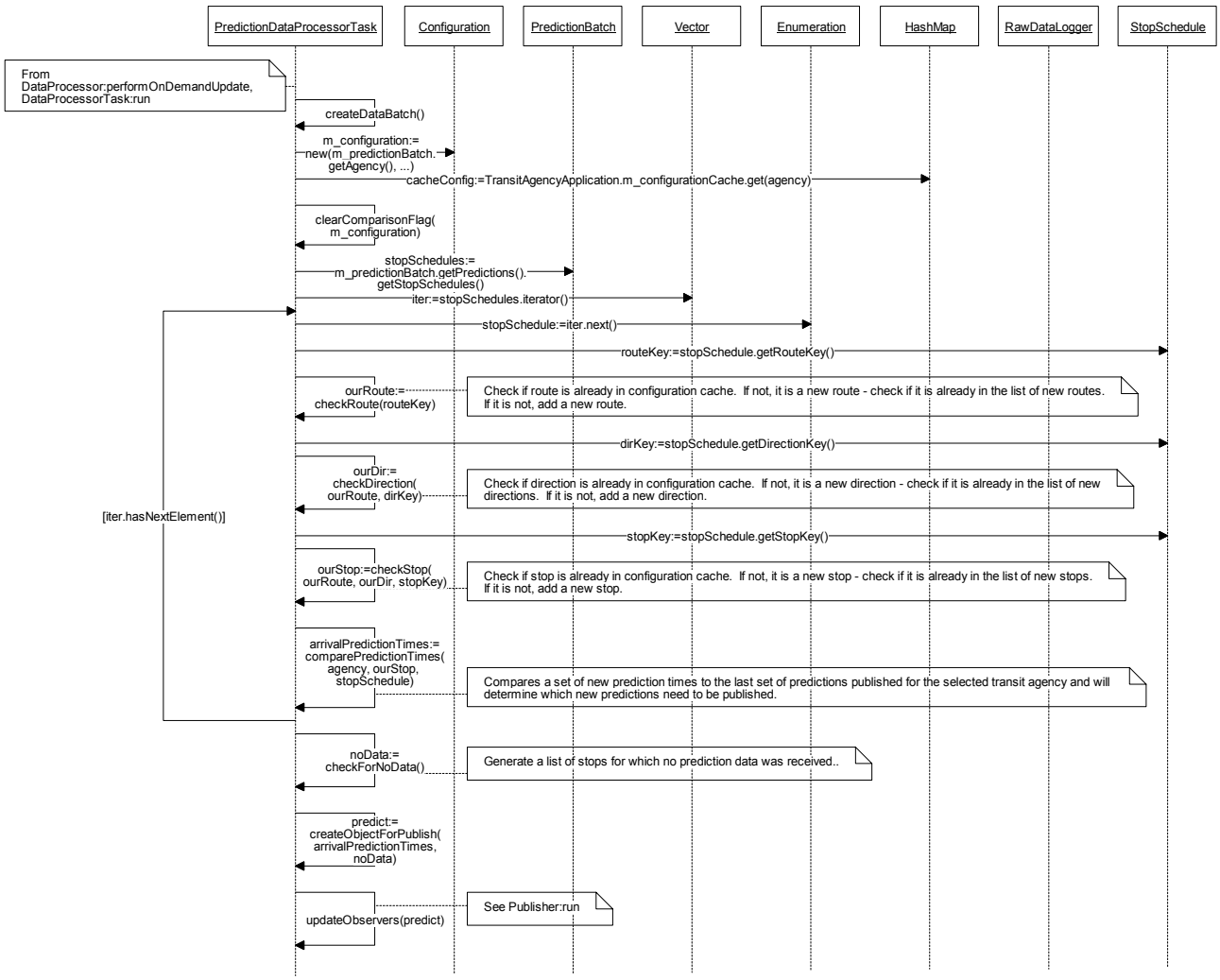

Figure 36. PredictionDataProcessorTask:processData (Sequence Diagram)

3.4.25. FTPDataRetriever:getArrivalStatusData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated arrival status data via an FTP server.

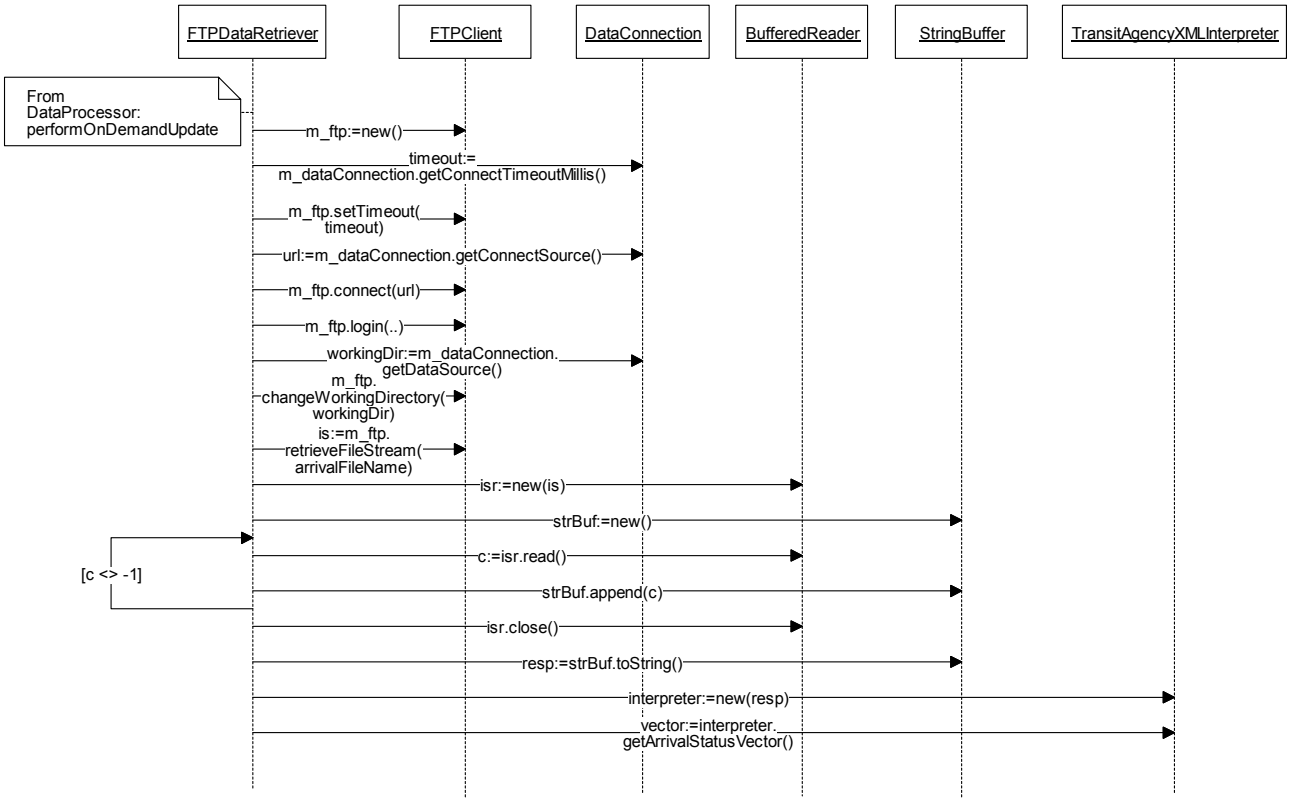

Figure 37. FTPDataRetriever:getArrivalStatusData (Sequence Diagram)

3.4.26. WSDataRetriever:getConfigurationData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated configuration data via agency web services.

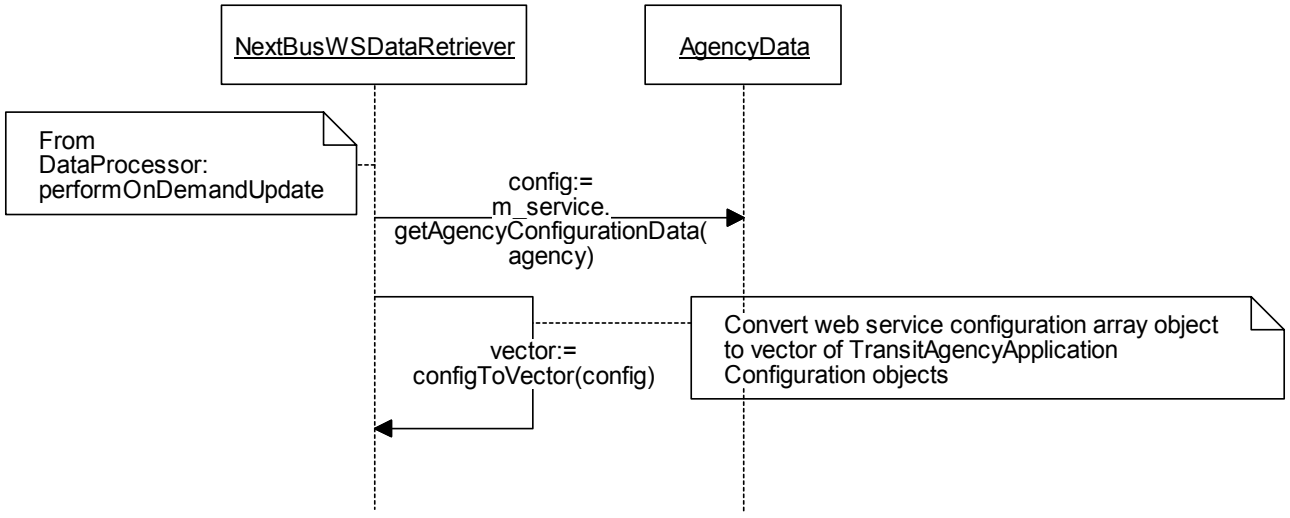

Figure 38. WSDataRetriever:getConfigurationData (Sequence Diagram)

3.4.27. JMSPubSubscribeDataRetriever:onMessage (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated configuration, prediction, or arrival status data via subscriptions to a Java Message Service (JMS) server.

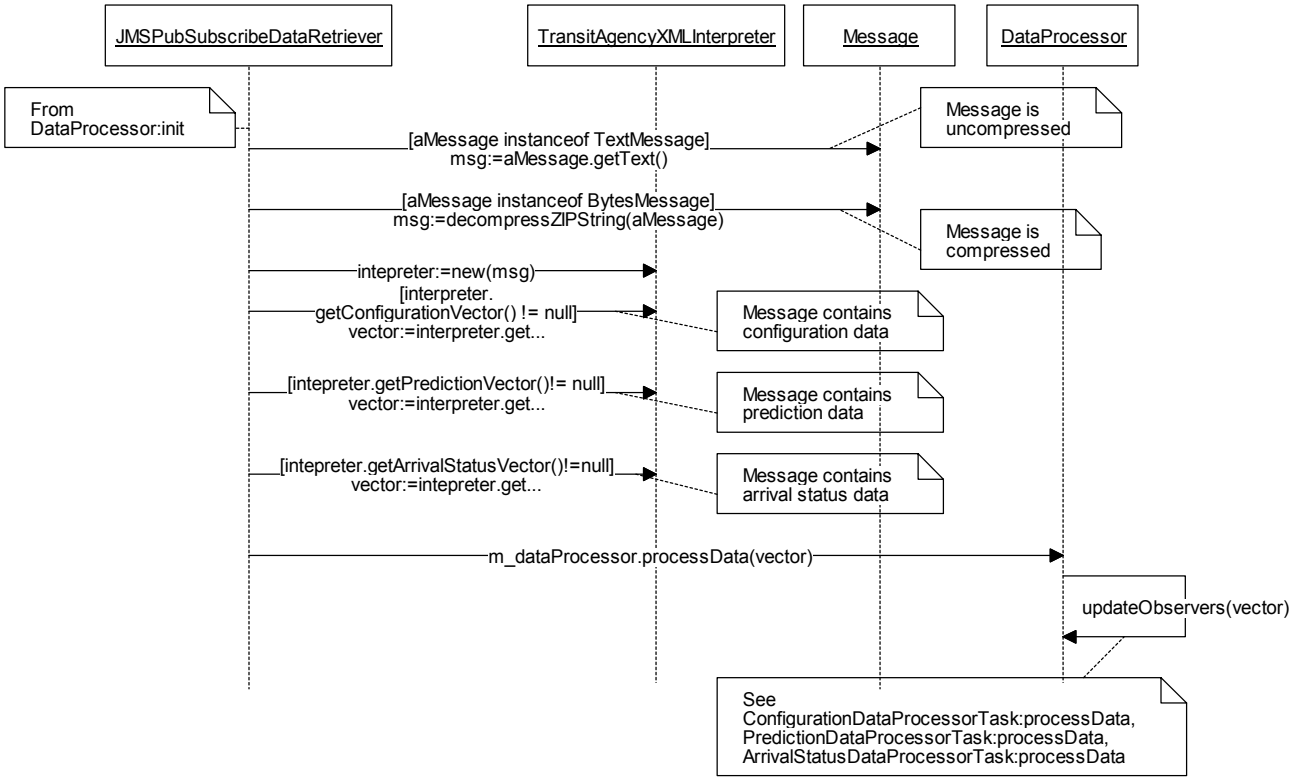

Figure 39. JMSPubSubscribeDataRetriever:onMessage (Sequence Diagram)

3.4.28. JMSRequestReplyDataRetriever:getArrivalStatusData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated arrival status data via the Java Message Service.

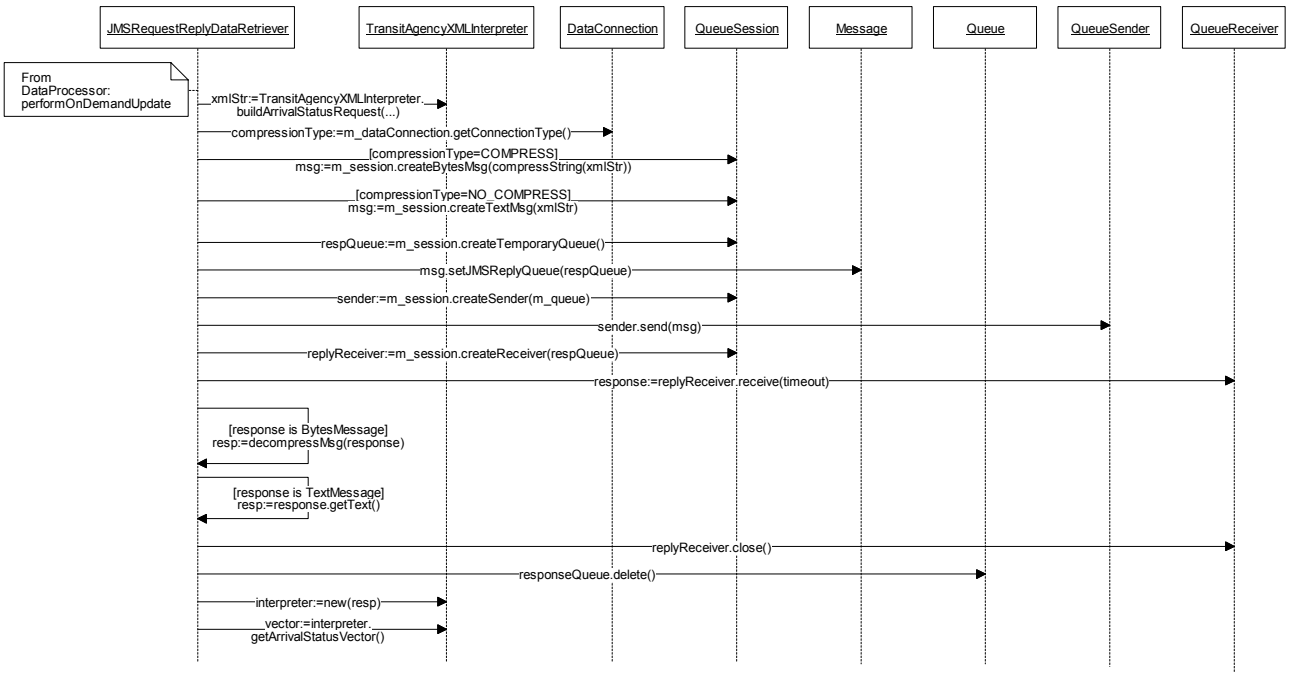

Figure 40. JMSRequestReplyDataRetriever:getArrivalStatusData (Sequence Diagram)

3.4.29. Publisher:run (Sequence Diagram)

This sequence diagram shows the steps necessary to receive prediction data to publish.

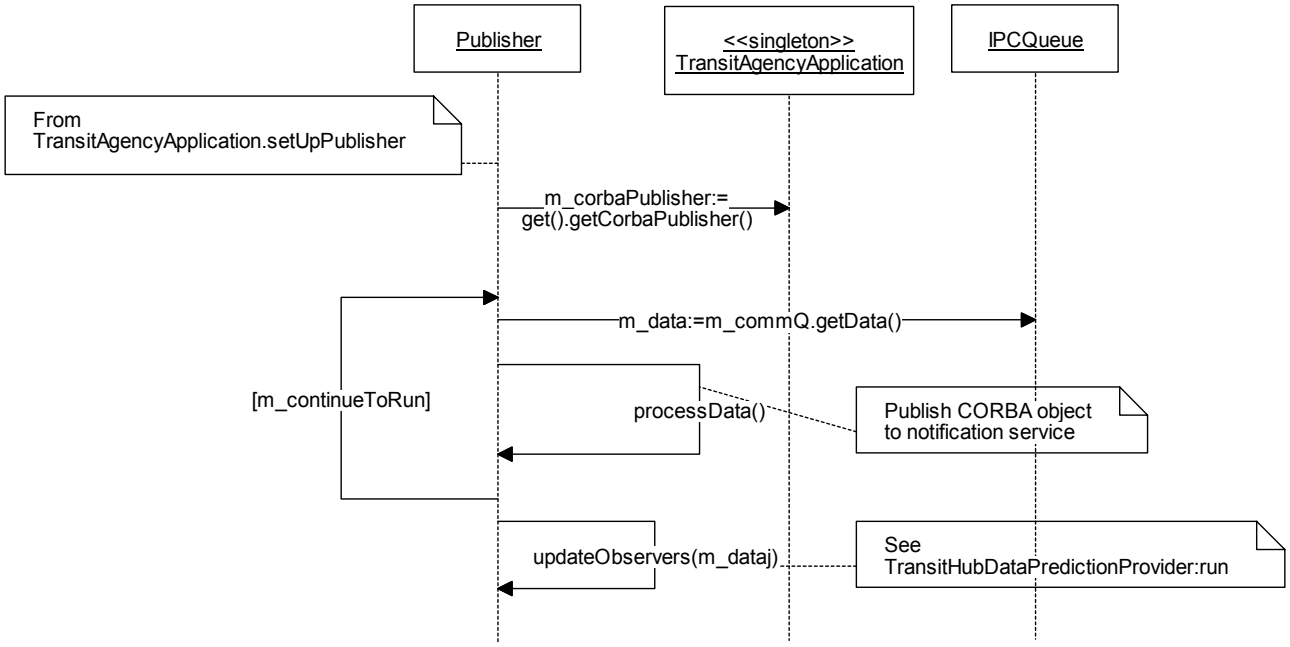

Figure 41. Publisher:run (Sequence Diagram)

3.4.30. Publisher:setUpPublisher (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of the prediction publisher task.

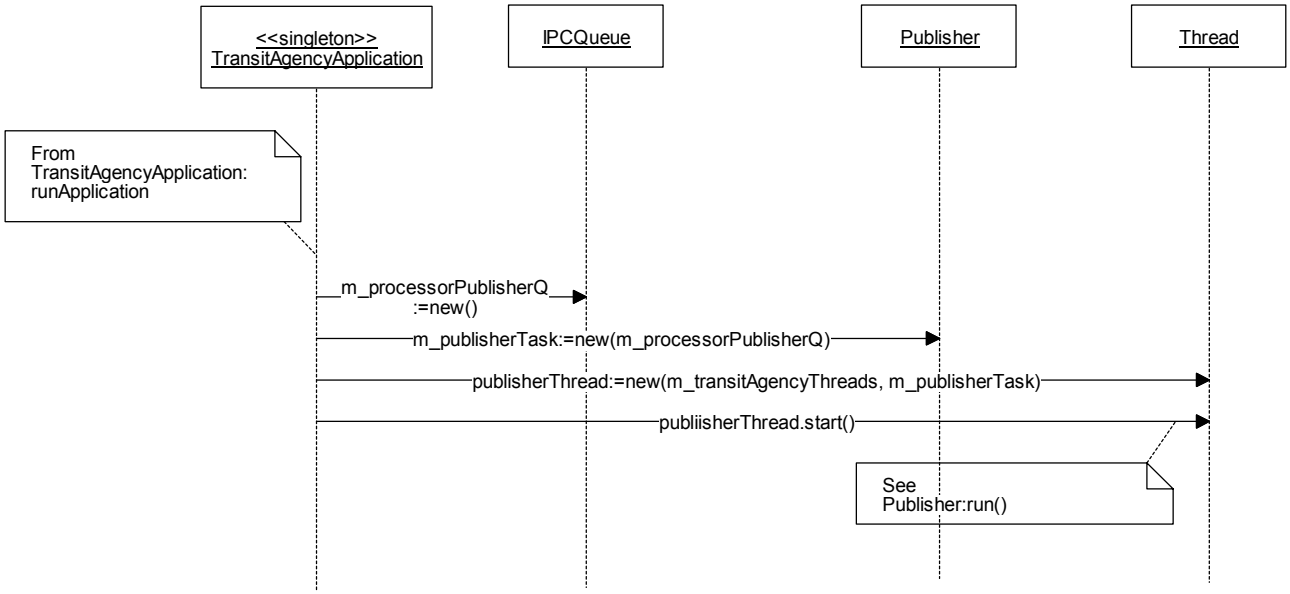

Figure 42. Publisher:setUpPublisher (Sequence Diagram)

3.4.31. TransitHubPredictionDataProvider:run (Sequence Diagram)

This sequence diagram shows the steps necessary to receive prediction data to save to the system database.

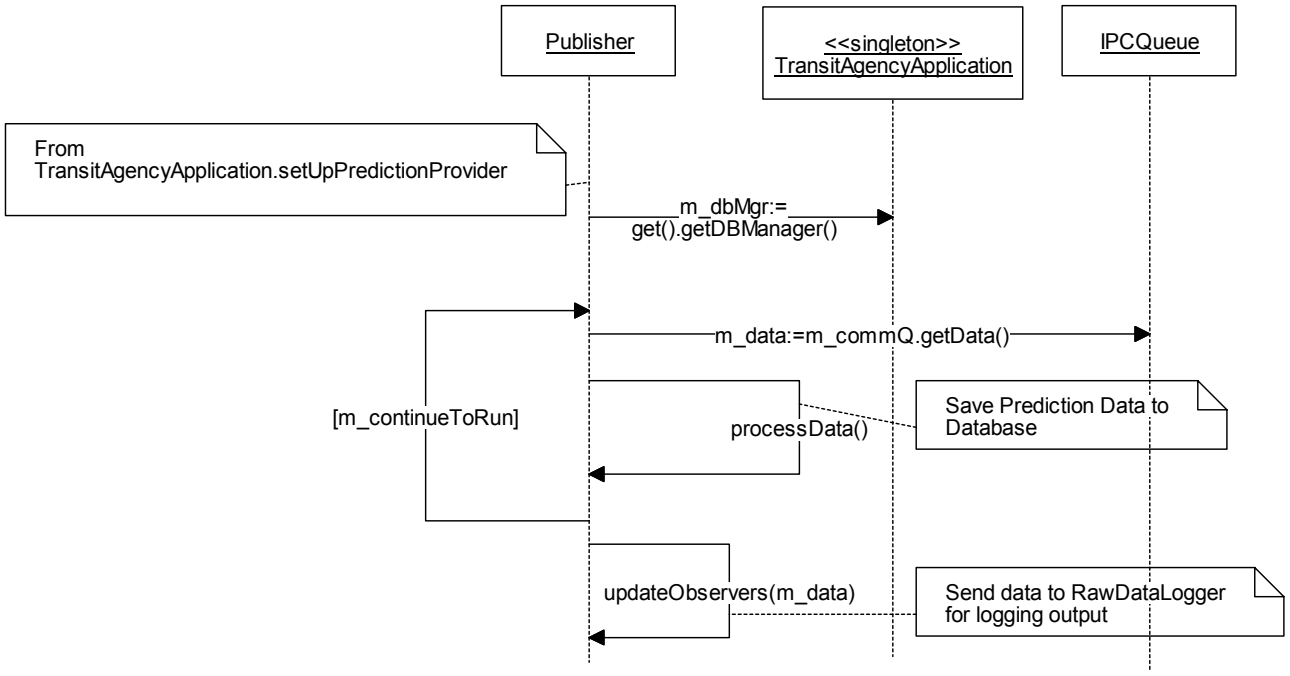

Figure 43. TransitHubPredictionDataProvider:run (Sequence Diagram)

3.4.32. TransitHubPredictionDataProvider:setUpPredictionProvider (Sequence Diagram)

This sequence diagram shows the steps necessary to initialize and start the main thread of the transit hub prediction data provider task.

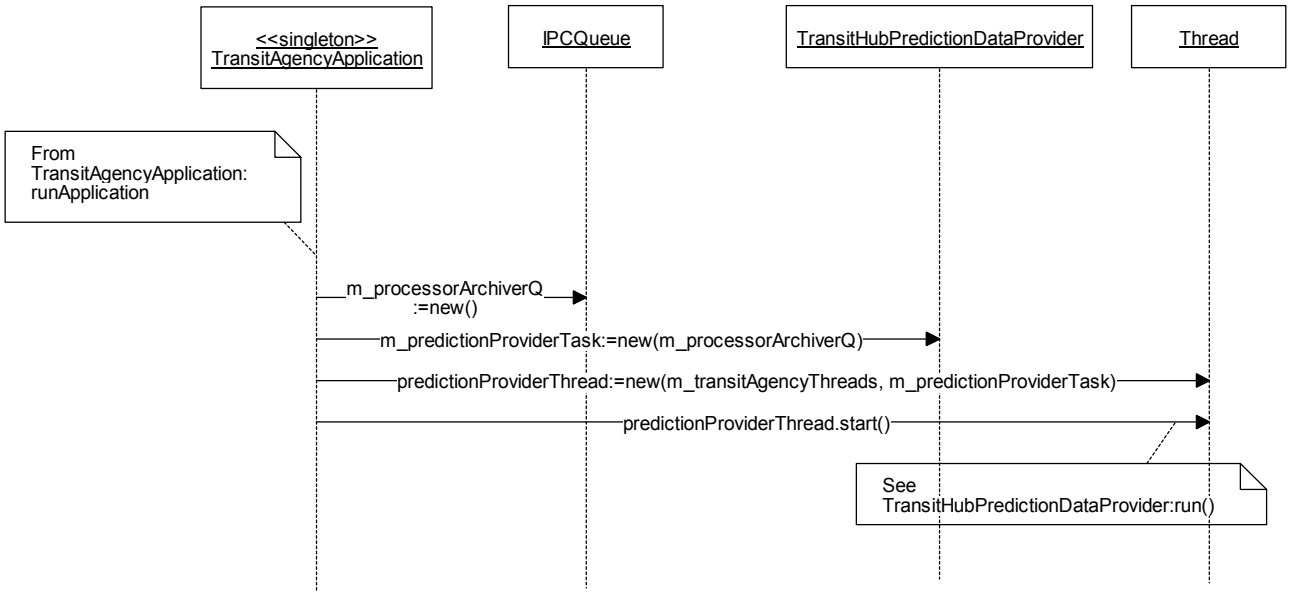

Figure 44. TransitHubPredictionDataProvider:setUpPredictionProvider (Sequence Diagram)

3.4.33. WSDataRetriever:getArrivalStatusData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated arrival status data via agency web services.

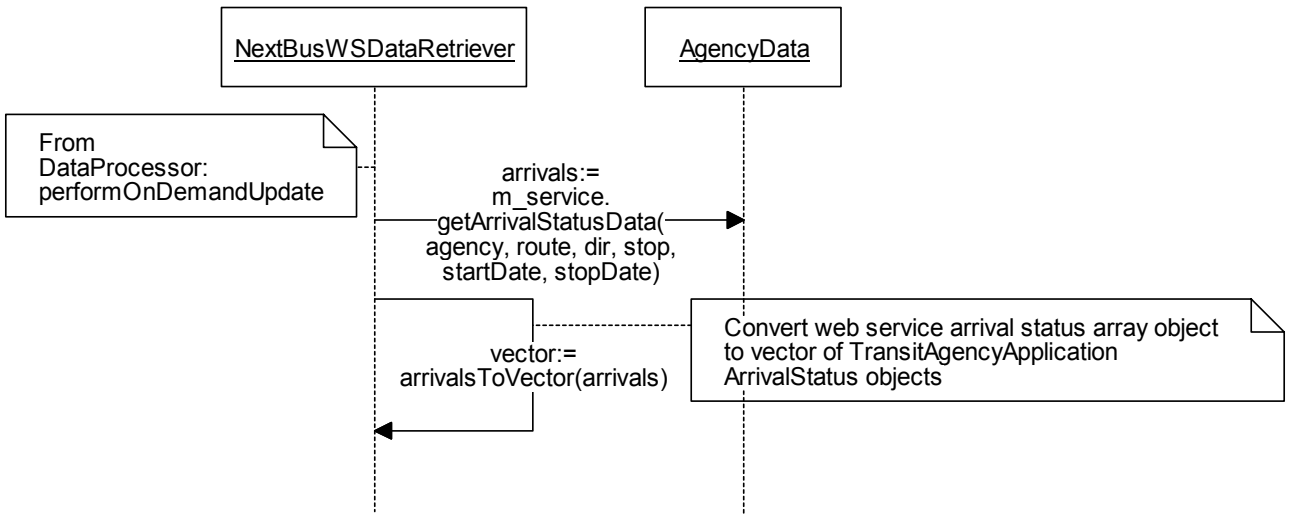

Figure 45. WSDataRetriever:getArrivalStatusData (Sequence Diagram)

3.4.34. WSDataRetriever:getPredictionData (Sequence Diagram)

This sequence diagram shows the steps necessary to retrieve updated prediction data via agency web services.

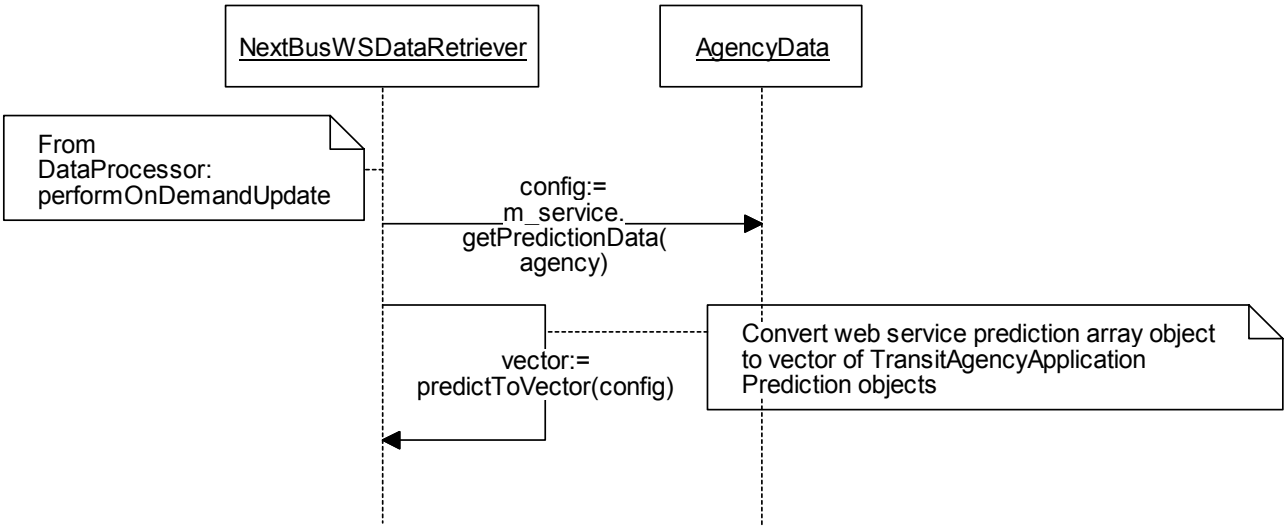

Figure 46. WSDataRetriever:getPredictionData (Sequence Diagram)

4 Packaging

4.1 Packaging (Component Diagram)

The package diagram shows the packages and dependences of the TransitAgencyApplication package and their dependencies. Only the direct dependencies of the main package that are not part of the core Java packages are shown.

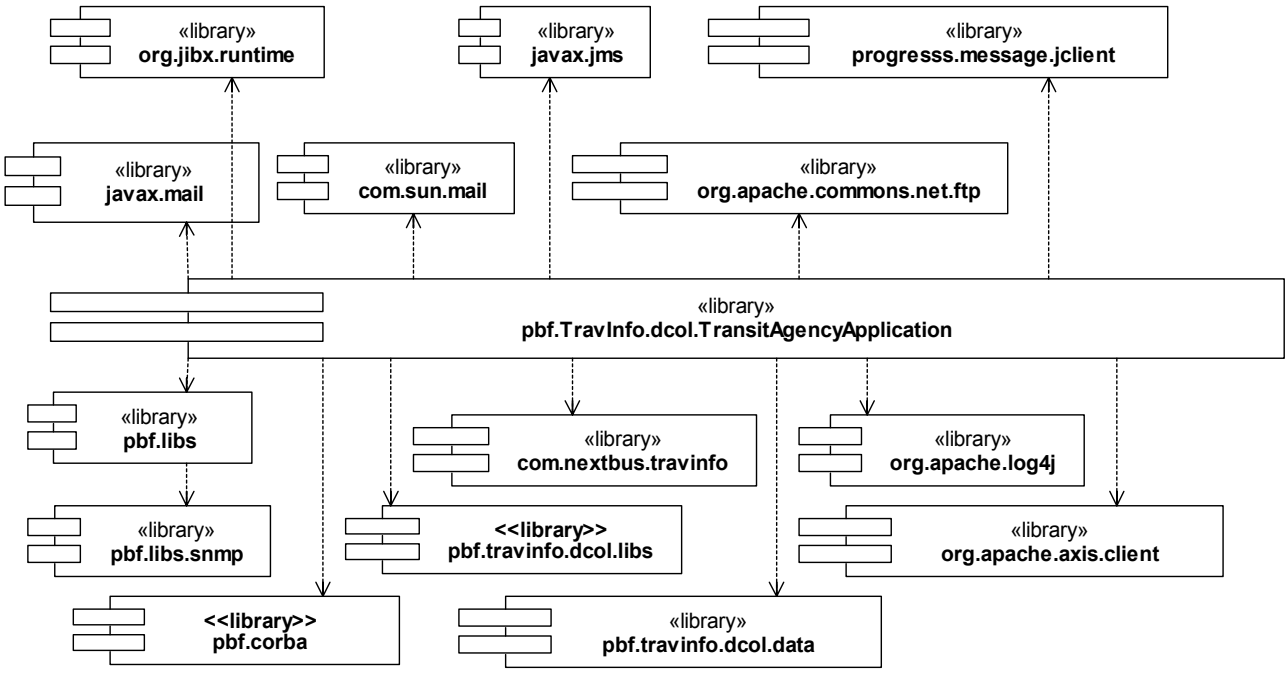

Figure 47. Packaging (Component Diagram)

5 Deployment

The deployment diagram shows where each of the system modules related to the TransitAgencyApplication will execute. The TransitAgencyApplication service typically, but not necessarily, will run on the same server as the CORBA Naming and Notification services. Upon startup, the system will read configuration data from the customer-supplied database server and register for notifications of new messages from the Java Message Service (JMS) that will execute on a server provided by the customer. The system may also periodically read data from agency web services, FTP servers, or the NextBus web service. Agencies may write agency configuration changes, predicted arrival times, and actual arrival times for each route to the JMS or may provide them via web service calls, or may provide the data via files on an FTP server. The system will write changes to the database and the CORBA Notification service. The second web server will contain the web page to be used by transit hubs to display arrival predictions. The page will read data from the database server.

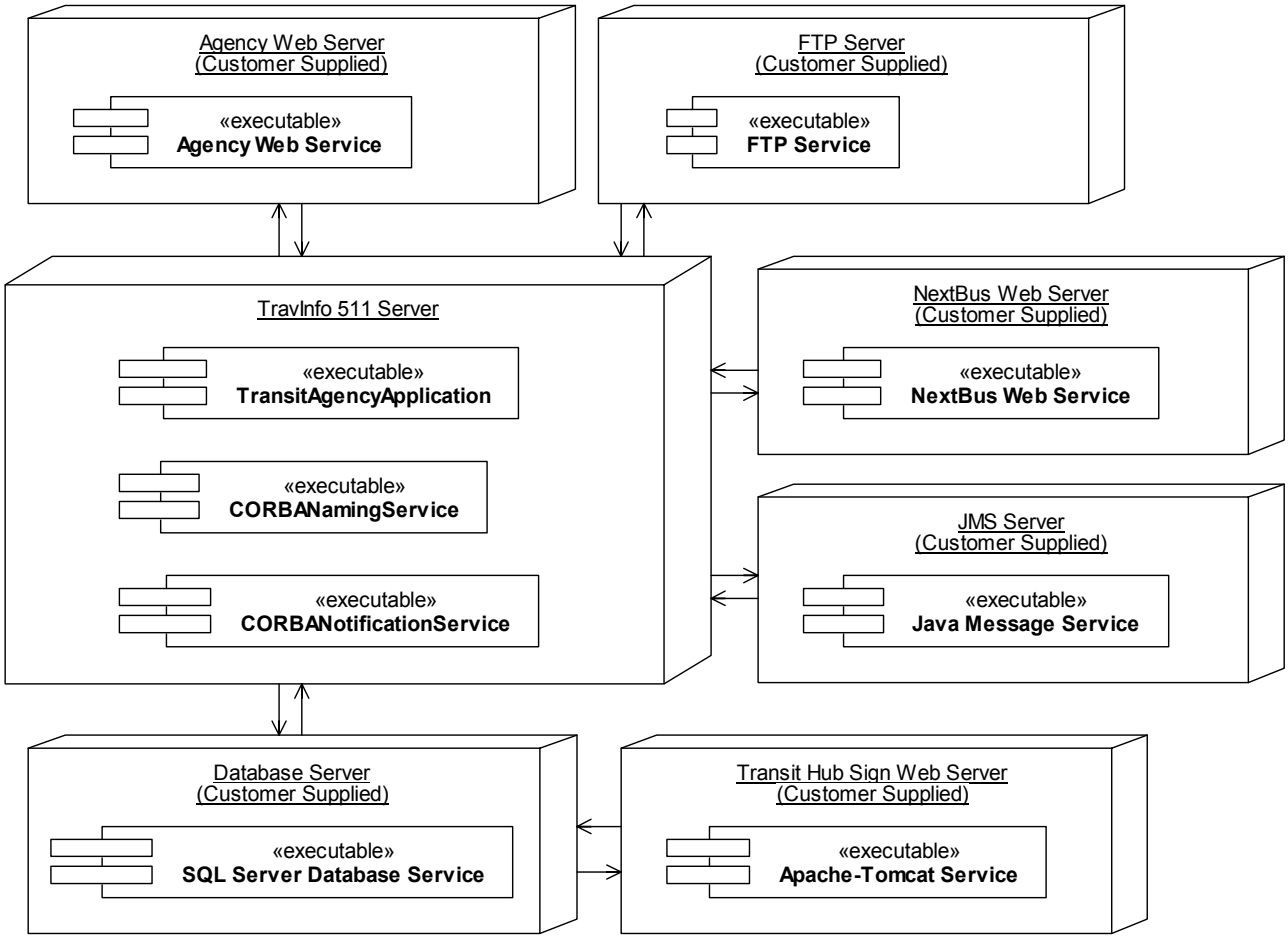

Figure 48. Deployment (Deployment Diagram)

Appendix A. TAA JMS System Data Specification

This section contains the Extensible Markup Format (XML) document type definitions (DTDs) for the messages exchanged between the transit agency Java Message Service (JMS) server, and the TAA system. Currently, the TAA system periodically retrieves data from websites devoted to each transit agency via the NextBus system. The NextBus system accesses these websites via web services to download XML data that contains both agency configuration and arrival prediction update information. The next version of the TAA will access this data either from NextBus web services, transit agency web services, and/or from a central JMS server. The TAA will continue to request periodic updates of the agency configuration data by issuing a configuration request message to the JMS server and receiving a configuration response message. To obtain arrival predictions from some agencies, the TAA will subscribe to changes to the arrival times via prediction updates sent from the JMS server.

Configuration Request Message

To obtain updated configuration information from the agencies, 511 issues a configuration request message to the transit agency's JMS client via the JMS server. There is no mandatory data within the request message. The format of the message that the transit agency will receive is documented in the XML DTD shown below:

Configuration Request:

```
<!-- Request sent for on-demand updates for agency
     configuration information. -->
<!DOCTYPE SystemInformationRequest [
<!ELEMENT SystemInformationRequest>
<!ATTLIST SystemInformationRequest
      Agency          CDATA  #Format: Comma separated list>
 ]>
```

Configuration Response Message

Upon issuance of a configuration request message, a configuration response message will be sent. The response message will contain either a list of errors or the requested configuration data. An error record will contain the following information (data element names are shown in parenthesis):

- The integer error code if applicable (**errorCode**)
- The text of the error (**errorText**)

Each configuration data record must contain the following:

- The name of the agency (**agency**)
- The version number of the configuration data (used to determine when changes have occurred since the last configuration update) (**version**)
- The date/time of the current snapshot of data (**TimeStamp**)
- The total number of stops (**numStops**)

- An optional description of the data – e.g. description of data quality (**msg**)
- The list of bus routes for the agency (**Route**)

  Each agency may have one or more bus routes. Each bus route record must contain the following information (data element names are shown in parenthesis):

- An identifier or alias that uniquely distinguishes each route from all others (**key**)
- The name of the route – usually a street name (**title**)
- The list of directions and/or destinations of the route (**Direction**)

  Each route may have one or more directions/destinations. Each direction/destination record must contain the following information (data element names are shown in parenthesis):

- An identifier or alias that uniquely distinguishes each direction from all others (**key**)
- The name of the direction – for example inbound or outbound (**title**)
- The list of stops in each direction/destination (**Stop**)

  Each direction may have one or more stops. Each stop record must contain the following information:

- An identifier or alias that uniquely distinguishes each stop from all others (**key**)
- The name of the stop (**title**)
- Optional street name for the stop (**onStreet**)
- Optional intersection name for the stop (**atStreet**)

  The hierarchical relationship between all of the aforementioned data elements is documented in the XML DTD shown below.

  Configuration Response:

```
<!-- Configuration information response received. -->
<!DOCTYPE ConfigurationDataMessage [
<!ELEMENT ConfigurationDataMessage (Error* | ConfigurationData*) >
<!ELEMENT Error >
<!ATTLIST Error
      errorCode         CDATA
      errorText         CDATA #REQUIRED >
<!ELEMENT ConfigurationData (Route+ ) >
<!ATTLIST ConfigurationData
agency                        CDATA #REQUIRED
                      version        CDATA #REQUIRED
                      numStops              CDATA #REQUIRED
TimeStamp                     CDATA #REQUIRED
#Format: YYYY/MM/DD HH:MM:SS -hhmm
                              #where -hhmm is the time offset
                             msg                     CDATA >
<!ELEMENT Route (Direction+) >
<!ATTLIST Route
key                           CDATA #REQUIRED
title                         CDATA #REQUIRED >
<!ELEMENT Direction (Stop+ ) >
<!ATTLIST Direction
key                           CDATA #REQUIRED
title                         CDATA #REQUIRED
```

```
pattern     >

<!ATTLIST Stop
key                    CDATA #REQUIRED
title          CDATA #REQUIRED
onStreet   CDATA
atStreet   CDATA
position       (START | STOP | EN-ROUTE) #REQUIRED >]>
```

Prediction Data Message

To obtain departure/arrival time predictions, the 511 system (as well as any other applications) may issue a request, or subscribe to updates from the JMS server. The JMS server will then send a prediction data message to all subscribers whenever departure/arrival time predictions change. Within this message, a prediction data record must be provided for each agency. The response message will contain either a list of errors or the requested prediction data. An error record will contain the following information (data element names are shown in parenthesis):

- The integer error code if applicable (**errorCode**)
- The text of the error (**errorText**)

Each prediction data record must contain the following information (data element names are shown in parenthesis):

- The name of the agency (**agency**)
- The version number of the configuration data (used to determine when changes have occurred since the last configuration update) (**version**)
- The date/time of the current snapshot of data (**TimeStamp**)
- The total number of stops (**numStops**)
- The list of departure/arrival time predictions – one for each route (**Predictions**)

Each agency may have one or more bus routes. Each bus route may possess multiple directions/destinations and may have several stops. Thus, each route must contain a list of the departure/arrival time predictions for each stop in all directions/destinations. Each prediction record must contain the following information (data element names are shown in parenthesis):

- The identifier or alias that uniquely distinguishes each route (**route**)
- The name of the direction/destination (**dir**)
- The identifier or alias that uniquely distinguishes each stop (**stop**)
- An optional message containing information concerning the route (**msg**)
- A list of predicted arrival and departure times for each stop (**Ptimes**)
- The type of prediction; either arrival or departure (**PredictionType**)
- Each stop may have one or more departure/arrival time predictions. Each departure/arrival time record must contain the time at which a vehicle is predicted to arrive at or depart from the stop (**PredictionTime**).

The hierarchical relationship between all of the aforementioned data elements is documented in the XML DTD shown below.

Prediction Response:

```
<!-- Prediction information received. -->
<!DOCTYPE PredictionDataMessage [
<!ELEMENT PredictionDataMessage (Error* | PredictionData*) >
<!ELEMENT Error >
<!ATTLIST Error
      errorCode         CDATA
      errorText         CDATA #REQUIRED >
<!ELEMENT PredictionData (Predictions+) >
<!ATTLIST PredictionData
       agency         CDATA #REQUIRED
       version        CDATA #REQUIRED
       numStops               CDATA #REQUIRED
       TimeStamp              CDATA #REQUIRED
                                    #Format: YYYY/MM/DD HH:MM:SS -hhmm
                                    #where -hhmm is the time offset >
<!ELEMENT Predictions (Ptimes*) >
<!ATTLIST Predictions
       route             CDATA #REQUIRED
       dir            CDATA #REQUIRED
      pattern
       stop                   CDATA #REQUIRED
       msg                    CDATA >
<!ATTLIST Ptimes

PredictionType     (D | A ) #REQUIRED
                            #Format: D - Departure Time
                            #        A - Arrival Time
       PredictionTime CDATA #REQUIRED
#Format: YYYY/MM/DD HH:MM:SS -hhmm
      #where -hhmm is the time offset > ]>
```

Arrived Status Request Message

To obtain updated actual arrival times of the vehicles from the agencies, 511 issues an arrived status request message to the JMS server, which will get forwarded to the specific agencies. Each record optionally may contain the following information (data element names are shown in parenthesis):

- The name of the agency (**agency**)
- The identifier or alias that uniquely distinguishes each route (**route**)
- The identifier of the direction, for example, 'I' or 'O' for inbound and outbound respectively (**dir**)
- The identifier of the stop (**stop)**
- The start time for which data is requested (**StartTime)**
- The stop time for which data is requested (**StopTime)**
- The identifier of the vehicle (**VehicleId)**

All of the attributes are optional. If an attribute is left blank or not included, all data for that attribute should be retrieved (e.g. if attribute 'route' is null or blank, data for all routes should be retrieved).

The starttime and stoptimes fields are the time range for which status is being requested. Typically, this message should be requested once per day, and should be for the full previous day. For example, the Arrived Status Request Message could be configured to be sent once per day at 2AM. If it was sent on February 2, 2006, the start date would be 2/1/2006 12:00:00 AM, and end date would be 2/1/2006 11:59:59 PM

The format of the message is documented in the XML DTD shown below:

Arrival Status Request:

```
<!-- Request sent to the JMS server for updates for actual vehicle arrival and departure
times. -->
<!DOCTYPE ArrivalStatusRequest
<!ELEMENT ArrivalStatusRequest>
<!ATTLIST ArrivalStatusRequest
       agency            CDATA #Format: Comma separated list
      route                  CDATA #Format: Comma separated list
      dir                    CDATA #Format: Comma separated list
      stop                   CDATA #Format: Comma separated list
        startTime            CDATA #Format: YYYY/MM/DD HH:MM:SS
        stopTime             CDATA #Format: YYYY/MM/DD HH:MM:SS >
```

Arrived Status Response Message

In response to arrived status request messages, the transit agencies will send an arrived status response message. The response message will contain either a list of errors or the requested arrival status data. An error record will contain the following information (data element names are shown in parenthesis):

- The integer error code if applicable (**errorCode**)
- The text of the error (**errorText**)

Each arrived status response record must contain the following information (data element names are shown in parenthesis):

- The name of the agency (**agency**)
- The date/time of the current snapshot of data (**TimeStamp**)
- The identifier or alias that uniquely distinguishes each route (**route**)
- The name of the direction/destination (**dir**)
- The identifier or alias that uniquely distinguishes each stop (**stop**)
- The unique identifier of the vehicle (**VehicleId)**
- The time at which the bus arrived at the stop (**ArrivalTime**)

The relationship between all of the aforementioned data elements is documented in the XML DTD shown below.

Arrival Status Response:

```
<!-- Arrival Status information received. -->
<!DOCTYPE ArrivalStatusDataMessage [
<!ELEMENT ArrivalStatusDataMessage (Error* | ArrivalStatusDataMessage*) >
<!ELEMENT Error >
<!ATTLIST Error
      errorCode         CDATA
      errorText         CDATA #REQUIRED >
<!ELEMENT ArrivalStatusData >
<!ATTLIST ArrivalStatusData
       agency         CDATA #REQUIRED
       TimeStamp              CDATA #REQUIRED
                                    #Format: YYYY/MM/DD HH:MM:SS
       route            CDATA #REQUIRED
       dir            CDATA #REQUIRED
```

```
        stop                    CDATA #REQUIRED
        VehicleId               CDATA #REQUIRED
        ArrivalTime             CDATA #REQUIRED
                                #Format: YYYY/MM/DD HH:MM:SS >
]>
```

Appendix B. TAA System Properties File

The TAA system retrieves user-configurable parameters from a properties file. The parameters that are pertinent to the TAA system are defined in the table below.

Properties File Parameters

Parameter Name	Description	Format
PluggablePublishingApp.className	Name of this service	Alpha-numeric
NotificationService.PersistentConnections	Use persistent connections to preserve client functionality during notification service shutdowns	Either true or false (default false)
NotificationService.ProxyIDPersistenceFilename	Filename used for persistent connections	NotificationService PersistentProxyIDs. props
NotificationService.MaxEventsPerConsumer	Max. number of queued events per consumer	Numeric (default 10000) dependent
NamingService.URL	location of root NamingContext object served by the CORBA Naming Service	Alpha-numeric -
NotificationService.URL	location of EventChannelFactory object served by the CORBA Notification Service	Alpha-numeric
jdbc.drivers	JDBC driver class name	Alpha-numeric
TransitAgency.RawDataLoggerName	file name used to log the raw data received	Either [drive:][path]\ [filename] or [\\server\\][path]\ [filename]
TransitAgency.LogInputs	Flag indicating whether raw data received should be logged	True or False (default true)
TransitAgency.LogInputsName	Name of file for logging input data	Either [drive:][path]\ [filename] or [\\server\\][path]\ [filename]
TransitAgency.LogOutputs	Flag indicating whether output data should be logged	True or False (default true)
TransitAgency.LogOutputsName	File name for logging output data	Either [drive:][path]\ [filename] or [\\server\\][path]\ [filename]
TransitAgency.LogRetain	The number of days to keep data log archive files before deleting them	Integer – values < 0 indicate log files are never deleted (default 7)
TransitAgency.DBConnectURL	DB Connection String for Database services specified in milliseconds	Alpha-numeric - exact format is database dependent
TransitAgency.DBUserName	User name to be used to connect	Alpha-numeric

	to the database	
TransitAgency.DBPassword	Password to be used to connect to the database	Alpha-numeric
TransitAgency.BlockCORBAPublish	Indicates whether CORBA publishing will occur during processing	True or false (default false)
TransitAgency.ConfigurationProcessorTime	Time of day (HH:MM) to process configuration change data	HH:MM (default 03:00)
TransitAgency.ConfigurationRptTime	Time of day (HH:MM) to generate the Configuration Report	HH:MM (default 06:00)
TransitAgency.ConfigurationRptFrequency	Indicates how frequently to run the Configuration Report	D=Daily, W=Weekly, M=Monthly (default W)
TransitAgencyApplication.MaxLogFileSize	Maximum size of a log file version in bytes	Integer > 0 (default 10000000 bytes)
TransitAgencyApplication.MaxLogFileVersionNumber	Maximum number of log files to keep each day	Integer > 0 (default 100)
TransitAgencyApplication.RunConfigurationProcessor	Indicates whether to automatically retrieve configuration updates	True or False (default False)
TransitAgencyApplication.SavePredictionsToDB	Indicates whether to save prediction information to the system database	0 – Save only transit hub data, 1 – Save no data, 2 – Save all data (default 0)
TransitAgencyApplication.ChangeThresholdForPublish	The amount a new prediction must differ from the last one(s) sent before it will be published again (in minutes).	Integer >= 0 (default 1)
TransitAgencyApplication.TimeWindow	Duration (in minutes) beyond which reported predictions will be discarded.	Integer > 0 (default 30)
TransitAgencyApplication.RecentMessageStatsIntervalSec	Interval in seconds of recent message stats interval.	Integer > 0 (default 60)
TransitAgencyApplication.ThreadMonitorPeriodSec	The interval, in seconds, for when TransitAgencyApplication DI checks for threads that have stopped running.	Integer > 0 (default 60)
TransitAgencyApplication.MaxNumOfPredictionsPerStop	The maximum number of predictions to save per stop before discarding.	Integer > 0 (default 3)
TransitAgencyApplication.PredictionInterval	Time interval after which emails are sent if particular stops do not receive prediction data from TransitAgencyApplication (minutes).	Integer > 0 (default 60 min)
TransitAgencyApplication.UseArrivalStatusDatabaseStops	Indicates whether to use the list defined in the database for requesting arrival statuses.	True or False (default True)
TransitAgencyApplication.UpdateInMemoryConfigurations	Indicates whether to update in-memory copy of agency configurations.	True or False (default False)

TransitAgencyApplication.OperatorEmails	Comma-separated list of emails of administrators to be notified when errors occur	Alpha-numeric
TransitAgencyApplication.Mail.From	Sender of the emails to administrators (not used by some email servers).	Alpha-numeric
TransitAgencyApplication.Mail.Host	Email server host name for outgoing messages.	Alpha-numeric
TransitAgencyApplication.Mail.UserName	Email server host login username.	Alpha-numeric
TransitAgencyApplication.Mail.Password	Email server host login password.	Alpha-numeric
TransitAgencyApplication.SendMissingStopsEmail	Indicator for sending missing stops emails.	True or False (default False)

For explanations of the property file parameters that are used by the JacORB ORB, consult the JacORB documentation.

Appendix C. TAA Web Service Definition Language (WSDL) File

The version of the TAA web service WSDL file used for this design is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
File Name     :  AgencyDataService.wsdl
Prepared by   :  Telvent/Farradyne

   Copyright 2007, Telvent/Farradyne Inc.
                   All Rights Reserved

This file is property of Metropolitan Transportation Commission (MTC).  Any unauthorized use or
duplication of this file, or removal of the header, constitutes
theft of intellectual property.

Description   :  This file contains the WSDL for agency data for the 511 project.  Descriptions of
                 the three (3) public web methods follows:

   getAgencyConfigurationData - Request for agency configuration data.
        parameter in0 - Optional comma separated list of agency names for which configuration data
will be returned.
        returns - ConfigurationDataMessage containing either an array of ConfigurationData objects or
an array
                  of Error objects.

   getPredictionData - Request for vehicle arrival prediction updates.
        parameter in0 - Optional comma separated list of agency names for which prediction data will
be returned.
        returns - PredictionDataMessage containing either an array of PredictionData objects or an
array
                  of Error objects.

   getArrivalStatusData - Request for actual vehicle arrival/departure updates.
         parameter in0 - Optional comma separated list of agency names for which arrival status data
will be returned.
         parameter in1 - Optional comma separated list of route codes for which arrival status data
will be returned.
         parameter in2 - Optional comma separated list of direction/destination codes for which
arrival status data will be returned.
         parameter in3 - Optional comma separated list of stop codes for which arrival status data
will be returned.
         parameter in4 - Start time. If specified, only arrival times after the start time will be
returned.
         parameter in5 - Stop time. If specified, only arrival times before the start time will be
returned.
        returns - ArrivalStatusDataMessage containing either an array of ArrivalStatusData objects or
an array
                  of Error objects.

History:

-->
<wsdl:definitions                                              targetNamespace="urn:AgencyDataService"
xmlns:apachesoap="http://xml.apache.org/xml-soap"                   xmlns:impl="urn:AgencyDataService"
xmlns:intf="urn:AgencyDataService"          xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://DefaultNamespace"                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://DefaultNamespace" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="urn:AgencyDataService"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrivalStatusData">
    <sequence>
     <element name="agency" nillable="true" type="soapenc:string"/>
     <element name="arrivalTime" nillable="true" type="xsd:dateTime"/>
```

```
  <element name="dir" nillable="true" type="soapenc:string"/>
  <element name="route" nillable="true" type="soapenc:string"/>
  <element name="stop" nillable="true" type="soapenc:string"/>
  <element name="vehicleId" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="Error">
 <sequence>
  <element name="errorCode" type="xsd:int"/>
  <element name="errorText" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="ArrivalStatusDataMessage">
 <sequence>
  <element name="arrivalStatus" nillable="true" type="impl:ArrayOf_tns1_ArrivalStatusData"/>
  <element name="errors" nillable="true" type="impl:ArrayOf_tns1_Error"/>
 </sequence>
</complexType>
<complexType name="Stop">
 <sequence>
  <element name="key" nillable="true" type="soapenc:string"/>
  <element name="position" nillable="true" type="soapenc:string"/>
  <element name="title" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="Direction">
 <sequence>
  <element name="key" nillable="true" type="soapenc:string"/>
  <element name="stop" nillable="true" type="impl:ArrayOf_tns1_Stop"/>
  <element name="title" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="Route">
 <sequence>
  <element name="direction" nillable="true" type="impl:ArrayOf_tns1_Direction"/>
  <element name="key" nillable="true" type="soapenc:string"/>
  <element name="title" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="ConfigurationData">
 <sequence>
  <element name="agency" nillable="true" type="soapenc:string"/>
  <element name="errors" nillable="true" type="impl:ArrayOf_tns1_Error"/>
  <element name="msg" nillable="true" type="soapenc:string"/>
  <element name="numStops" type="xsd:int"/>
  <element name="route" nillable="true" type="impl:ArrayOf_tns1_Route"/>
  <element name="timeStamp" nillable="true" type="xsd:dateTime"/>
  <element name="version" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="ConfigurationDataMessage">
 <sequence>
  <element name="configurations" nillable="true" type="impl:ArrayOf_tns1_ConfigurationData"/>
  <element name="errors" nillable="true" type="impl:ArrayOf_tns1_Error"/>
 </sequence>
</complexType>
<complexType name="Ptimes">
 <sequence>
  <element name="predictionTime" nillable="true" type="xsd:dateTime"/>
  <element name="predictionType" nillable="true" type="soapenc:string"/>
  <element name="vehicleId" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="Predictions">
 <sequence>
  <element name="dir" nillable="true" type="soapenc:string"/>
  <element name="msg" nillable="true" type="soapenc:string"/>
  <element name="ptimes" nillable="true" type="impl:ArrayOf_tns1_Ptimes"/>
  <element name="route" nillable="true" type="soapenc:string"/>
  <element name="stop" nillable="true" type="soapenc:string"/>
 </sequence>
</complexType>
<complexType name="PredictionData">
 <sequence>
  <element name="agency" nillable="true" type="soapenc:string"/>
  <element name="errors" nillable="true" type="impl:ArrayOf_tns1_Error"/>
```

```
    <element name="numStops" type="xsd:int"/>
    <element name="predictions" nillable="true" type="impl:ArrayOf_tns1_Predictions"/>
    <element name="timeStamp" nillable="true" type="xsd:dateTime"/>
    <element name="version" nillable="true" type="soapenc:string"/>
   </sequence>
  </complexType>
  <complexType name="PredictionDataMessage">
   <sequence>
    <element name="errors" nillable="true" type="impl:ArrayOf_tns1_Error"/>
    <element name="predictions" nillable="true" type="impl:ArrayOf_tns1_PredictionData"/>
   </sequence>
  </complexType>
 </schema>
 <schema targetNamespace="urn:AgencyDataService" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://DefaultNamespace"/>
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
  <complexType name="ArrayOf_tns1_ArrivalStatusData">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:ArrivalStatusData[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Error">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Error[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Stop">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Stop[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Direction">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Direction[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Route">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Route[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_ConfigurationData">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:ConfigurationData[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Ptimes">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Ptimes[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_Predictions">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:Predictions[]"/>
    </restriction>
   </complexContent>
  </complexType>
  <complexType name="ArrayOf_tns1_PredictionData">
   <complexContent>
    <restriction base="soapenc:Array">
     <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:PredictionData[]"/>
```

```
     </restriction>
    </complexContent>
   </complexType>
  </schema>
 </wsdl:types>
   <wsdl:message name="getPredictionDataResponse">
      <wsdl:part name="getPredictionDataReturn" type="tns1:PredictionDataMessage"/>
   </wsdl:message>

   <wsdl:message name="getArrivalStatusDataResponse">
      <wsdl:part name="getArrivalStatusDataReturn" type="tns1:ArrivalStatusDataMessage"/>
   </wsdl:message>
   <wsdl:message name="getAgencyConfigurationDataRequest">
      <wsdl:part name="in0" type="soapenc:string"/>
   </wsdl:message>
   <wsdl:message name="getAgencyConfigurationDataResponse">
      <wsdl:part name="getAgencyConfigurationDataReturn" type="tns1:ConfigurationDataMessage"/>
   </wsdl:message>
   <wsdl:message name="getPredictionDataRequest">
      <wsdl:part name="in0" type="soapenc:string"/>
   </wsdl:message>
   <wsdl:message name="getArrivalStatusDataRequest">
      <wsdl:part name="in0" type="soapenc:string"/>
      <wsdl:part name="in1" type="soapenc:string"/>
      <wsdl:part name="in2" type="soapenc:string"/>
      <wsdl:part name="in3" type="soapenc:string"/>
      <wsdl:part name="in4" type="xsd:dateTime"/>
      <wsdl:part name="in5" type="xsd:dateTime"/>
   </wsdl:message>
   <wsdl:portType name="AgencyData">
      <wsdl:operation name="getArrivalStatusData" parameterOrder="in0 in1 in2 in3 in4 in5">
         <wsdl:input message="impl:getArrivalStatusDataRequest" name="getArrivalStatusDataRequest"/>
         <wsdl:output                                    message="impl:getArrivalStatusDataResponse"
name="getArrivalStatusDataResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getAgencyConfigurationData" parameterOrder="in0">
         <wsdl:input                                  message="impl:getAgencyConfigurationDataRequest"
name="getAgencyConfigurationDataRequest"/>
         <wsdl:output                                message="impl:getAgencyConfigurationDataResponse"
name="getAgencyConfigurationDataResponse"/>
      </wsdl:operation>
      <wsdl:operation name="getPredictionData" parameterOrder="in0">
         <wsdl:input message="impl:getPredictionDataRequest" name="getPredictionDataRequest"/>
         <wsdl:output message="impl:getPredictionDataResponse" name="getPredictionDataResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="AgencyDataServiceSoapBinding" type="impl:AgencyData">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getArrivalStatusData">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getArrivalStatusDataRequest">
            <wsdlsoap:body                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getArrivalStatusDataResponse">
            <wsdlsoap:body                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getAgencyConfigurationData">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getAgencyConfigurationDataRequest">
            <wsdlsoap:body                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getAgencyConfigurationDataResponse">
            <wsdlsoap:body                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="getPredictionData">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getPredictionDataRequest">
            <wsdlsoap:body                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:input>
```

```
         <wsdl:output name="getPredictionDataResponse">
            <wsdlsoap:body                  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:AgencyDataService" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="AgencyDataService">
      <wsdl:port binding="impl:AgencyDataServiceSoapBinding" name="AgencyDataService">
         <wsdlsoap:address location="http://localhost:8080/axis/services/AgencyDataService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Appendix D. Interface Definition Language File

```
/*

File Name    :  DataCollection.idl

Prepared By  :  PB Farradyne

+------+   Copyright 2003, PB Farradyne
| PBF  |                   All Rights Reserved
+------+-----------------------------------------------------------
This file is property of PB Farradyne.  Any unauthorized use or
duplication of this file, or removal of the header, constitutes
theft of intellectual property.

Description  : This file contains the DataCollection IDL module
               for the Data Collection subsystem of the TravInfo project.

History      :

$Log$

*/
#ifndef _DCOL_IDL_
#define _DCOL_IDL_


/**
* This module contains definitions for interfacing with the TravInfo
* Data Collection subsystem.
*/
module DataCollection
{
   /**
   * The name of the CORBA event channel used to push LinkConditionEvent events when
   * the link condition changes.
   */
   const string EVENT_CHANNEL_LINK_CONDITION = "LinkConditionEventChannel";

   /**
   * The name of the CORBA event channel used to push LinkConditionEvent events when a
   * new historical link dataset becomes active.
   */
   const string EVENT_CHANNEL_HISTORICAL_LINK_CONDITION = "HistoricalLinkConditionEventChannel";

   /**
   * The name of the CORBA event channel used to push SegmentConditionEvent events.
   */
   const string EVENT_CHANNEL_SEGMENT_CONDITION = "SegmentConditionEventChannel";

   /**
   * The name of the CORBA event channel used to push AveragedSegmentConditionEvent events.
   */
   const string EVENT_CHANNEL_AVERAGED_SEGMENT_CONDITION = "AveragedSegmentConditionEventChannel";

   /**
   * The name of the CORBA event channel used to push ProbeDataEvent events.
   */
   const string EVENT_CHANNEL_PROBE_DATA = "ProbeDataEventChannel";

   /**
   * The name of the CORBA event channel used to push all LinkDataEvent events
   * from all available sources.
   */
   const string EVENT_CHANNEL_FULL_LINK_DATA = "FullLinkDataEventChannel";

   /**
   * The name of the CORBA event channel used to push updated LinkDataEvent events
   * from all available sources when any of the link data values change.
   */
   const string EVENT_CHANNEL_LINK_DATA_UPDATE = "LinkDataUpdateEventChannel";

   /**
   * The name of the CORBA event channel used to push LinkDataEvent events from all
```

```
* available sources filtered by priority.
*/
const string EVENT_CHANNEL_FILTERED_LINK_DATA = "FilteredLinkDataEventChannel";

/**
* The name of the CORBA event channel used to push all LinkDataEvent events from
* the sources allowing free distribution of their data (all link data sources but
* the SpeedInfo).
*/
const string EVENT_CHANNEL_FULL_LINK_DATA_LIMITED_SOURCES =
                                    "FullLinkDataLimitedSourcesEventChannel";

/**
* The name of the CORBA event channel used to push updated LinkDataEvent events
* when any of the link data values change. This stream includes only those link
* data sources that allow free data distribution (no SpeedInfo data).
*/
const string EVENT_CHANNEL_LINK_DATA_UPDATE_LIMITED_SOURCES =
                                    "LinkDataUpdateLimitedSourcesEventChannel";

/**
* The name of the CORBA event channel used to push IncidentDetailsEvent events
* when the detailed information about an incident alert changes.
*/
const string EVENT_CHANNEL_INCIDENT_DETAILS = "IncidentDetailsEventChannel";

/**
* The name of the CORBA event channel used to push IncidentUpdateEvent events
* when the additional information is added to an incident alert.
*/
const string EVENT_CHANNEL_INCIDENT_UPDATE = "IncidentUpdateEventChannel";

/**
* The name of the CORBA event channel used to push NextBusUpdateEvent events
* when data is available from NextBus.
*/
const string EVENT_CHANNEL_NEXTBUS_UPDATE = "NextBusUpdateEventChannel";

/**
* The name of the CORBA event channel used to push GeneralPurposeEvent events.
*/
const string EVENT_CHANNEL_GENERAL_PURPOSE = "GeneralPurposeEventChannel";

/**
* The name of the CORBA event channel used to push LinkDataWinnersEvent events.
*/
const string EVENT_CHANNEL_LINK_DATA_WINNERS = "LinkDataWinnersEventChannel";

/**
* The name of the CORBA event channel used to push InvalidSegmentTraversalEvent events.
*/
const string EVENT_CHANNEL_INVALID_SEGMENT_TRAVERSAL = "InvalidSegmentTraversalEventChannel";

/**
* The name of the CORBA event channel used to push LinkCongestionLevelEvent events.
*/
const string EVENT_CHANNEL_LINK_CONGESTION_LEVEL = "LinkCongestionLevelEventChannel";

/**
* The name of the CORBA event channel used to push CongestionReportStatusEvent events.
*/
const string EVENT_CHANNEL_CONGESTION_REPORT_STATUS = "CongestionReportStatusEventChannel";

/**
* The name of the CORBA event channel used to push NextBusUpdateEvent events
* when data is available from NextBus.
*/
const string EVENT_CHANNEL_TRANSIT_AGENCY_UPDATE = "TransitAgencyUpdateEventChannel";

/**
* This typedef defines the type of date/time fields used throughout the IDL.
* This number specifies (in UTC (Universal Coordinated Time) or Greenwich
* Mean Time) the number of seconds since the standard base time known as
* the "Unix epoch" (namely January 1, 1970, 00:00:00 GMT.)  This matches
* the time_t value in the C language and in Java it corresponds to
* Date.getTime() / 1000.  A value of 0 should be interpreted as an
```

```
* undefined date/time and this time should not be used for display or
* processing.
*/
typedef long UTCDateTime;


/**
* This structure describes the condition of a link.
* Volume and occupancy are either both specified or both not available.
*
* @member id            The link ID.
* @member speed         The average vehicle speed on the link in KPH
*                       (-1 if not available)
* @member travelTime    The average travel time on the link, in seconds.
*                       (-1 if not available)
* @member volume        Number of vehicles per hour per lane (must accompany occupancy)
*                       (-1 if not available)
* @member occupancy     Average percentage of time that roadway detectors are
*                       activated by traffic flow (0.0 - 100.0).  (must accompany volume)
*                       ( < 0 if not available)
*/
struct LinkCondition
{
   long   id;
   short  speed;        /* Optional */
   short  travelTime;   /* Optional */
   short  volume;       /* Optional (but accompanies occupancy) */
   float  occupancy;    /* Optional (but accompanies volume) */
};


/**
* Type used to return a sequence of LinkCondition structures.
*/
typedef sequence<LinkCondition> LinkConditionList;


/**
* A link condition list with source name included. The source name is the
* name of the program that produced the link condition data.
*
* @member sourceName The name of the program that produced the data.
* @member linkConditions The link conditions
*/
struct LinkConditionListWithSource
{
   string linkConditionSource;
   LinkConditionList linkConditions;
};


/**
* This structure describes the condition of a segment, calculated from
* a single vehicle sample (i.e., a segment traversed by a single vehicle).
*
* @member id            The segment ID.
* @member speed         The speed on the segment in KPH
*                       (-1 if not available)
* @member travelTime    The travel time on the segment in seconds.
*                       (-1 if not available)
*/
struct SegmentCondition
{
   long   id;
   short  speed;        /* Optional */
   short  travelTime;   /* Optional */
};


/**
* Type used to return a sequence of SegmentCondition structures.
*/
typedef sequence<SegmentCondition> SegmentConditionList;


/**
```

```
* This structure describes the condition of a segment as calculated
* using an average of potentially many vehicle samples.  Speed and
* travel time are optional fields, but at least one must be specified.
*
* @member id            The segment ID.
* @member probeCount    The number of vehicle probe samples included in the average, > 0
* @member speed         The average vehicle speed on the segment in KPH
*                       (-1 if not available)
* @member travelTime    The average vehicle travel time on the segment in seconds.
*                       (-1 if not available)
*/
struct AveragedSegmentCondition
{
   long   id;
   long   probeCount;
   short  speed;        /* Optional */
   short  travelTime;   /* Optional */
};


/**
* Type used to return a sequence of Averaged SegmentCondition structures.
*/
typedef sequence<AveragedSegmentCondition> AveragedSegmentConditionList;


/**
* This structure represents an instance of a vehicle's probe being read.
*
* @member encryptedID   The encrypted ID of the vehicle probe
* @member locationID    The location that the vehicle last passed
* @member readTime      The time when the probe was read by the reader
*/
struct ProbeData
{
   long encryptedID;
   long locationID;
   UTCDateTime readTime;
};


/**
* Type used to return a sequence of ProbeData structures.
*/
typedef sequence<ProbeData> ProbeDataList;


/**
* This structure represents an instance of an invalid segment traversal,
* i.e. a traversal between an ordered pair of Data Collection location IDs,
* which does not comprise a known (valid) Data Collection segment.
*
* @member startLocationID  The traversal's start location ID.
* @member endLocationID    The traversal's end location ID.
*/
struct InvalidSegmentTraversal
{
   long startLocationID;
   long endLocationID;
};


/**
* Type used to return a sequence of InvalidSegmentTraversal structures.
*/
typedef sequence<InvalidSegmentTraversal> InvalidSegmentTraversalList;


/**
* The type of traffic data being provided for a link.
*
* The enumerated values correspond to the values in TMDD-DE3426,
* so this enumeration should not be changed without consulting the
* current version of the TMDD spec.
*
* @member OTHER         Other, no additional information required
```

```
* @member OTHER_PLUS    Other, additional information required
* @member ACTUAL        Actual data
* @member RECONSTRUCTED Reconstructed data
* @member HISTORICAL    Historical data
* @member PREDICTED     Predicted data
* @member SMOOTHED      Smoothed data
* @member AVERAGED      Averaged data
*/
enum TMDD_LINK_DATA_TYPES
{
   OTHER,
   OTHER_PLUS,
   ACTUAL,
   RECONSTRUCTED,
   HISTORICAL,
   PREDICTED,
   SMOOTHED,
   AVERAGED
};


/**
* Valid values for direction of an event.
*
* @member UNKNOWN The event direction is unknown
* @member NORTH The event affects northbound traffic
* @member SOUTH The event affects southbound traffic
* @member EAST The event affects eastbound traffic
* @member WEST The event affects westbound traffic
* @member BOTH The event affects both directions of traffic
*              (unspecified whether it is east/west,
*               north/south, inner/outer loop, etc)
* @member NORTHSOUTH The event affects both northbound and southbound traffic
* @member EASTWEST The event affects both eastbound and westbound traffic
*/
enum EventDirection
{
   UNKNOWN,
   NORTH,
   SOUTH,
   EAST,
   WEST,
   BOTH,
   NORTHSOUTH,
   EASTWEST
};


/**
* A set of data elements for a specified link describing the traffic flow
* parameters.
*
* This structure is based on TMDD Current-Link-State message
* (currently section 4.2.1.3 of the TMDD specification).
*
* @member identifier    Unique alphanumeric identifier of the link. (<= 32 characters)
* @member dataType      The type of data currently available for this link.
* @member dataTypeOther An optional description of the other data type if dataType is OTHER_PLUS
*                       (empty string if not available / applicable)
* @member delay         Average additional time (in seconds) required to travel end to end on
*                       the link beyond the time recorded during free-flow conditions
*                       (-1 if not available)
* @member travelTime   The average travel time (in seconds) for a vehicle to travel
*                       end to end on the link
*                       (-1 if not available)
* @member volume        Average number of vehicles per hour for all lanes
*                       (-1 if not available)
* @member speedAverage Average speed of vehicles on the link in kilometers per hour (KM/h)
*                       (-1 if not available)
* @member density       Average number of vehicles per kilometer on the link.
*                       (-1 if not available)
* @member headway       Average estimated time between vehicles on the link, in seconds.
*                       (-1 if not available)
* @member occupancy     Average percentage of time that roadway detectors on the link are
*                       activated by traffic flow.
*                       (-1 if not available)
```

```
*/
struct LinkData
{
   string identifier;
   TMDD_LINK_DATA_TYPES dataType;
   string dataTypeOther;      /* Optional */
   long delay;                /* Optional */
   long travelTime;           /* Optional */
   long volume;               /* Optional */
   long speedAverage;         /* Optional */
   long density;              /* Optional */
   long headway;              /* Optional */
   long occupancy;            /* Optional */
};


/**
* Type used to return a sequence of LinkData structures.
*/
typedef sequence<LinkData> LinkDataList;


/**
* Contains the details representing the current state of an Incident alert,
* as obtained from the CHP (California Highway Patrol) interface.
*
* This corresponds to Framework message 5000 (TI-CHP-Raw-Incident-Data)
*
* @member candidateID            Event ID using a 19 character string in
*                                the format YYYYMMDDHHMMSSOrgID, where
*                                OrgID is 5 characters
* @member chpID                  CHP internal ID
* @member reportingOrgID         5-character Org ID (will always
*                                indicate CHP)
* @member confirmedDateTime      The time when the incident was
*                                confirmed, or 0 if not yet confirmed
* @member lastUpdateDateTime     The time when the incident was last
*                                updated, or 0 if not yet confirmed
* @member chpIncidentType        The name of the type of incident
* @member city                   The name of the city
* @member county                 The name of the county
* @member direction              The direction(s) of traffic affected
*                                (Set to UNKNOWN if not available)
* @member description            A description of the incident
* @member locationPage           Page in Thomas Bros. Map where incident
*                                location can be found.
* @member locationBeat           The beat (circuit traversed by CHP) in
*                                which the incident occurred
* @member location               Description of the incident location
* @member locationArea           The area in which the incident occurred
* @member latitude               The latitude in decimal degrees, or
*                                0.0 if not available
* @member longitude              The longitude in decimal degrees, or
*                                0.0 if not available
* @member incidentPriority       0=Unknown, 1=Highest, 10=Lowest
*                                (0 if not available)
* @member associatedIncidentIDs
* @member clearedDateTime        The time when the incident was cleared
* @member filedStatus
*/
struct CHPIncidentDetails
{
   string candidateID;
   string chpID;
   string reportingOrgID;
   UTCDateTime confirmedDateTime;
   UTCDateTime lastUpdateDateTime;
   string chpIncidentType;
   string city;                   /* Optional */
   string county;                 /* Optional */
   EventDirection direction;      /* Optional */
   string description;
   string locationPage;           /* Optional */
   string locationBeat;           /* Optional */
   string location;
   string locationArea;           /* Optional */
```

```
   double latitude;               /* Optional */
   double longitude;              /* Optional */
   long incidentPriority;         /* Optional */
   string associatedIncidentIDs;  /* Optional */
   UTCDateTime clearedDateTime;   /* Optional */
   string filedStatus;            /* Optional */
};

/**
* Type used to return a sequence of CHPIncidentDetails structures.
*/
typedef sequence<CHPIncidentDetails> CHPIncidentDetailsList;


/**
* Contains textual updates when new information is added to the incident
* alert, as obtained from the CHP interface.
*
* Corresponds to Framework message 5001 (TI-CHP-Incident-Status)
*
* @member candidateID            Event ID using a 19 character string in
*                                the format YYYYMMDDHHMMSSOrgID, where
*                                OrgID is 5 characters
* @member chpID                  CHP internal ID
* @member lastUpdateDateTime     The time when the incident was last
*                                updated, or 0 if not yet confirmed
* @member statusText             A short description of the status of the
*                                incident.
*/
struct CHPIncidentUpdate
{
   string candidateID;
   string chpID;
   UTCDateTime lastUpdateDateTime;
   string statusText;
};


/**
* Type used to return a sequence of CHPIncidentUpdate structures.
*/
typedef sequence<CHPIncidentUpdate> CHPIncidentUpdateList;

/**
* Contains next arrival time data for transit agency data received from NextBus.
* Each stop contains up to three predicted arrival times.
*
* Corresponds to Framework message 351 (NextBus_ArrivalTimes)
*
* @member stopID                 Identifies a stop. A stop is identified by
*                                the combination of a route, direction, and
*                                for a transit agency.
* @member nextArrivalTime        The next predicted arrival datetime for the
*                                stop.
* @member secondArrivalTime      The second predicted arrival time for the
*                                stop.
* @member thirdArrivalTime       The third predicted arrival time for the
*                                stop.
* @member NextBusMessage         Optional message related to the stop.
* @member Status                 Optional flag to indicate non-receipt of data
*                                from NextBus for this stop.  Values are as follows:
*                                0 or null - Normal prediction data received
*                                1 - Indicates that no data was received for this stop
*                                2 - This is the last arrival for this stop
*                                3 - There are no more arrivals for this stop
*/
struct NextBusArrivalTimes
{
   long    stopID;
   UTCDateTime nextArrivalTime;    /* Optional */
   UTCDateTime secondArrivalTime;  /* Optional */
   UTCDateTime thirdArrivalTime;   /* Optional */
   string  NextBusMessage;         /* Optional */
   string  status;                 /* Optional */
};
```

```
   /**
   * Type used to return a sequence of ArrivalTimes structures.
   */
   typedef sequence<NextBusArrivalTimes> NextBusArrivalTimesList;

   /**
   * Contains next arrival time data for transit agency data received from NextBus.
   * Contains the arrival time list and the no prediction stop list.
   *
   * Corresponds to Framework message 350 (NextBus-Update)
   *
   * @member agencyName             Name of transit agency
   * @member NBPredictionDateTime   Date/time that this data was received from NextBus.
   *                                This number specifies (in UTC (Universal Coordinated Time)
   *                                or Greenwich Mean Time) the number of seconds since the
   *                                 standard base time known as the "Unix epoch" (namely January 1,
1970, 00:00:00 GMT.)
   * @member arrivalTimes           The list of next arrival times.
   * @member agencyStatus           Flag to indicate non-receipt of data
   *                                from NextBus for an agency.  Values are as follows:
   *                                0 - Ok, prediction data received
   *                                1 - Indicates that no data was received for this agency
   */
   struct NextBusUpdate
   {
      string agencyName;
      UTCDateTime NBPredictionDateTime;
      NextBusArrivalTimesList arrivalTimes;   /* Optional */
      string agencyStatus;
   };

   /**
   * Type used to return a sequence of arrival times structures.
   */
   typedef sequence<UTCDateTime> ArrivalTimesList;

   /**
   * Contains next arrival time data for transit agency data received from TransitAgency.
   *
   * Corresponds to Framework message 351 (TransitAgency_ArrivalTimes)
   *
   * @member stopID                 Identifies a stop. A stop is identified by
   *                                the combination of a route, direction, and
   *                                for a transit agency.
   * @member nextArrivalTime        The list of predicted arrival datetimes for the
   *                                stop.
   * @member TransitAgencyMessage   Optional message related to the stop.
   * @member Status                 Optional flag to indicate non-receipt of data
   *                                from TransitAgency for this stop.  Values are as follows:
   *                                0 or null - Normal prediction data received
   *                                1 - Indicates that no data was received for this stop
   *                                2 - This is the last arrival for this stop
   *                                3 - There are no more arrivals for this stop
   */
   struct TransitAgencyArrivalTimes
   {
      long    stopID;
      ArrivalTimesList arrivalTimesArray; /* Optional */
      string  TransitAgencyMessage;      /* Optional */
      string  status;                    /* Optional */
   };

   /**
   * Type used to return a sequence of ArrivalTimes structures.
   */
   typedef sequence<TransitAgencyArrivalTimes> TransitAgencyArrivalTimesList;

   /**
   * Contains next arrival time data for transit agency data received from TransitAgency.
   * Contains the arrival time list and the no prediction stop list.
   *
   * Corresponds to Framework message 350 (TransitAgency-Update)
   *
   * @member agencyName             Name of transit agency
   *  @member  TransitAgencyPredictionDateTime      Date/time  that  this  data  was  received  from
TransitAgency.
```

```
   *                               This number specifies (in UTC (Universal Coordinated Time)
   *                               or Greenwich Mean Time) the number of seconds since the
   *                                standard base time known as the "Unix epoch" (namely January 1,
1970, 00:00:00 GMT.)
   * @member arrivalTimes          The list of next arrival times.
   * @member agencyStatus          Flag to indicate non-receipt of data
   *                               from TransitAgency for an agency.  Values are as follows:
   *                               0 - Ok, prediction data received
   *                               1 - Indicates that no data was received for this agency
   */
   struct TransitAgencyUpdate
   {
      string agencyName;
      UTCDateTime TransitAgencyPredictionDateTime;
      TransitAgencyArrivalTimesList arrivalTimes;   /* Optional */
      string agencyStatus;
   };


   /**
   * A set of data elements for general purpose data.
   *
   * @member majorCode     Major code
   * @member minorCode     Minor code
   * @member codeText      General purpose description
   */
   struct GeneralPurpose
   {
      long majorCode;
      long minorCode;
      string codeText;
   };

   /**
   * Type used to return a sequence of GeneralPurpose structures.
   */
   typedef sequence<GeneralPurpose> GeneralPurposeList;


   /**
   * A set of data elements for ALDF link data winners.
   *
   * @member linkID            Unique identifier of the link.
   * @member speedWinnerName   Name of winner link.
   */
   struct LinkDataWinners
   {
      long linkID;
      string speedWinnerName;
   };


   /**
   * Type used to return a sequence of LinkDataWinners structures.
   */
   typedef sequence<LinkDataWinners> LinkDataWinnersList;


   /**
   * The possible congestion levels.
   *
   * @member CONGESTION_GREEN        Normal traffic flow.
   * @member CONGESTION_YELLOW         Somewhat impaired traffic flow: worse than GREEN, but better
than RED.
   * @member CONGESTION_RED             Impaired traffic flow: worse than YELLOW, but better than
BLACK.
   * @member CONGESTION_BLACK        Stop-and-go traffic.
   * @member CONGESTION_UNKNOWN      Unknown traffic conditions.
   */
   enum LINK_CONGESTION_LEVELS
   {
      CONGESTION_GREEN,
      CONGESTION_YELLOW,
      CONGESTION_RED,
      CONGESTION_BLACK,
      CONGESTION_UNKNOWN
```

```
   };

   /**
   * This structure describes the level of congestion on a link.
   *
   * @member id                The link ID.
   * @member congestionLevel    The level of congestion as defined in the LINK_CONGESTION_LEVELS
enumeration.
   */
   struct LinkCongestionLevel
   {
      long   id;
      LINK_CONGESTION_LEVELS  congestionLevel;
   };


   /**
   * Type used to return a sequence of LinkCongestionLevel structures.
   */
   typedef sequence<LinkCongestionLevel> LinkCongestionLevelList;


   /**
   * A link congestion list with the report type included. The report type
   * can be either A (absolute congestion report) or R (congestion report
   * based on the comparison of the current link's speed with its historical
   * value for the given day/time).
   *
   * @member reportType The report type
   * @member linkCongestionList The list of LinkCongestionLevel objects.
   */
   struct LinkCongestionLevelListWithType
   {
      string reportType;
      LinkCongestionLevelList linkCongestionList;
   };


   /**
   * The possible congestion report status values.
   *
   * @member STATUS_NEW        New congestion report.
   * @member STATUS_UPDATED    Update to the existing congestion report.
   * @member STATUS_EXPIRED    Expired congestion report.
   * @member STATUS_CANCELLED  Congestion report cancelled by the user.
   */
   enum CONGESTION_REPORT_STATUS
   {
      STATUS_NEW,
      STATUS_UPDATED,
      STATUS_EXPIRED,
      STATUS_CANCELLED
   };


   /**
   * This structure describes the status of the congestion report.
   *
   * @member id                    The link ID.
   * @member speed                 The speed assigned to the link in KPH (-1 if not available).
   *                               This field may be skipped if the 'status' is STATUS_CLOSED.
   * @member estDuration           Estimated duration in minutes, i.e. the time interval for this
   *                               report to remain valid unless it gets manually cancelled before
   *                               (-1 if not available). This field may be skipped if the 'status'
   *                               is STATUS_CLOSED.
   * @member createdDateTime       Date/time this report was created. It will be used for
   *                               calculating the report's expiration time.
   * @member lastUpdatedDateTime   Date/time this report was last updated.
   * @member status                Congestion report status: new, update or closure.
   */
   struct CongestionReportStatus
   {
      long                      linkID;
      short                     speed;                /* Optional */
      long                      estDuration;          /* Optional */
      UTCDateTime               createdDateTime;
```

```
      UTCDateTime               lastUpdatedDateTime;
      CONGESTION_REPORT_STATUS  status;
   };


   /**
   * Type used to return a sequence of CongestionReportStatus structures.
   */
   typedef sequence<CongestionReportStatus> CongestionReportStatusList;


   /**
   * This enum lists the types of CORBA events that may be pushed.
   *
   * @member LinkConditionEvent            pushed when an updated link condition is known
   * @member SegmentConditionEvent         pushed when an updated segment condition is known
   * @member AveragedSegmentConditionEvent pushed when an updated averaged segment condition is
known
   * @member ProbeDataEvent               pushed when new probe data is available
   * @member LinkDataEvent                pushed when a link's status changes
   * @member IncidentDetailsEvent             pushed when an incident's details (state data) are
changed
   * @member IncidentUpdateEvent          pushed when new information is added to an event
   * @member NextBusUpdateEvent               pushed when new data from NextBus is available (next
arrivals)
   * @member GeneralPurposeEvent          pushed when new general purpose data is available
   * @member LinkDataWinnersEvent         pushed when link data winners list is available
   * @member InvalidSegmentTraversalEvent pushed when an invalid segment traversal is detected
   * @member LinkCongestionLevelEvent     pushed when a new link congestion report is created
   * @member CongestionReportStatusEvent      pushed when a congestion report is added or
cancelled/expired
   * @member TransitAgencyUpdateEvent        pushed when new data from TransitAgency is available
(next arrivals)
   *
   * @see DataCollectionEvent
   */
   enum DataCollectionEventType
   {
      LinkConditionEvent,
      SegmentConditionEvent,
      AveragedSegmentConditionEvent,
      ProbeDataEvent,
      LinkDataEvent,
      IncidentDetailsEvent,
      IncidentUpdateEvent,
      NextBusUpdateEvent,
      GeneralPurposeEvent,
      LinkDataWinnersEvent,
      InvalidSegmentTraversalEvent,
      LinkCongestionLevelEvent,
      CongestionReportStatusEvent,
      TransitAgencyUpdateEvent
   };

   /**
   * This union identifies the data to be passed with events that are pushed
   * through the CORBA event service.
   *
   * @see DataCollectionEventType
   */
   union DataCollectionEvent switch(DataCollectionEventType)
   {
      case LinkConditionEvent:
         LinkConditionListWithSource linkConditionEventInfo;
      case SegmentConditionEvent:
         SegmentConditionList segmentConditionEventInfo;
      case AveragedSegmentConditionEvent:
         AveragedSegmentConditionList averagedSegmentConditionEventInfo;
      case ProbeDataEvent:
         ProbeDataList probeDataEventInfo;
      case LinkDataEvent:
         LinkDataList linkDataEventInfo;
      case IncidentDetailsEvent:
         CHPIncidentDetailsList incidentDetails;
      case IncidentUpdateEvent:
         CHPIncidentUpdateList incidentUpdate;
```

```
      case NextBusUpdateEvent:
         NextBusUpdate nextBusUpdate;
      case GeneralPurposeEvent:
         GeneralPurposeList generalPurpose;
      case LinkDataWinnersEvent:
         LinkDataWinnersList linkDataWinners;
      case InvalidSegmentTraversalEvent:
         InvalidSegmentTraversalList invalidSegmentTraversal;
      case LinkCongestionLevelEvent:
         LinkCongestionLevelListWithType linkCongestionEventInfo;
      case CongestionReportStatusEvent:
         CongestionReportStatusList congestionReportStatusEvent;
      case TransitAgencyUpdateEvent:
         TransitAgencyUpdate transitAgencyUpdate;
   };
};

#endif
```