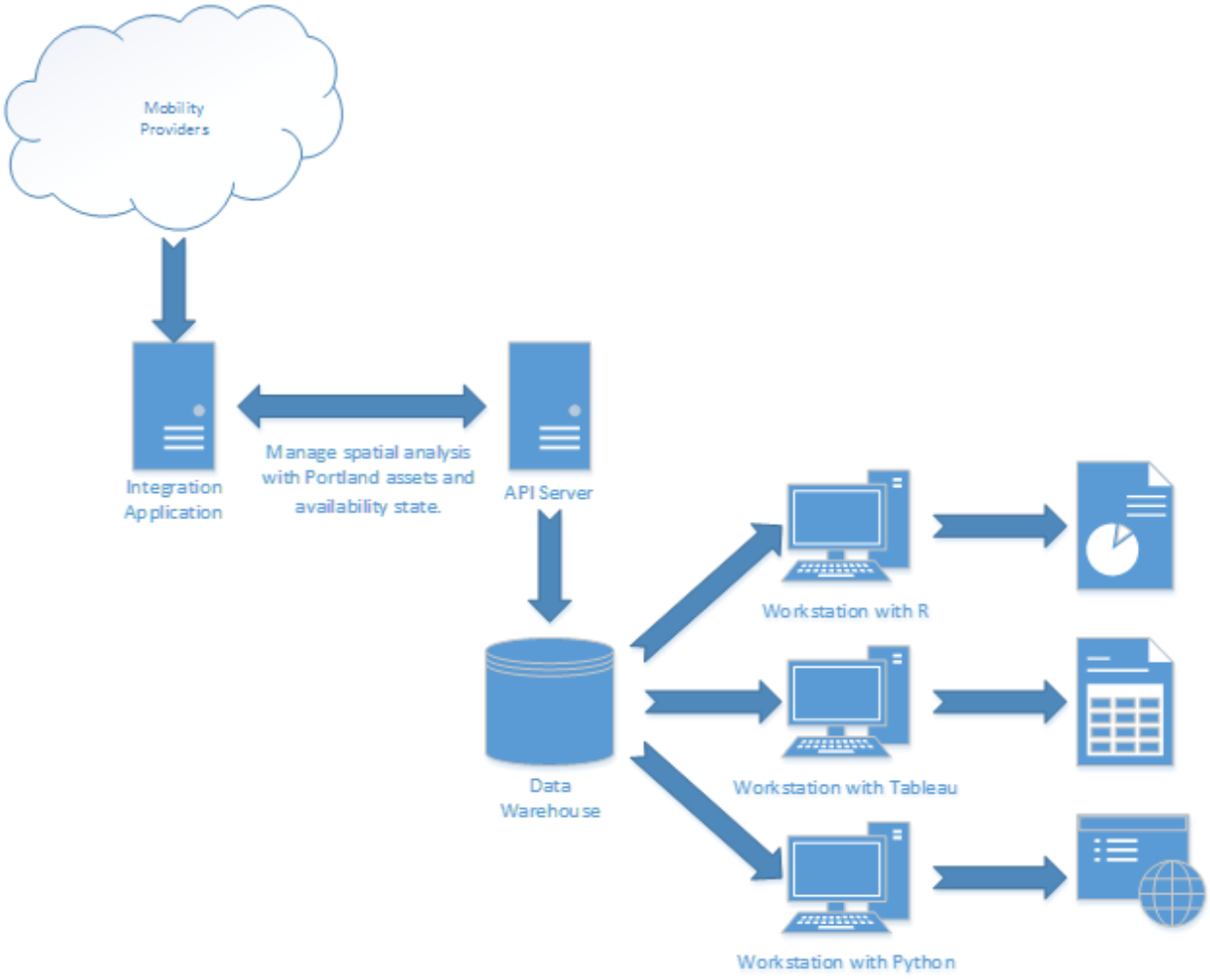
Appendix D: API Data Methodology and Limitations/Assumptions

Overview

For this pilot, PBOT asked that all mobility providers comply with an API specification based off of a version of the City of Los Angeles's Mobility Data Specification (MDS) available as of the time that the pilot permit application was posted. Portland's Bureau of Technology Services (BTS) Vertical Applications team created an application to manage the availability state of vehicles and perform some geospatial analysis of availability and trips, relating them to assets and geometric areas around the city to assist with permit compliance and data analysis. This analysis was performed by using an API server that managed interaction with the data warehouse created for this pilot. The integration application downloaded data from mobility providers, querying the API server for how the retrieved geometries related to assets and geometries stored in the data warehouse. The figure below illustrates this flow of data as well as providing some examples of products created from the data warehouse by PBOT data analysts.

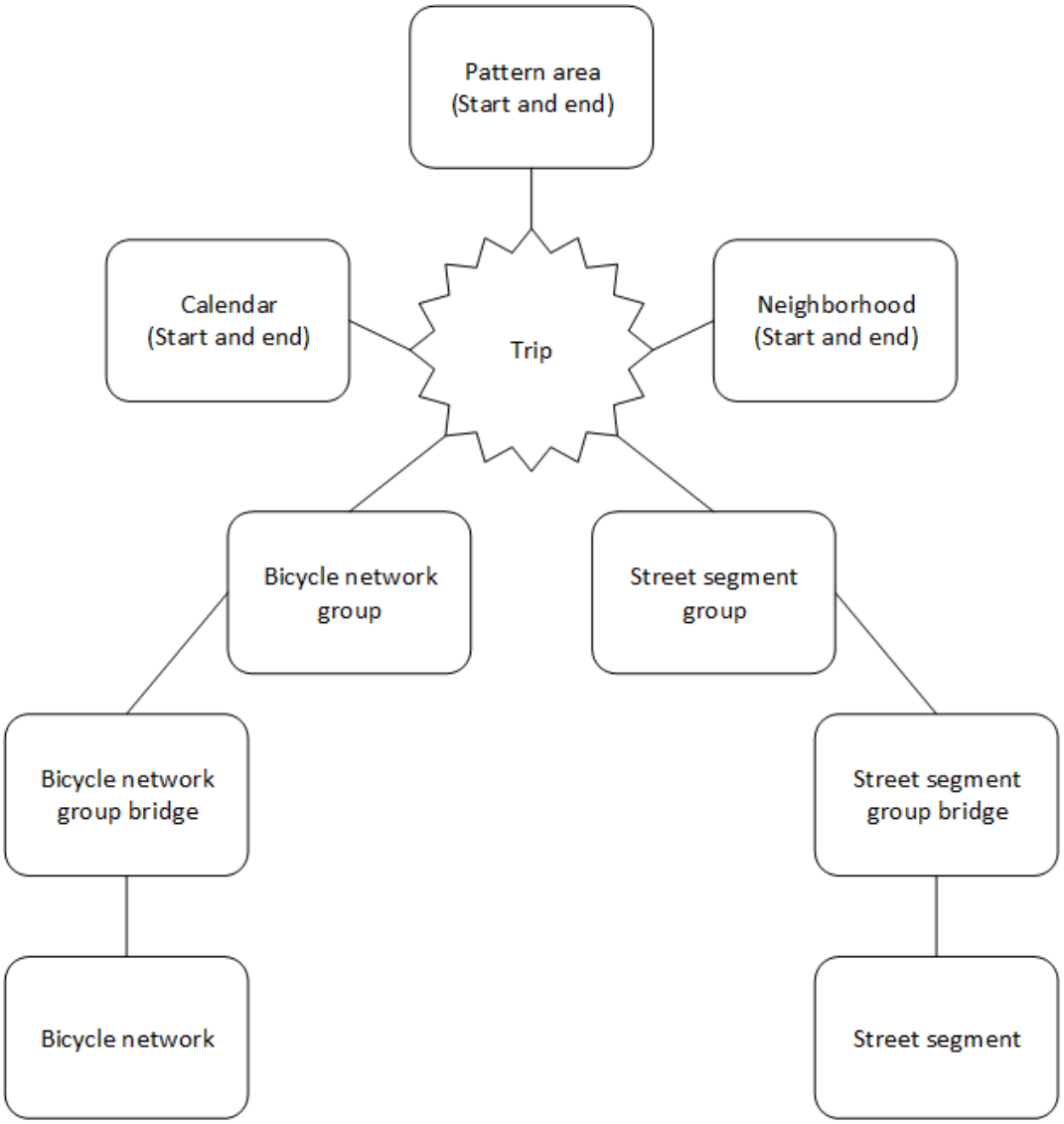


Applications and frameworks used

BTS wrote both the integration application and API server using .NET Core 2.1 for the application framework. The data warehouse used PostgreSQL 9.6 with the PostGIS extension installed.

Data Warehouse

BTS's team created a data warehouse to aid in analyzing data retrieved from mobility providers and to efficiently store data for the duration of the pilot. The data warehouse followed the principles of data warehouse schema construction as much as possible, creating facts and shared dimensions to relate them, although some fact data -- notably collisions -- related to trips data. The figure below illustrates some of the fact and dimension relationships that define the data warehouse for trip data. Using this structure, we were able to efficiently store the paths used most often by scooter users and should be able to use that data to analyze Portland's street and bicycle network as defined by use, rather than by geographic location.

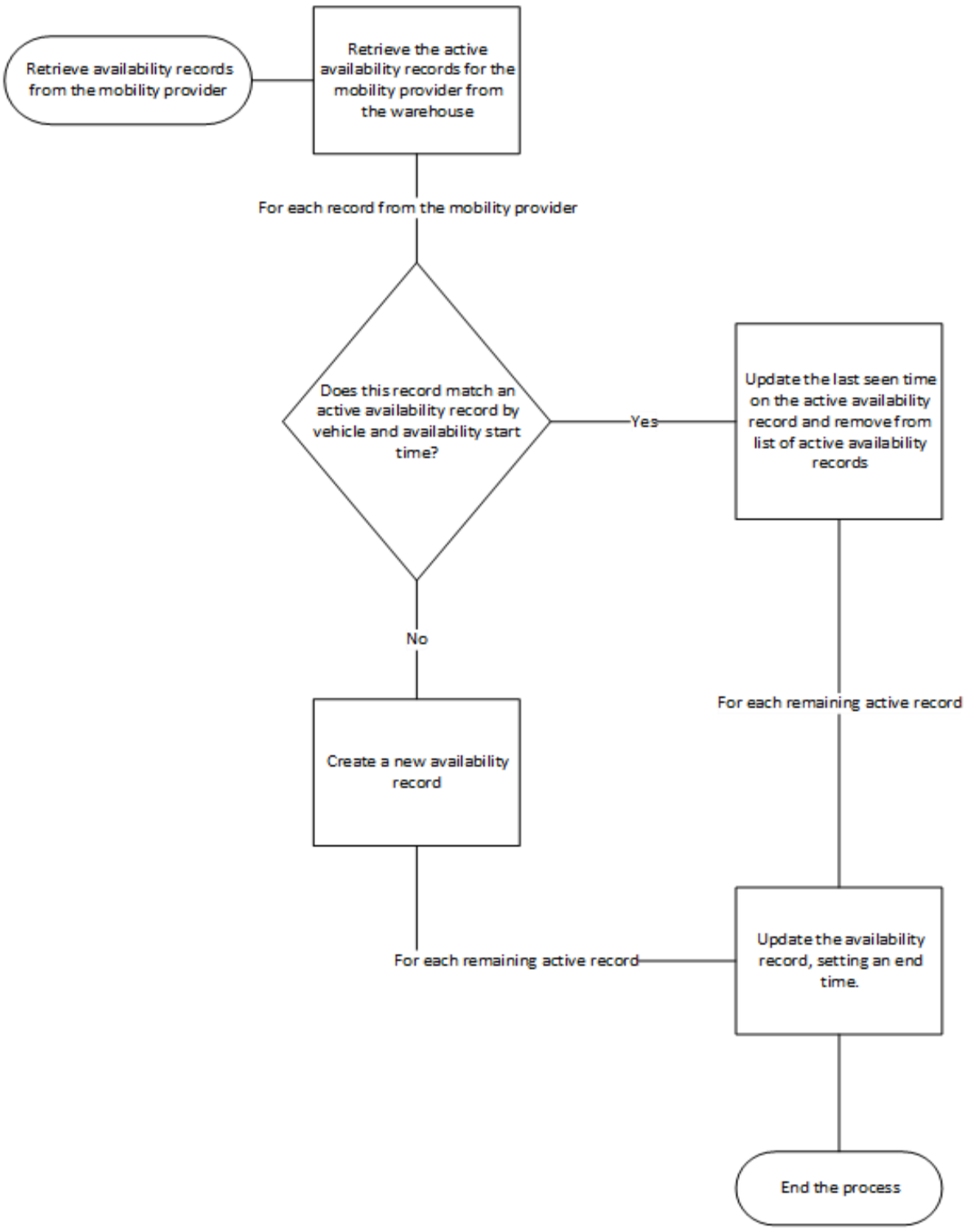


In order to relate trips and availability state in near real-time to geometric objects such as street segments, neighborhoods, neighborhood pattern areas, and bicycle network, the integration application would make API server calls with the geometric object to perform that query as it processed responses from the mobility providers. In order to relate trips to street segments or the bicycle network, each object was stored as a geometry consisting of a 100-foot octagon around the midpoint of the object. An octagon provided a compromise of performance during processing. If a trip intersected that object, then a bridge record was created to show that the trip touched that object.

API limitations

In order to implement the system described above, the integration application became a long-running service. This architecture posed problems and required regular maintenance to keep running. This was a result of the API specification used for this pilot and the interpretation of mobility providers. The API specification required that availability data be provided in a manner that allowed historic queries to show which vehicles were available at any time. Unfortunately, some mobility providers did not maintain availability data in a manner that allowed them to implement that fully, and instead the availability API was implemented as a real-time feed of which scooters are available as of the API call. This forced BTS to create this long-running service that could maintain records of availability status. While this implementation was not the intent of the permit API requirements, having standardization of implementations is preferable to maintaining different code bases for different providers.

The figure below illustrates the workflow of how BTS created availability state records.



In addition to availability data limitations and compromises, the API specification did not describe how mobility providers should implement paging when data becomes too large to return in one response. Providers diverged on how to implement this mechanism, and thus required separate implementations

to query APIs for data. While the ideal would have been one cade base to query all mobility provider APIs, the API specification was not drafted well enough to achieve that goal.

Fortunately, both of these limitations are addressed in newer versions of the Mobility Data Specification, so newer permits should use them for guidance on how to avoid these issues in their implementations.