Bus Tracker

A bus tracking app optimized for CMU students

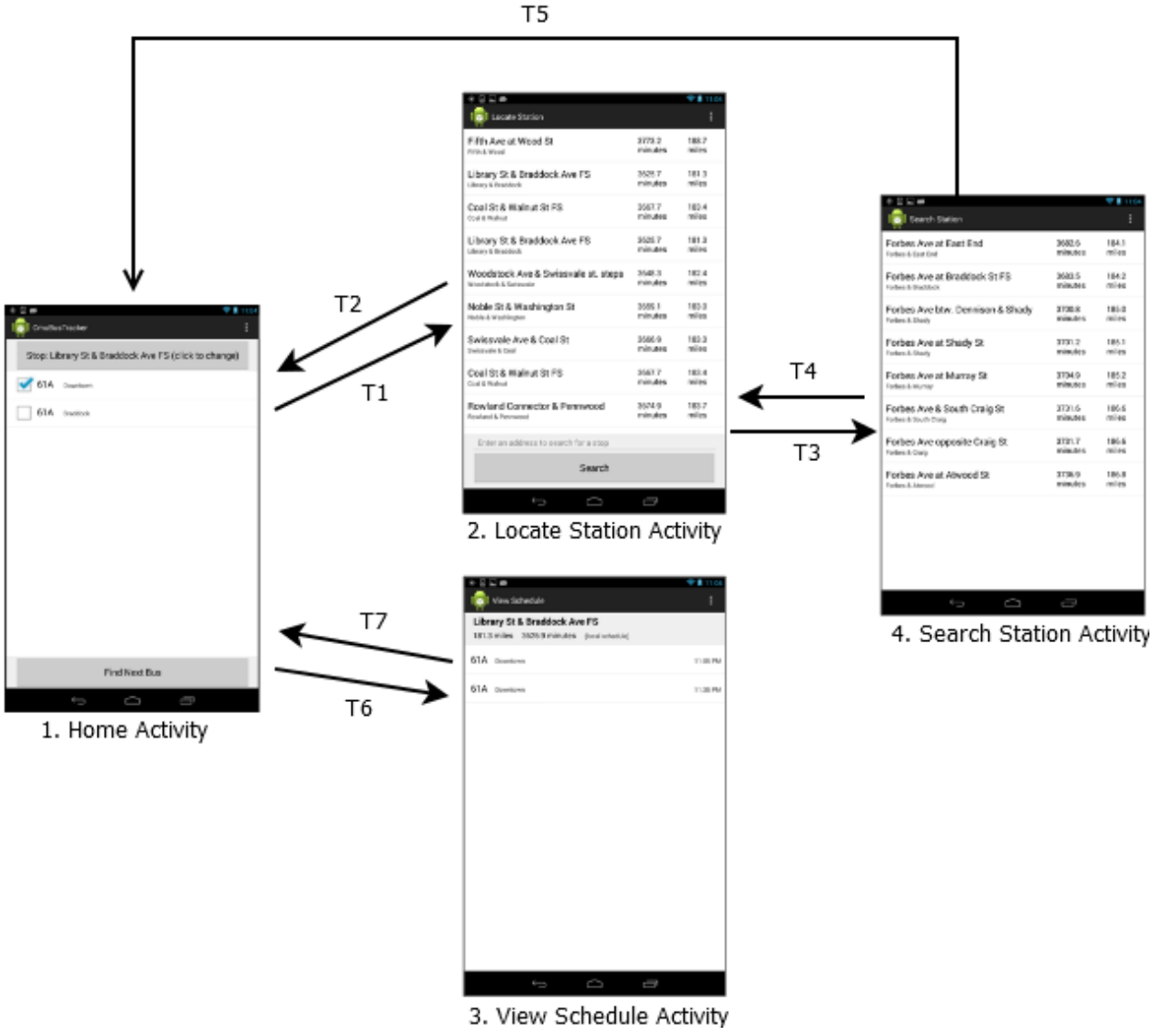
Design Document

Prepared by:

Daniel Stoll
Ryler Hockenbury
Evgeny Toropov

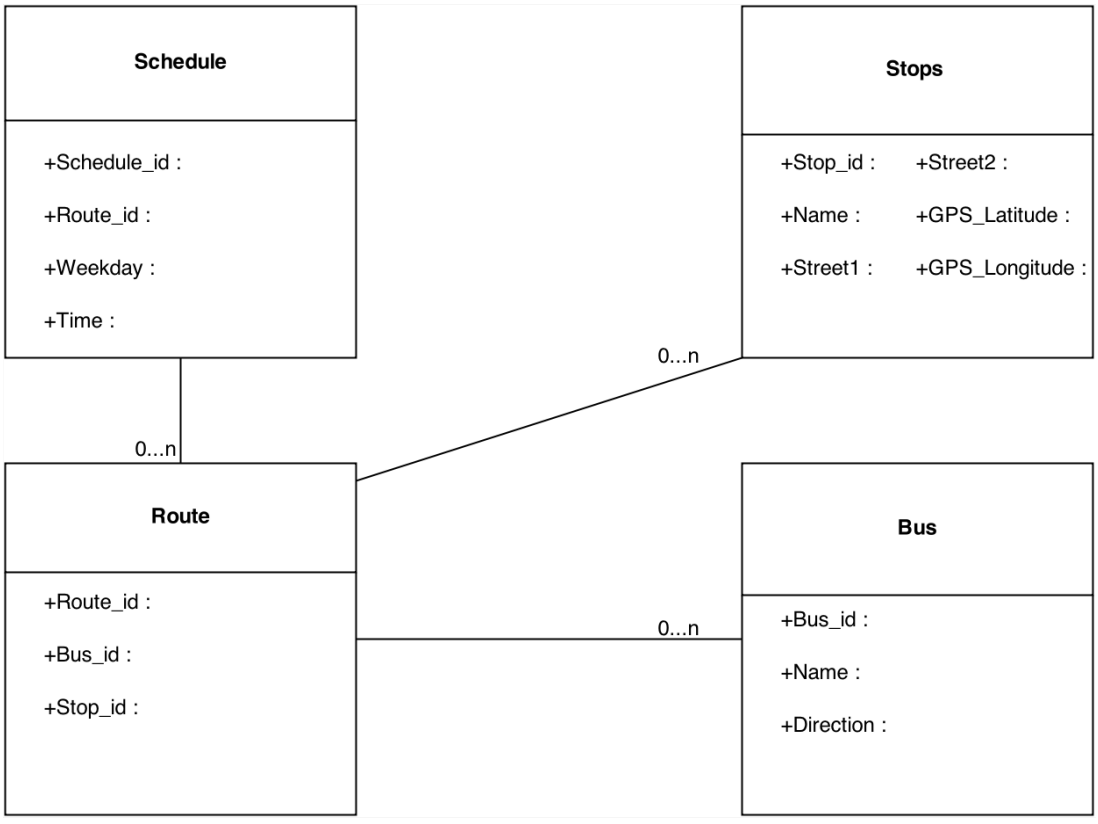
I. Presentation Layer Design

The presentation layer was designed to bring the most relevant information directly to the user, rather than making the user search for what he/she needs. For example, in the home activity the station is auto-populated with the closest station to the user because the closest station is a likely choice, and all the stops are listed by distance from the user on the locate stop activity. The use of horizontal swipes to move between activities also adds intuitive user friendly navigation to the app. Motions on the locate station activity and the view schedule activity enables a “shake to refresh” feature. Objects are transferred between activities using the parcelable interface to keep user wait-time minimized. Also, buttons and text are large to make on-the-go use easy, and to accommodate users wearing gloves in the colder season.



II. Database Design and Schema

The database was designed to provide the easiest methods to link together a Bus, Stop, and Time. The way we designed the database was to have one database each of Stops and Buses, match each Bus to a Stop in our routes table, and then match each route to a Time in our Schedule table. We also have a weekday section in our schedule database to account for the fact that the bus schedule changes on different days of the week. For example, say we wanted to get a list of the times that a list of buses would pass through a certain stop. First, we would find the matching stop and bus ids associated with the given stop and bus names. Next, we would search the route table for any route ids where the bus and stop ids match up. Finally, we would search the schedule database and return all the times associated with the route id that we found.



III. Server Design

The goal of the server is to keep the current schedule (valid for several months), track the real-time bus positions (when the Port Authority service becomes available), to provide remote phone clients with schedule or real-time data, and to send phone clients updated database. The used server is Tomcat 7.0.47.

The communication with the app is via several servlets that answer at the routes:

- queryMockServlet.java - for debugging and testing. Always sends the same data. The url is "webserver/querymock"
- queryServlet.java - collects and sends the schedule for the requested pair (stop, buses). The url is "webserver/query"
- getDatabaseServlet.java - sends the current database in a .db file. The url is "webserver/getdatabase"

Besides processing app requests the server implements web interface. The bus look up interface pipeline is the same as on the app.

The web interface is implemented via several jsp pages + beans that answer at the routes:

- chooseBuses.jsp - the homepage for web interface. It is the bus station look-up page. The url is "webserver"
- chooseStop.jsp - is the next page, where a user has to pick buses for the chosen station. The url is "webserver/chooseBuses"
- viewSchedule.jsp - is the final page, where a user sees the bus arrival times. The url is webserver/viewSchedule

In order to avoid code duplication the src code common to both the app and the webserver is stored in the app project structure. The webserver project accesses that code by creating symbolic links to those classes. These common classes include basic entities like Bus or Stop, and the network connection protocol, that describes the url parameter string and the return message in case of errors.

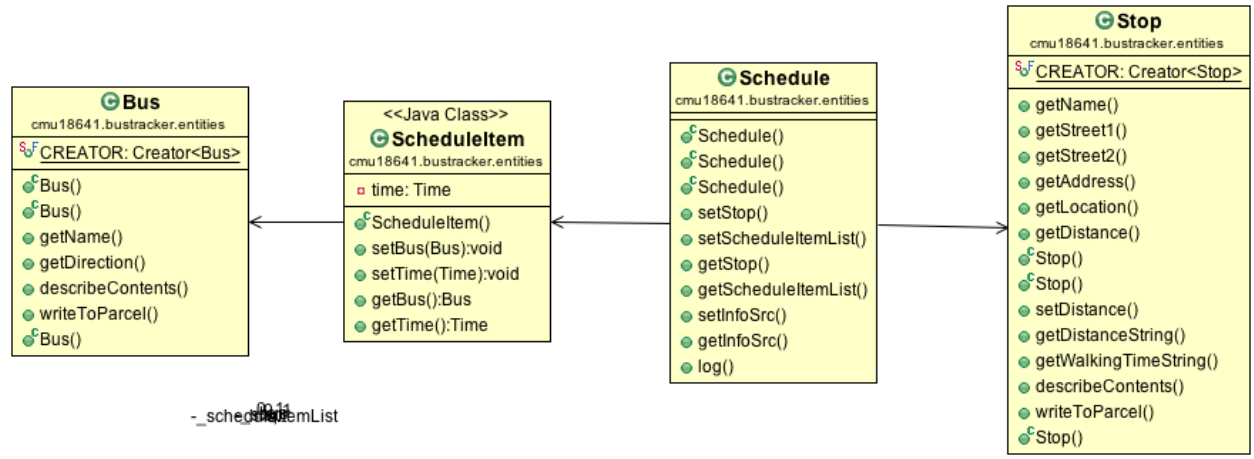
IV Package design

App:

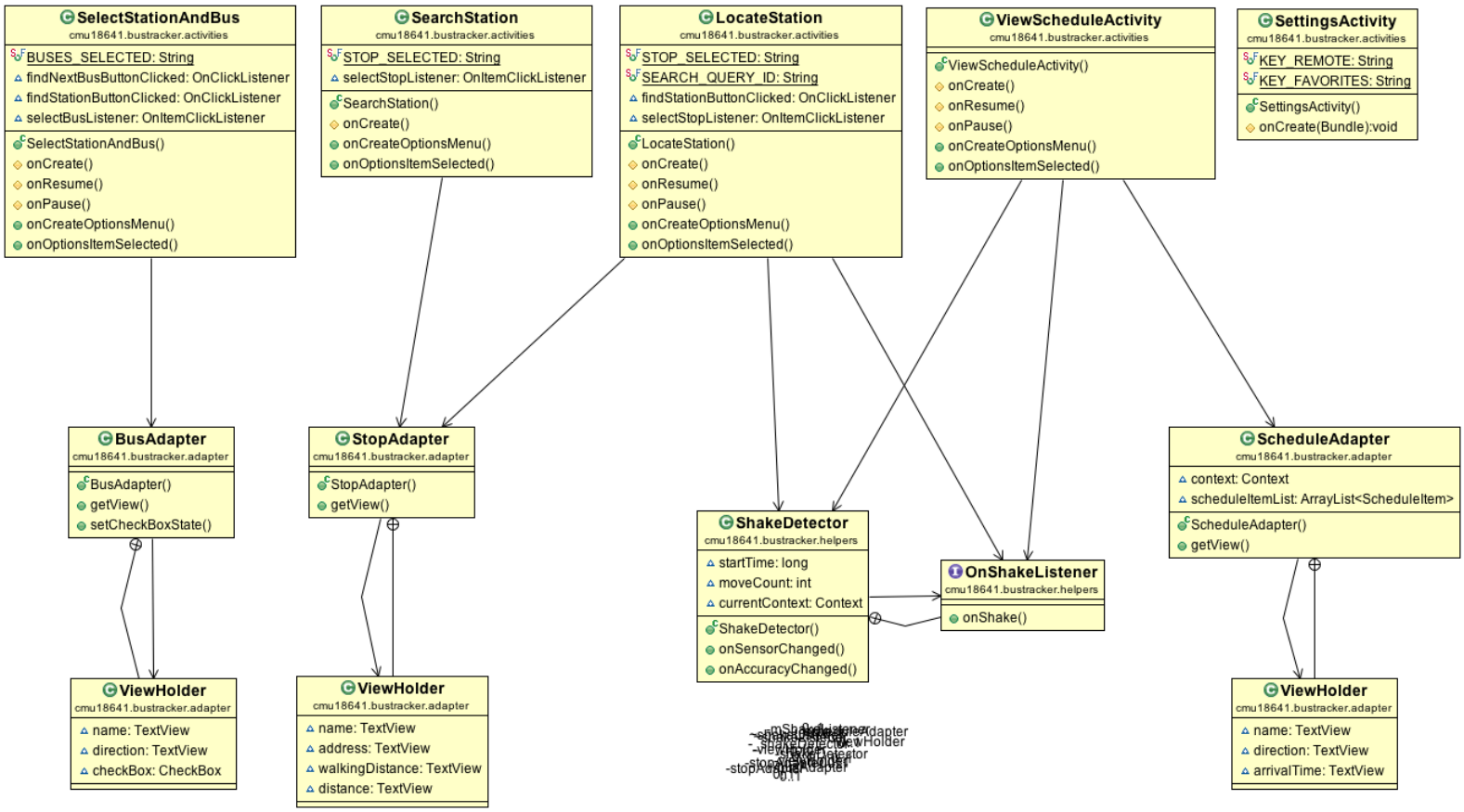
- activities
- adapter elements to display by activities
- entities data types of buses, stops, and schedules
- exceptions
- helpers for activities
- dblayout connecting to local database
- common this directory used by both app and the server projects
    - common.entities: bare minimum from entities to send to the server and back
    - common.protocols: conventions to encode and parse data for app and server
- ws the component that process all types of requests. GlobalConnector is the interface for activities.
    - ws.local processing requests using local database
    - ws.remote processing requests by querying the server

V. Class Diagram

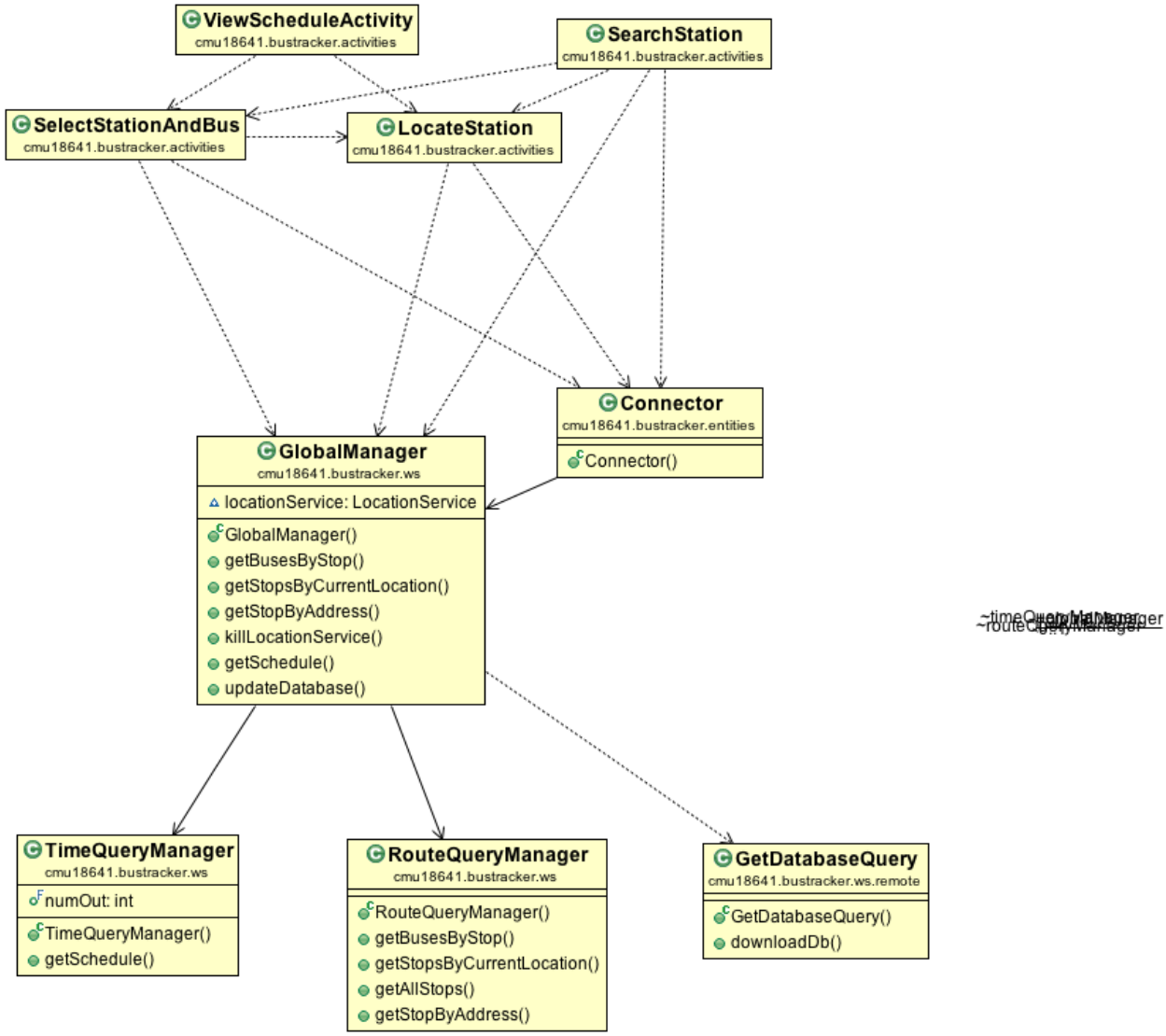
Model:



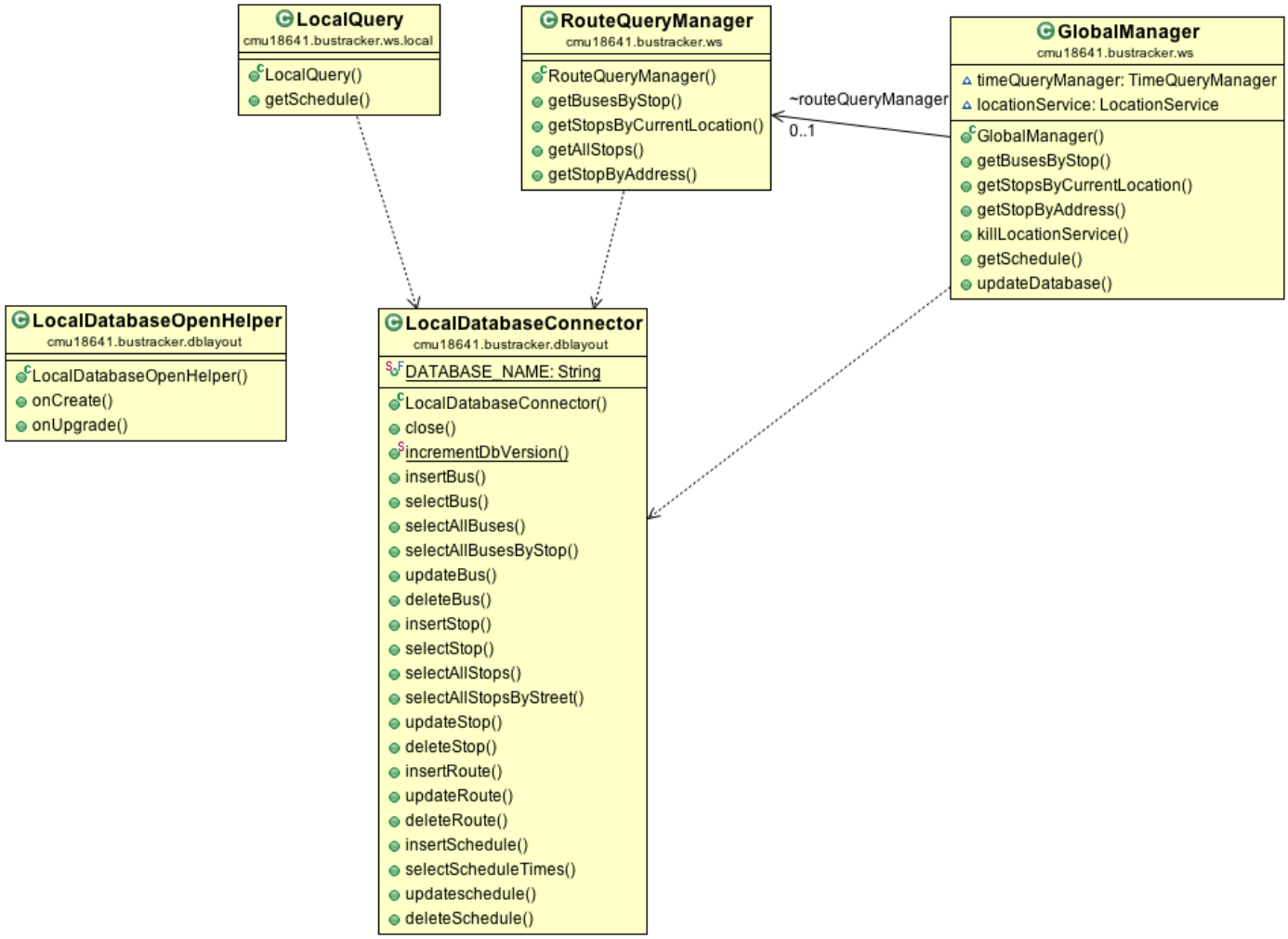
Activities & Adapters:



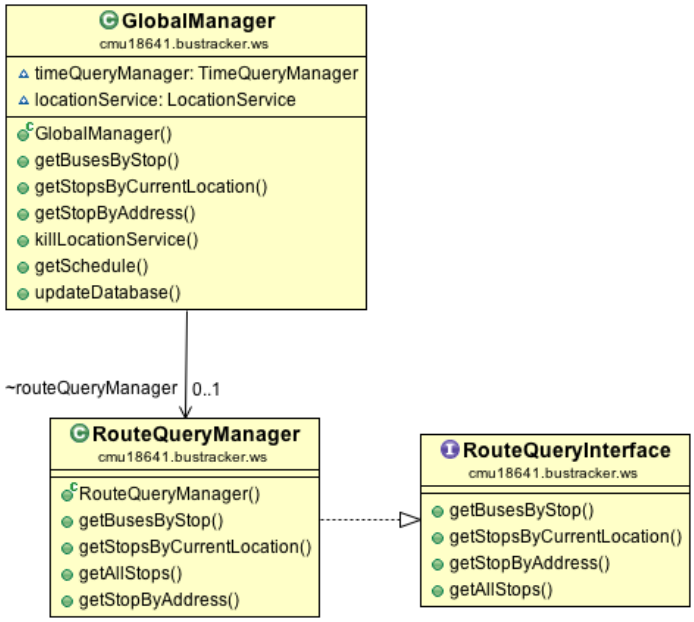
Activities & Business Layer:



Database:


LocalQuery
cmu18641.bustracker.ws.local
LocalQuery()
getSchedule()
RouteQueryManager
cmu18641.bustracker.ws
RouteQueryManager()
getBusesByStop()
getStopsByCurrentLocation()
getAllStops()
getStopByAddress()
~routeQueryManager
0..1
GlobalManager
cmu18641.bustracker.ws
timeQueryManager: TimeQueryManager
locationService: LocationService
GlobalManager()
getBusesByStop()
getStopsByCurrentLocation()
getStopByAddress()
killLocationService()
getSchedule()
updateDatabase()
LocalDatabaseOpenHelper
cmu18641.bustracker.dblayout
LocalDatabaseOpenHelper()
onCreate()
onUpgrade()
LocalDatabaseConnector
cmu18641.bustracker.dblayout
DATABASE_NAME: String
LocalDatabaseConnector()
close()
incrementDbVersion()
insertBus()
selectBus()
selectAllBuses()
selectAllBusesByStop()
updateBus()
deleteBus()
insertStop()
selectStop()
selectAllStops()
selectAllStopsByStreet()
updateStop()
deleteStop()
insertRoute()
updateRoute()
deleteRoute()
insertSchedule()
selectScheduleTimes()
updateschedule()
deleteSchedule()

Route Query:



Time Query: