Smart Mobility: Behavioral Data Collection and Simulation

by Akshay Padmanabha

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2017

Author: _____

Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by: _____

Moshe Ben-Akiva, Professor of Civil and Environmental Engineering, Thesis Supervisor
May 26, 2017

Accepted by: _____

Christopher Terman, Chairman, Masters of Engineering Thesis Committee

Smart Mobility: Behavioral Data Collection and Simulation

by Akshay Padmanabha

Submitted to the
Department of Electrical Engineering and Computer Science

May 26, 2017

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

On-demand ridesharing services, such as Uber and Lyft, and autonomous vehicles are significantly changing the landscape of transportation and mobility. In light of these disruptions, we aim to determine consumer preferences with regards to transportation and use this data to simulate and analyze the urban effects of smart mobility solutions. We collect behavioral data using Future Mobility Sensing (FMS), a smartphone and prompted-recall-based integrated activity-travel survey, and create simulations using the data with SimMobility, a simulation platform that integrates various mobility-sensitive behavioral models with state-of-the-art scalable simulators to predict the impact of mobility demands on transportation networks, intelligent transportation services, and vehicular emissions. Enhancing these projects with on-demand preferences, individual patterns, and incentives as inputs, we aim to simulate and analyze a wide range of viable smart mobility solutions.

Thesis Supervisor: Moshe Ben-Akiva
Title: Professor of Civil and Environmental Engineering

Table of Contents

1. Introduction

1a. Smart Mobility Solutions

One of the most pressing concerns of urban transportation is the growing, aging, and urbanizing population. New alternative forms of transportation are required to satisfy the various needs of travelers, who demand more personalized and flexible mobility. In this regard, smart mobility has received a great deal of attention in both industry and society during the last decade [1].

On-demand services are examples of smart mobility; the idea is to provide a more traveler-centric environment through convenient alternatives in an efficient manner. Demand-responsiveness is increased in such situations, as the operation of such systems is based on the received demand (hailing a ride) rather than being fixed in advance. Therefore, the decisions on the supply side are mostly dynamic and leverage emerging mobility services (e.g., autonomous and app-based shared-ride services).

A specific example of a smart mobility solution is autonomous mobility on-demand, which provides one-way car-sharing with self-driving vehicles, and has emerged as a promising solution for urban transportation. Actual implementation of an autonomous transportation service is facilitated through the optimization of fleet size and the rebalancing of vehicles during high demand periods. Likewise, another solution is flexible mobility-on demand (e.g., Uber and Lyft), which provides personalized and optimized services to travelers in real-time allowing for flexibility both on the operator and traveler side. These technologies are designed to deal with recent trends that emphasize more flexibility through the use of shared-ride services and integration of multimodal mobility options.

Determining these smart mobility solutions and analyzing them is a principal area of research for the Intelligent Transportation Systems Lab at MIT and is being tackled through Mobility of the Future, a project that explores and analyzes the future of mobility, funded by the MIT Energy Initiative [2]. Additionally, determining consumer preferences for these smart mobility solutions will be done as a part of "Sustainable Travel Incentives with Prediction, Optimization and Personalization" (Tripod), a system that incentivizes travelers to pursue specific routes, modes of travel, departure times, vehicle types, and driving styles in order to reduce energy use. Specifically, we enhance existing tools in both of these projects to better analyze smart mobility solutions.

1b. Mobility of the Future

The Intelligent Transportation Systems Lab at MIT is conducting research to achieve two goals:

1) Understand and replicate mobility- and energy-related urban dynamics in worldwide metropolitan areas
2) Build an enhanced laboratory to simulate individual traveler reaction and transportation system performance for future scenarios, including mobility services, vehicle and fuel technologies, and energy and environmental policies

To reach these objectives, we propose Mobility of the Future, a project that explores and analyzes the current and future interactions between transportation, energy, and the environment. The project aims to answer questions regarding the impact of emerging technology trends and decarbonization policies. These changes occur on three main levels:

1) **Global**, which focuses on trends, commodity prices, energy demand, and emissions
2) **National**, which focuses on country-specific responses and contribution
3) **Urban**, which focuses on consumer behaviors, mobility patterns, and energy use

As shown in Figure 1, the Mobility of the Future project is at the intersection of consumer choice and public policy, since this is the type of mobility we expect to see in the future. To determine the specifics of this mobility, we focus on five main areas of interest: vehicles, fuels, technology, infrastructure and new modalities, with the last three being the main focus of mobility simulations.

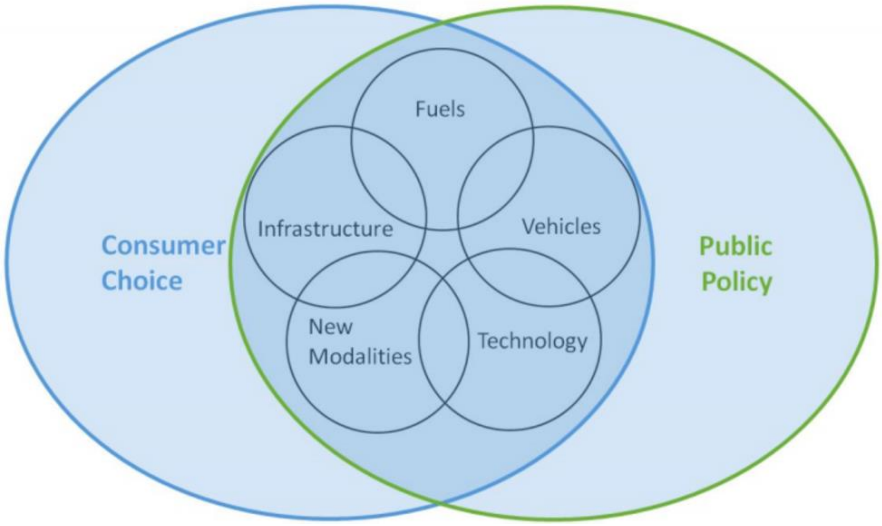


Figure 1: Scope of the Mobility of the Future project

The project strives to find efficient combinations of mobility services and vehicle types to satisfy energy needs, consumer preferences, and environmental commitments in the cities of the future. In order to do this successfully, we collect behavioral data regarding transportation preferences through the Tripod project, described in the next section.

1c. Tripod

MIT, along with the University of Massachusetts at Amherst, is developing and testing its "Sustainable Travel Incentives with Prediction, Optimization and Personalization" (Tripod), a system that incentivizes travelers to pursue specific routes, modes of travel, departure times, vehicles types, and driving styles in order to reduce energy use. Tripod relies on an app-based travel incentive tool designed to influence users' travel choices by offering them real-time information and rewards [3].

The incentives scheme is straightforward: travelers are presented with personalized menus via a smartphone app and are offered "tokens" for a variety of energy-reducing travel options in terms of various parameters. Such options are presented with information to help travelers understand the energy and emissions consequences of their choices. By accepting and performing a specific travel option, travelers earn tokens that depend on the system-wide energy savings they create, encouraging them to consider not only their own energy cost, but also the impact of their choice on the system. Tokens could then be redeemed for services at participating vendors and transportation agencies, or transferred between users.

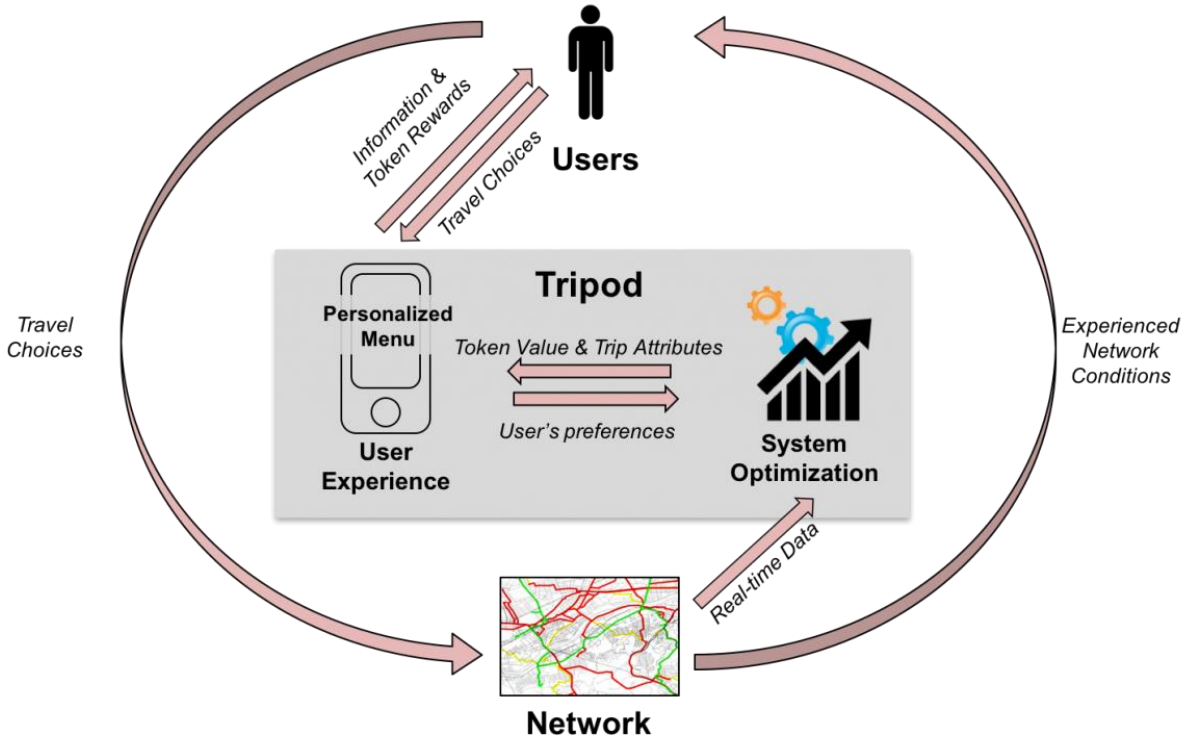


Figure 2: Overview of the framework of the current version of Tripod

As shown in Figure 2, Tripod focuses on determining user data regarding stated preferences for mobility. Obtaining this data using a tool called Future Mobility Sensing (FMS), we are able to use this data in simulations through a tool called SimMobility to obtain smart mobility solutions. These two tools are further described in the next section.

2. Methods

2a. Future Mobility Sensing (FMS)

We use Future Mobility Sensing (FMS) to determine user preferences regarding transportation options. Specifically, FMS is a smartphone and prompted-recall-based integrated activity-travel survey [4]. It uses a combination of a smartphone app, available for Android and iOS, and an online prompted recall survey to collect both demographic and travel data from participants. Data collected from the smartphone app include raw data (e.g., GPS and Wi-Fi data) as well as contextual data (e.g., user history and weather) and are combined to infer what stops were made and what transportation modes were used to determine what the user was doing (Figure 3).

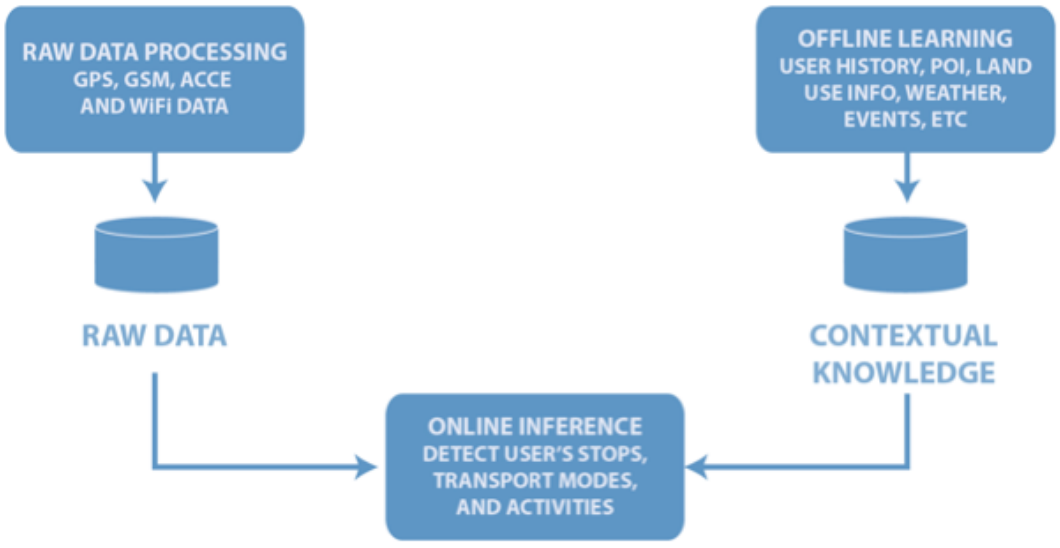


Figure 3: Data collection and analysis using FMS

These data are uploaded to a central server, mapped, analyzed, and made accessible to the participant from the project website, where he or she is asked to provide detailed travel information via a prompted-recall survey. The detailed, accurate data collected by FMS can be used for transportation modeling and urban planning.

We enhance FMS to conduct a stated preference experiment, which asks a user what transportation mode he or she would use when given a multitude of choices. The choices given to the user correspond to various smart mobility solutions and allow us to determine which transportation modes are viable as mobility solutions of the future, which can then be simulated for further analysis. Using a combination of factual and attitudinal variables with choice sets offered in a context-aware basis, we can estimate demand for these different drivetrains and incorporate these preferences into SimMobility, described in the following section.

Specifically, this stated preference experiment will be carried out in Boston, MA, and will consist of a diverse set of people who currently use a wide array of transportation modes with varying frequencies. We design the different alternatives displayed in the experiment, implement the survey, and test and deploy the finished product to be used in the experiment.

2b. SimMobility

Additionally, we have developed a multiscale agent-based traffic simulator called SimMobility [5], which combines behavioral model with regard to land use, transportation, and communication. Specifically, SimMobility consists of three levels of simulation, each of which focuses on a crucial aspect of mobility (Figure 4):

1) **Long-Term**, which focuses on how location choices and land development affect aggregation of people in an urban area and car ownership
2) **Mid-Term**, which focuses on how daily activity and mobility patterns of people affect what trips are made and the accessibility of different areas in an urban environment
3) **Short-Term**, which focuses on how high resolution travel behavior affects performance of different transportation modes and mobility solutions

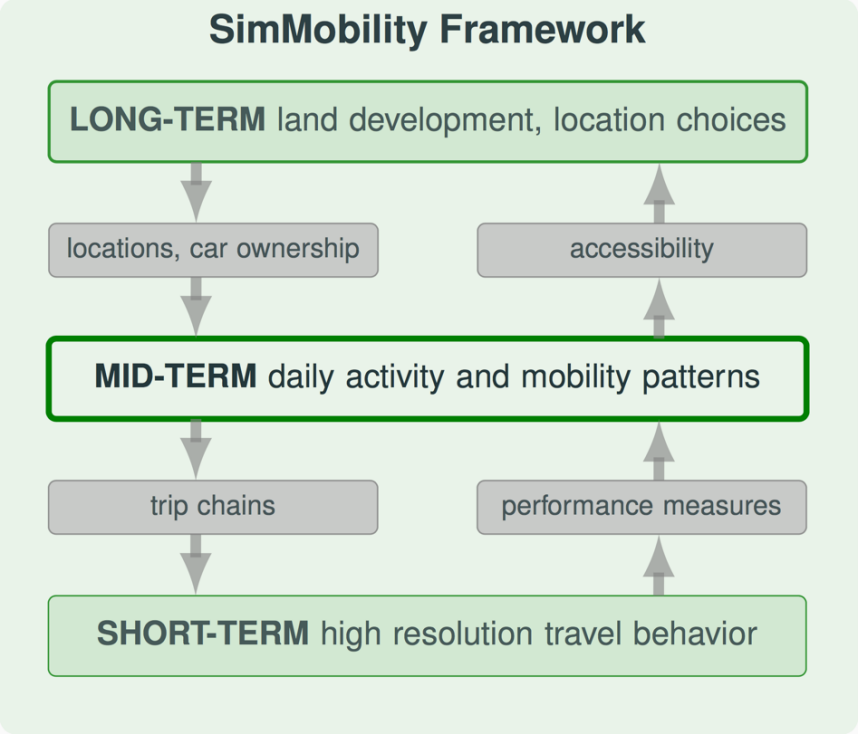


Figure 4: Overview of the framework of the current version of SimMobility

We extend SimMobility in the mid-term and short-term, as these levels are the ones that are significantly impacted by smart mobility solutions. Clarifying, short-term focuses on simulating vehicles and individual movement throughout a given city and mid-term focuses on vehicle supply and demand as well as people's trip preferences.

Specifically, we implement a smart mobility controller in SimMobility, which will be able to handle a variety of alternative technologies, such as autonomous vehicles and on-demand ridesharing services. The controller will take user demand and vehicle fleet supply as inputs, assign supply to demand, and handle price and dispatch optimizations, rerouting vehicles, and cancellations and trips changes. With this controller, SimMobility will be able to simulate user and urban responses to new transportation modes.

Thus, enhancing FMS to collect behavioral data and using this data in SimMobility to simulate various transportation scenarios allows us to effectively predict behavior and obtain viable smart mobility solutions (Figure 5).

Figure 5: Conceptual diagram of FMS and SimMobility inputs for urban scenario analyses

3. Enhancements

3a. Enhancements to Future Mobility Sensing (FMS)

FMS employs a survey that requests stated preferences from the user, which are context-based, meaning that they are based on the user and the trip that was made. Currently, FMS obtains mode choice and departure/arrival time from users in the survey. However, we argue that these two data points are necessary but insufficient to determine preferences for smart mobility solutions – we must introduce not only extra choices in the survey, but also new modes to truly determine context-based stated preferences.

Thus, we begin by including three extra choices for FMS to use in the survey (Figure 6):

1) **Route choice**, e.g., whether a user would rather take a shorter trip but with tolls
2) **Driving style**, e.g., whether a user prefers riding alone or carpooling
3) **Trip cancellation**, e.g., whether a user would take the trip altogether

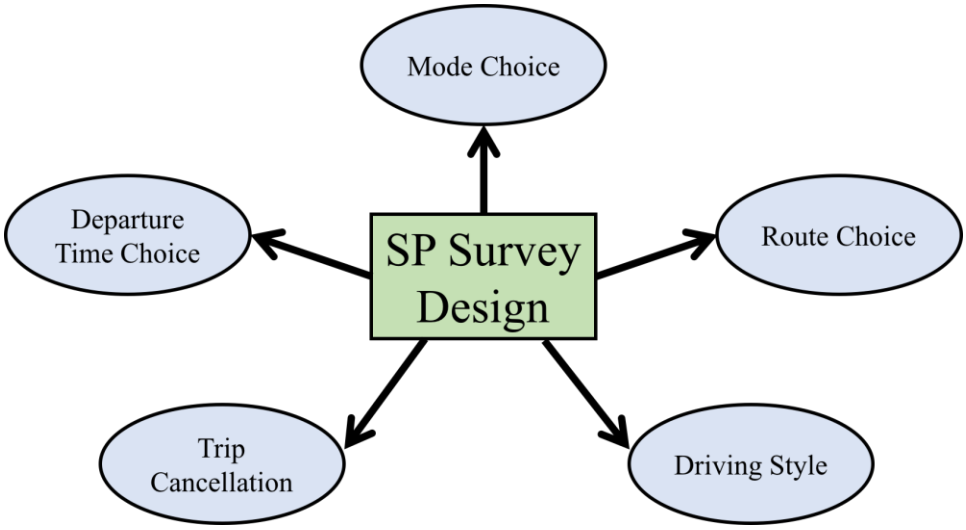


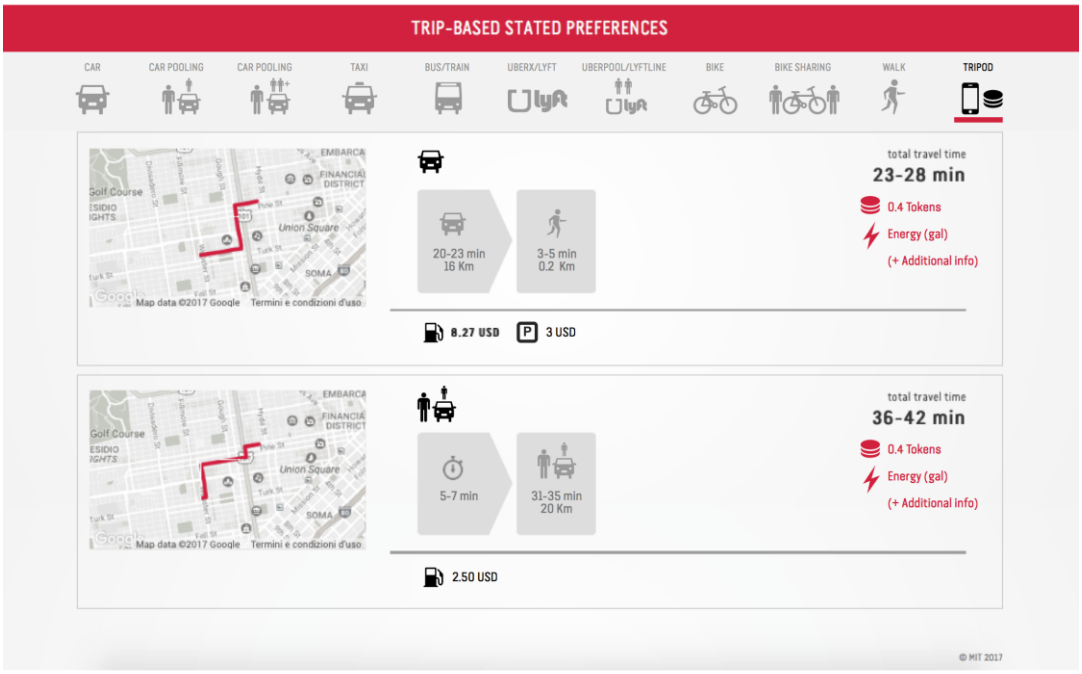
Figure 6: Design of the FMS survey with multiple alternatives

Additionally, the current survey includes a limited number of transportation modes: walking, biking, public transit, driving alone, and taking a taxi. However, with recent disruptions to mobility, we see that there are many more modes that need to be accounted for when enhancing the survey. These include the following, with some being smart mobility options:

1) Bike-sharing (such as Hubway)
2) Carpooling with 1 extra person
3) Carpooling with 2 or more extra people
4) Car-sharing (such as Zipcar)

5) Public transportation and walking
6) Public transportation and driving
7) On-demand services (such as Uber and Lyft)
8) Shared on-demand services (such as uberPOOL and Lyft Line)

Combining the extra user choices along with the new modes, we enhance FMS by providing context-based alternatives to users after they have completed a trip. These alternatives are classified by mode, with modes having multiple alternatives, and the alternatives are randomly generated. As shown in Figure 7, a web interface has been created to allow for these additional modes and choices. Additionally, a user is given “tokens” for choosing certain modes. These tokens serve as incentives for choosing a mode that reduces energy consumption when compared to a more energy-intensive alternative.



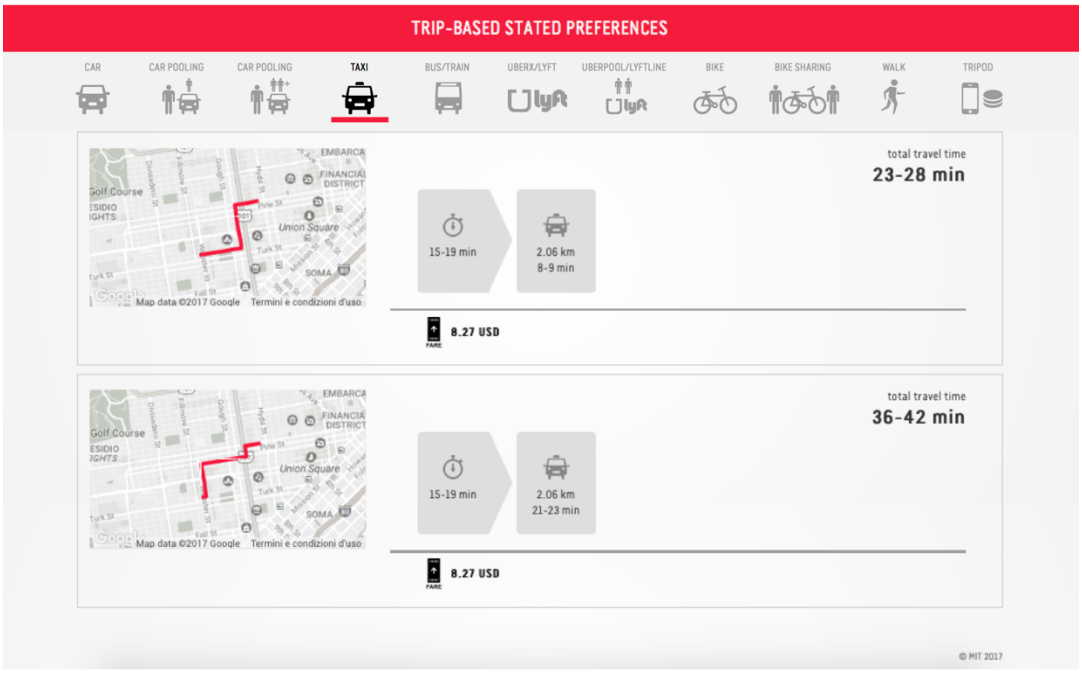


Figure 7: Mockup of alternative modes and alternative choices for each mode in FMS

When designing and implementing these changes, we must appropriately provide alternatives to the user. For example, a user should not be given two options that are very similar: taking an Uber now versus taking an Uber 5 minutes from now. Additionally, a user should never be provided with an alternative that costs less and takes less time than any other alternative. Thus, we must not only employ variability in the code but also checks and balances to give the user options that make sense with his/her trip. Furthermore, the application must properly allocate tokens for each provided alternative. For example, choosing to walk should provide more tokens than taking a car. The number of tokens should be directly related to the amount of energy saved by taking the alternative transportation option.

In terms of implementation, we discuss the availabilities and attributes of modes, the variability between options, and checks to make sure options are consistent with each other. The availabilities of each new mode are as follows.

Mode	**Availability Conditions**
Bike-sharing	Walking distance to bike-sharing station is less than a walking threshold; and total biking distance is less than a biking threshold
Carpooling with 1 person	Always available
Carpooling with 2+ people	Always available
Car-sharing	Walking distance to car-sharing station is less than a walking threshold
Public transportation and walking	Distance from the origin to the nearest station is less than a walking threshold; and distance to the destination from the nearest station is less than a walking threshold
Public transportation and driving	Owning a driver's license; and a minimum of one household automobile
On-demand services	Always available (if the user does not have an account with the specified on-demand service, notify the user that she must do so in order to use this mode)
Shared on-demand services	Always available (if the user does not have an account with the specified on-demand service, notify the user that she must do so in order to use this mode)

With these availabilities, we now describe the attributes of each mode and how they compare to modes that are already implemented in FMS, as follows.

Mode	**Attributes**
Bike-sharing	Similar to the bike mode, but includes access and egress times, along with rental costs.
Carpooling with 1 person	Similar to the drive alone mode, but includes waiting time and a higher travel time. Costs are up to the user to split.
Carpooling with 2+ people	Similar to the drive alone mode, but includes waiting time and a higher travel time than carpooling with 1 person. Costs are up to the user to split.
Car-sharing	Similar to the drive alone mode, but includes access and egress times, along with rental costs and no gas costs.
Public transportation and walking	Up to 3 legs of public transportation and each leg can either be a bus or a train. Costs are fare costs.
Public transportation and driving	Up to 2 legs of public transportation and each leg can either be a bus or a train. Costs are fare costs as well as parking costs.
On-demand services	Similar to the taxi mode, but has lower costs.
Shared on-demand services	Similar to the on-demand services mode, but has a higher travel time and lower costs.

When displaying modes and alternatives to the FMS user, our algorithm must provide certain fields so that the user is able to make an educated choice when choosing an alternative method of transportation. These fields are used to create an easy-to-understand interface for the user as depicted in Figure 7 and are detailed below.

Mode	**Displayed Fields**
Bike-sharing	Travel time, access time, egress time, percentage of the trip that has a bike lane, rental cost, membership cost (if the user is not a member of the service), and tokens
Carpooling with 1 person	Travel time, access time, egress time, parking time, parking cost, toll cost, fuel cost, waiting time, and tokens
Carpooling with 2+ people	Travel time, access time, egress time, parking time, parking cost, toll cost, fuel cost, waiting time, and tokens
Car-sharing	Travel time, access time, egress time, parking time, parking cost, toll cost, rental cost, membership cost (if the user is not a member of the service), and tokens
Public transportation and walking	Travel time, access time, egress time, fare cost, number of transfers, mode for each leg (e.g., bus or train), waiting time, and tokens
Public transportation and driving	Travel time, access time, egress time, parking time, parking cost, toll cost, fuel cost, fare cost, number of transfers, mode for each leg (e.g., bus or train), waiting time, and tokens
On-demand services	Travel time, fare cost, waiting time, tokens
Shared on-demand services	Travel time, fare cost, waiting time, tokens

Now we discuss the variability of alternatives within a given mode:

- *Alternatives for bike-sharing*: There are no alternatives for bike-sharing.
- *Alternatives for carpooling with 1 person*: Similar to the alternatives of driving alone, we have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion) and an alternative of a different route choice.
- *Alternatives for carpooling with 2+ people*: Similar to the alternatives of carpooling with 1 person, we have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion) and an alternative of a different route choice.
- *Alternatives for car-sharing*: Similar to the alternatives of driving alone, we have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion) and an alternative of a different route choice.
- *Alternatives for public transportation and walking*: We have an alternative of a different route choice (e.g., different number/types of legs).
- *Alternatives for public transportation and driving*: Similar to the alternatives of driving alone, we have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion) and an alternative of a different route choice (for both the car route and the public transportation legs).
- *Alternatives for on-demand services*: We have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion).
- *Alternatives for shared on-demand services*: We have an alternative of taking a delayed trip (with an increase in tokens, as this lowers congestion).

Finally, we ensure that all modes and alternatives are consistent with each other by performing the following checks:

1) Driving alone should not be cheaper than taking the bus. If this were the case, there would be no reason to take the bus (from a purely economic standpoint).
2) Taking a taxi should be both faster and cheaper than Uber. If this were the case, there would be no reason to use Uber (from a purely economic standpoint).
3) Walking should not be faster than any other mode. If this were the case, walking would be strictly preferred to any other mode.
4) There should not be excessive differences in time between modes. This indicates to us that an assumption we made about a certain mode is incorrect.

Upon enhancing FMS with these changes, we plan to carry out an FMS pilot with 400 sampled individuals in the Greater Boston Area. The data obtained from this pilot will be used for preference modeling in SimMobility, along with the enhancements described in the next section, to determine the effects of smart mobility on an exemplary city.

3b. Enhancements to SimMobility

The enhancements to SimMobility will focus on a “smart mobility controller” that can handle simulations of smart mobility solutions. This controller must be general enough to handle different modes of transportation, including autonomous vehicles and transportation on-demand, among others. Thus, the smart mobility controller must be aware of the status of all the vehicles it dispatches through efficient and secure communication and must handle all supply and demand changes.

The specific requirements for the smart mobility controller can be summarized as follows:

1) Capability of handling a client’s pick-up request
2) Assignment of smart mobility vehicle to client
3) Creating a route from the client’s pick-up location to his/her final destination
4) Monitoring service status over the vehicle fleet

Specifically, the controller flow is as follows and is detailed in Figure 8. The client requests a ride by providing ride details to the smart mobility controller. The controller determines which vehicles are available in the client’s vicinity and sends the client’s request to those vehicles. Then, these vehicles can choose to accept or reject the client’s request and the smart controller matches a vehicle to a client, with or without input from the client. Furthermore, if a cancellation or error is made during the trip, the controller must be able to handle the situation smoothly and easily.

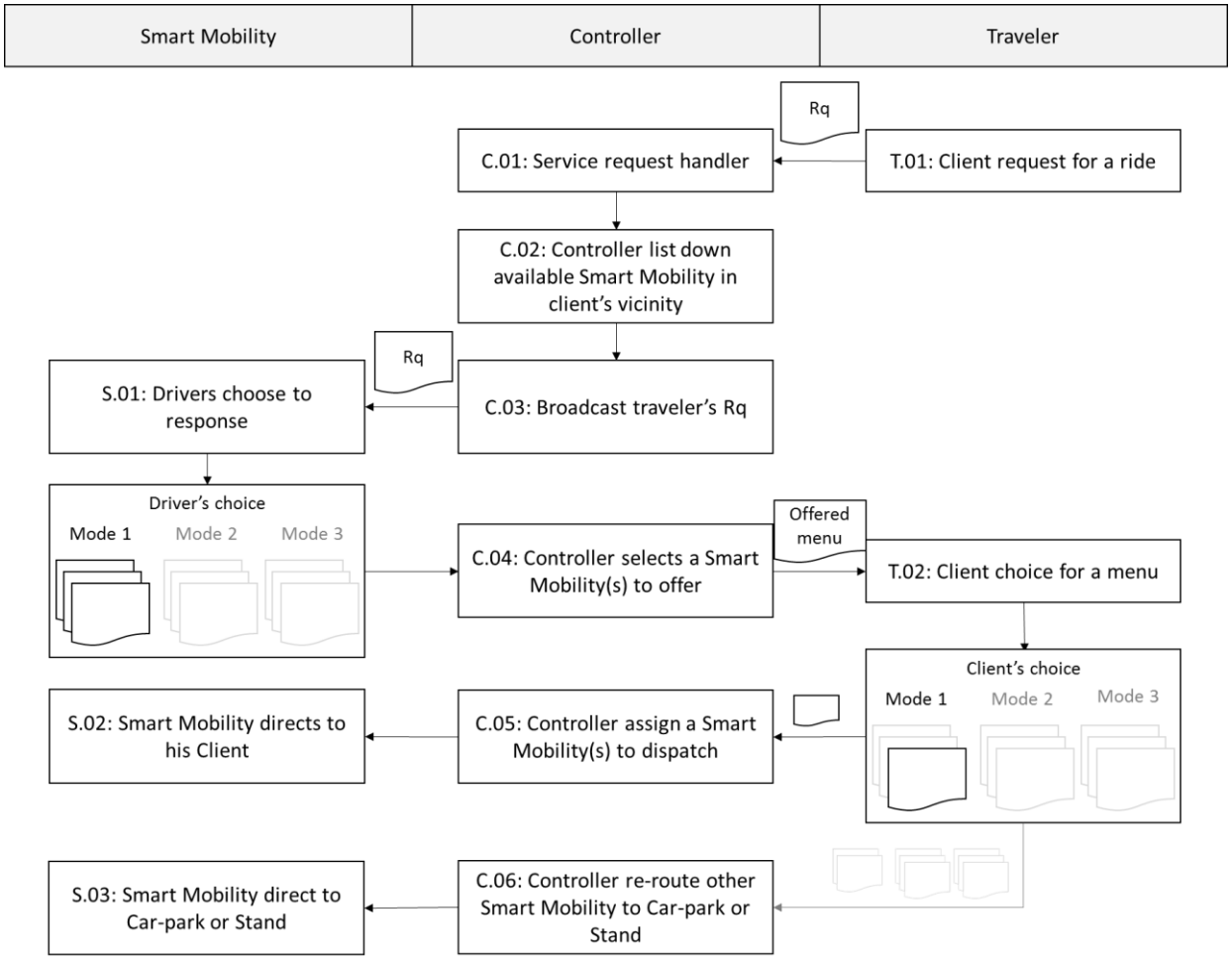


Figure 8: Proposed overall design for smart mobility controller

3b. 1) Taxi Driver Implementation

As an initial implementation of smart mobility controller, we begin by creating a generic "mobility service controller" and testing it with taxi driver agents in SimMobility. We first describe the taxi driver agent that is created in SimMobility, followed by how the generic mobility service controller is able to be extended to handle smart mobility solutions.

The taxi driver agent is modeled as an independent agent that interacts with other agents in SimMobility, specifically, the passengers and the mobility service controller. These taxi drivers can be associated with multiple controllers (e.g., a taxi controller and an on-demand controller). As such, the behavior of the taxi driver is dependent not only on the agent's characteristics, but also the controllers with which the agent is associated. For example, a mobility service controller that a taxi driver is "subscribed" to might not allow for the driver's preferences when assigning passengers, thus eliminating the taxi driver's freedom to choose a client.

The taxi driver is simulated using an event-based framework – that is, the driver has specific states she can be in and there are events that trigger changes in her state. We note that these events can be triggered because of other agents (e.g., mobility service controllers) or because of the taxi driver herself. We define 9 states for the taxi driver, as denoted below:

States	**Description**
DRIVER_START	Initialization of the taxi driver agent
CRUISE	Cruising in search of a passenger
DRIVE_TO_TAXISTAND	Driving to a taxi stand to pick up passengers
QUEUEING_AT_TAXISTAND	Queueing at a taxi stand to pick up passengers
DRIVE_WITH_PASSENGER	Driving with a passenger to a destination
DRIVE_FOR_DRIVER_CHANGE_SHIFT	Driving to a location to switch drivers
DRIVE_FOR_BREAK	Driving to a location for a break
DRIVER_ON_BREAK	Driver on break
DRIVE_ON_CALL	Driver is currently satisfying an on-call request

For these states, there are nine major types of trigger events:

1) Pickup event: This event occurs when a taxi driver picks up a traveler. This pick up can happen on the road or at a taxi stand.
2) Drop-off event: This event occurs when the taxi drops a passenger off at a location.
3) Join queue event: This event occurs when the taxi joins the queue at a taxi stand.
4) Cruising too long event: This event occurs when the taxi driver has been cruising unsuccessfully for too long. In this case, the taxi driver reevaluates her strategy.
5) Queueing too long event: This event occurs when the taxi driver has been queueing unsuccessfully for too long. In this case, the taxi driver reevaluates her strategy.
6) End of shift event: This event occurs when the taxi driver has reached the end of her shift. In this case, the taxi driver must figure out if and where to drop off the car.
7) Break event: This event occurs when the taxi driver decides to take a break.
8) Controller request event: This event occurs when a mobility service controller to which the taxi driver is subscribed sends a pickup request to the taxi driver.
9) Controller information event: This event occurs when a mobility service controller to which the taxi driver is subscribed sends information to the taxi driver regarding redirections (e.g., information about high-demand areas).

Using these defined states and events, we are able to model multiple scenarios, such as pickups, drop-offs, serving clients, queueing at taxi stands, changes in shifts, taking breaks, on-call bookings, trip requests and responses, and reevaluation of strategies. Figure 9 shows how some of these events and states interact with each other. When a taxi driver is available, she can pick up passengers in 3 ways: by controller request, by taxi stand, and by responding to hailing. The taxi driver is then available after dropping off her passengers to pick up more passengers. If the taxi driver is available and has not picked up passengers for a while, she can

then join the queue at a taxi stand. Likewise, if the taxi driver has been in the queue at a taxi stand for too long, she can go back to cruising.

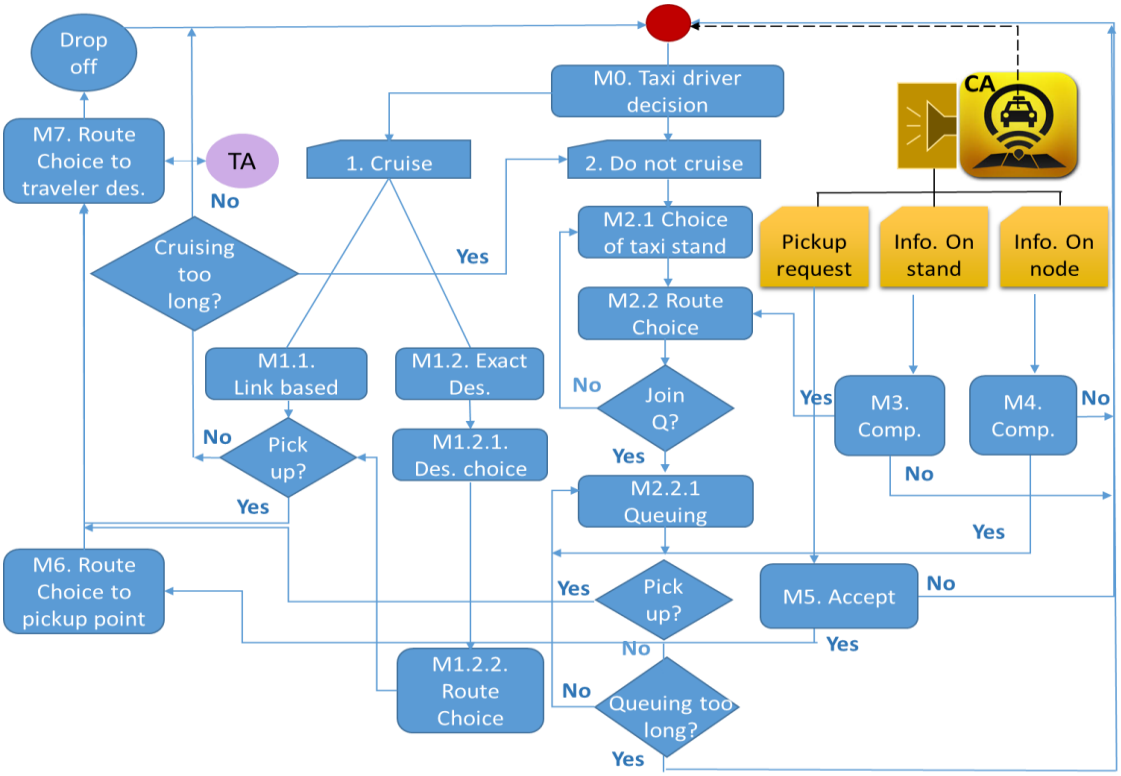


Figure 9: Proposed overall design for the taxi driver agent

3b. 2) Mobility Service Controller Design

We now illustrate how the taxi driver agent interacts with mobility service controllers. Mobility service controllers in SimMobility consist of 3 main components: an initializer, a fleet manager, and a service monitor. The initializer initializes the fleet based on information regarding the network topology, historical mobility demand, fleet size, and corresponding infrastructure. Furthermore, the fleet manager handles servicing the traveler after receiving her travel request. It processes the request, offers options to the traveler, receives a response, and, if necessary, dispatches a vehicle (and can provide routing). The service monitor monitors pickups and drop-

offs, number of subscribed drivers, number of available drivers, the current positions of the drivers, and demand (including request information and spatial distribution). The high-level architecture of a generic mobility service controller is presented in Figure 10.

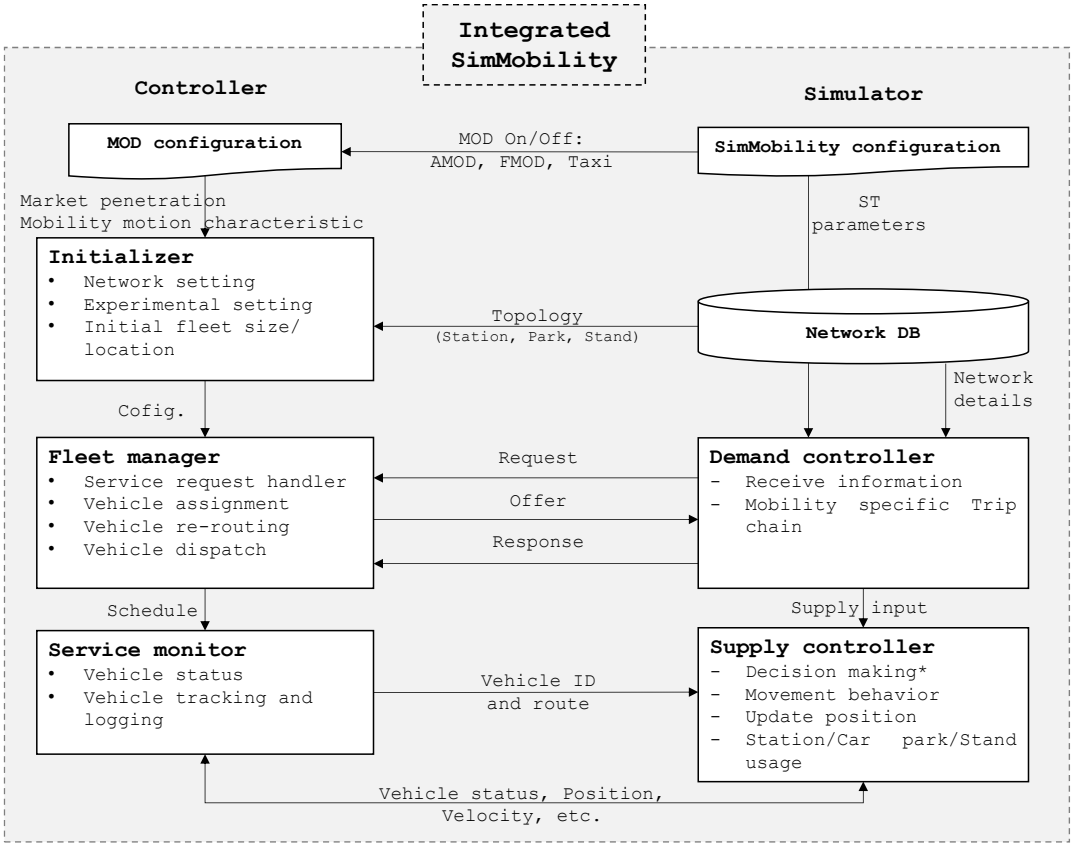


Figure 10: High-level architecture of a generic mobility service controller

We now delve into the details of the implementation of a mobility service controller. We first describe the communication between a controller and users and between a controller and drivers. We do this by following a sequence of messages that occurs during the “normal” operation of a mobility service controller (note: in certain scenarios, some of these messages will not be sent) as illustrated in Figure 11.

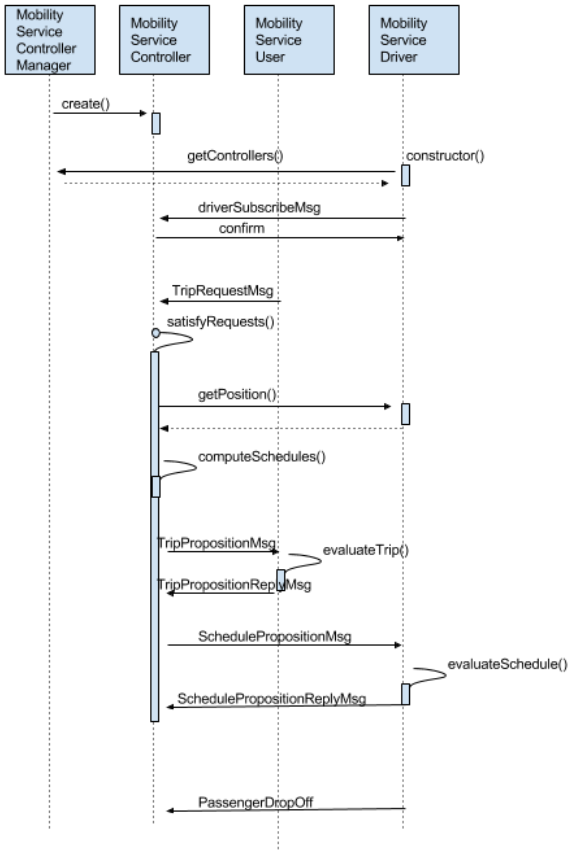


Figure 11: The general sequence of messages sent between a mobility service controller and its users and drivers

The first message sent in the sequence of communication is a DriverSubscribeMessage sent from a mobility service driver (e.g., a taxi driver) to the mobility service controller. This message indicates that a driver wants to subscribe to a particular service (e.g., Uber). As a response, the mobility service controller sends a confirmation, thus notifying the driver that she has successfully subscribed to the service.

Next, when a mobility service user wants to request a trip, she sends a TripRequestMessage to the controller. This message contains the following fields:

- *serviceId*, which indicates which service the user would like to use
- *subServiceIdList*, which indicates which sub-service the user would like to use (e.g., uberPOOL within Uber)
- *originLocationCenter*, which indicates the origin location of the user
- *originLocationRadius*, which indicates the radius from the origin location that the user is willing to commute to for the pickup
- *destLocationCenter*, which indicates the destination location for the user
- *destLocationRadius*, which indicates the radius from the destination location that the user is willing to commute from for the dropoff
- *earliestPickUpTime*, which indicates the earliest time the user is willing to be picked up
- *latestPickUpTime*, which indicates the latest time the user is willing to be picked up
- *earliestDropOffTime*, which indicates the earliest time the user is willing to be dropped off
- *latestDropOffTime*, which indicates the latest time the user is willing to be dropped off
- *sharingPreference*, which indicates the number of people with which the user is willing to share her trip

Then, upon receiving this message, the controller adds the request to its queue of requests. Occasionally, the controller runs *satisfyRequests()*, which attempts to satisfy the requests in the controller's queue. In doing so, the controller calls *getPosition()* on each available driver to optimize matching between available drivers and requests. Finally, the controller runs *computeSchedules()*, which constructs schedules for drivers to pick up passengers.

With hypothetical schedules constructed for drivers, the mobility service controller sends a TripPropositionMessage to each user who made a request, letting her know the trip that the controller can provide. This message contains the following fields:

- *originLocation*, which indicates the pickup location
- *destLocation*, which indicates the dropoff location
- *earliestPickUpTime*, which indicates the earliest time the user will be picked up
- *latestPickUpTime*, which indicates the latest time the user will be picked up
- *earliestDropOffTime*, which indicates the earliest time the user will be dropped off
- *latestDropOffTime*, which indicates the latest time the user will be dropped off
- *sharingPreference*, which indicates the number of people with which the user might share her trip
- *price*, which indicates the price scheme used for the trip, along with an estimated price (or range)

Upon receiving this message, the mobility service user evaluates the proposed trip and sends a TripPropositionReplyMessage back to the controller, indicating whether the user accepts or rejects the trip proposition.

If the mobility service controller receives a positive reply from the user, the controller will send a SchedulePropositionMessage to the respective mobility service driver. This message contains a trip, which is a list of locations paired with which passenger is being picked up or dropped off at each location, and possibly a route (in certain cases, the driver is able to choose her own route and in other cases, the driver must strictly follow the given route).

Once a driver receives this message, the driver evaluates the proposed schedule and sends a SchedulePropositionReplyMessage back to the controller, indicating whether the driver accepts or rejects the schedule proposition.

Finally, once passengers get picked up and dropped off, the mobility service controller keeps track of whether the mobility service drivers it is in charge of are available or unavailable at a given point in time.

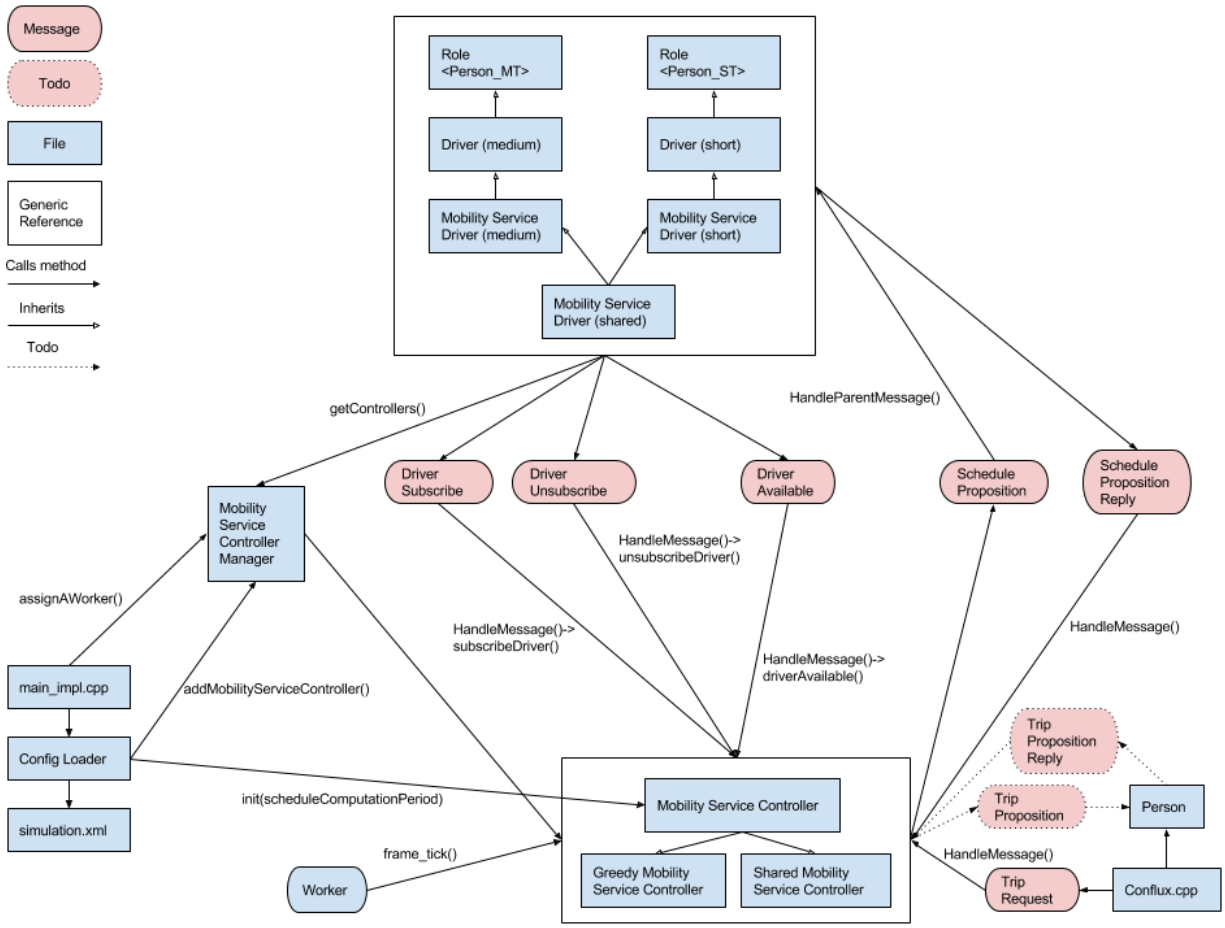


Figure 12: Implementation of mobility service controllers in SimMobility

Following this communication protocol, we now implement a generic mobility service controller class within SimMobility, along with two specific implementations – a greedy mobility service controller and a shared mobility service controller. The descriptions of these implementations will be based on Figure 12.

We begin by defining a mobility service controller manager. This entity keeps track of all instantiated controllers in the simulation and updates them as necessary. Specifically, the mobility service controller manager has a method *addMobilityServiceController()* that adds an instantiated mobility service controller to its list of controllers. This list of controllers gets updated every frame tick (since the manager is assigned a worker), and each mobility service controller has a parameter that determines how often it services requests. Furthermore, mobility service drivers and mobility service users can access the list of service controllers using *getControllers()*. When instantiating mobility service controllers, the configuration loader in SimMobility reads “simulation.xml”, which contains parameters for each controller.

In our SimMobility tests, we use taxi drivers in mid-term and short-term as our mobility service drivers. As such, we have a generic abstract mobility service driver class which is extended by these taxi drivers. This is also the case for mobility service controllers – we have a generic mobility service controller which is extended by specific service controllers.

Mobility service controllers handle messages using *HandleMessage()*. This method calls specific controller methods based on the message: driver subscribe and unsubscribe messages call *subscribeDriver()* and *unsubscribeDriver()*, respectively. Likewise, mobility service drivers handle messages using *HandleParentMessage()*.

Mobility service drivers can be subscribed to multiple mobility service controllers – in SimMobility, we have taxi drivers subscribed to a subset of on-hail controllers and on-demand controllers. We now describe the general states that a taxi driver in SimMobility can be in when subscribed to these controllers, as depicted in Figure 13.

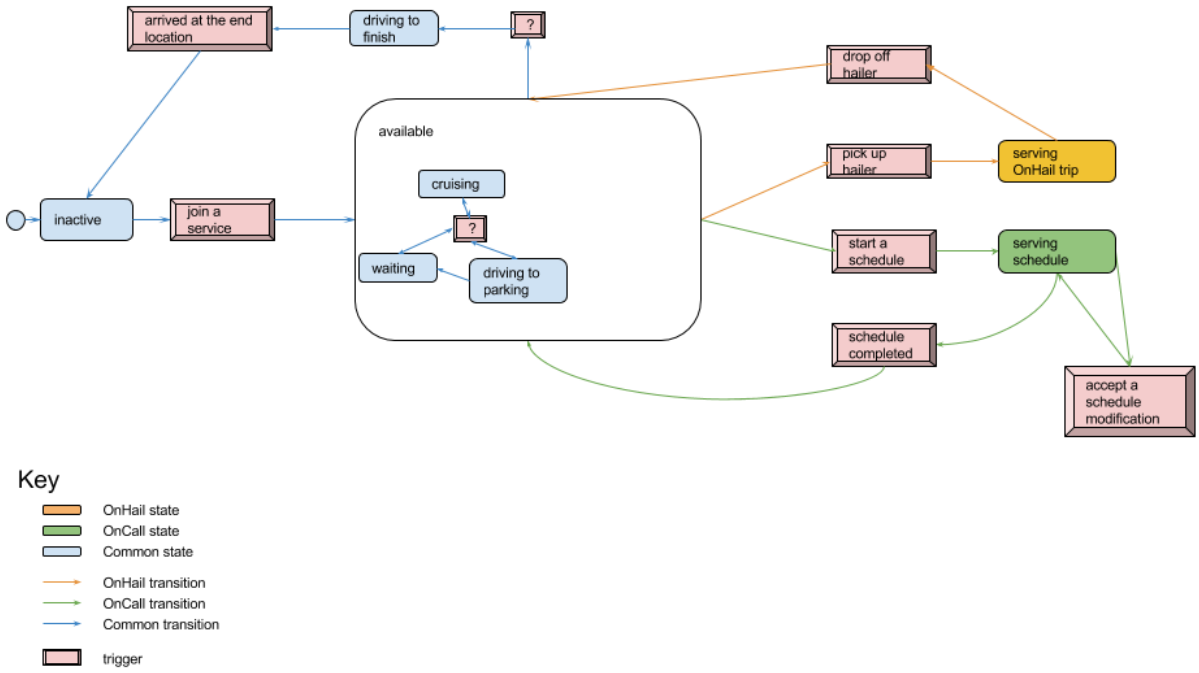


Figure 13: States of the taxi driver agent in SimMobility when subscribed to only on-hail controllers and on-demand controllers

The taxi driver begins in an inactive state. She then joins a service and enters an “available” state. In this state, the driver can either be cruising, waiting for passengers (e.g., at a taxi stand), or driving to parking (e.g., to take a break). In any of these cases, the driver can service a trip in two ways: the driver can either pick up a hailing passenger, or the driver can agree to a schedule proposition provided by an on-demand mobility service controller. In the first case, the driver simply picks up a hailing passenger and drops her off at the relevant destination – there are no messages or instructions from the on-hail mobility service controller telling the driver what to do. In the second case, the driver receives and responds affirmatively to a schedule proposition provided by the on-demand mobility service controller. Then, the driver picks up the relevant passengers and drops them off at their respective drop-off locations. Finally, the taxi driver becomes inactive again when she unsubscribes from all her controllers and drives to a location to end her shift.

3b. 3) Mobility Service Controller Implementation

We now describe the implementation details of the two smart mobility service controllers tested in SimMobility – a greedy mobility service controller and a shared mobility service controller. Figure 14 illustrates the code hierarchy of these classes and their relationships to the mobility service controller manager, mobility service drivers, mobility service users, and system-wide messaging.

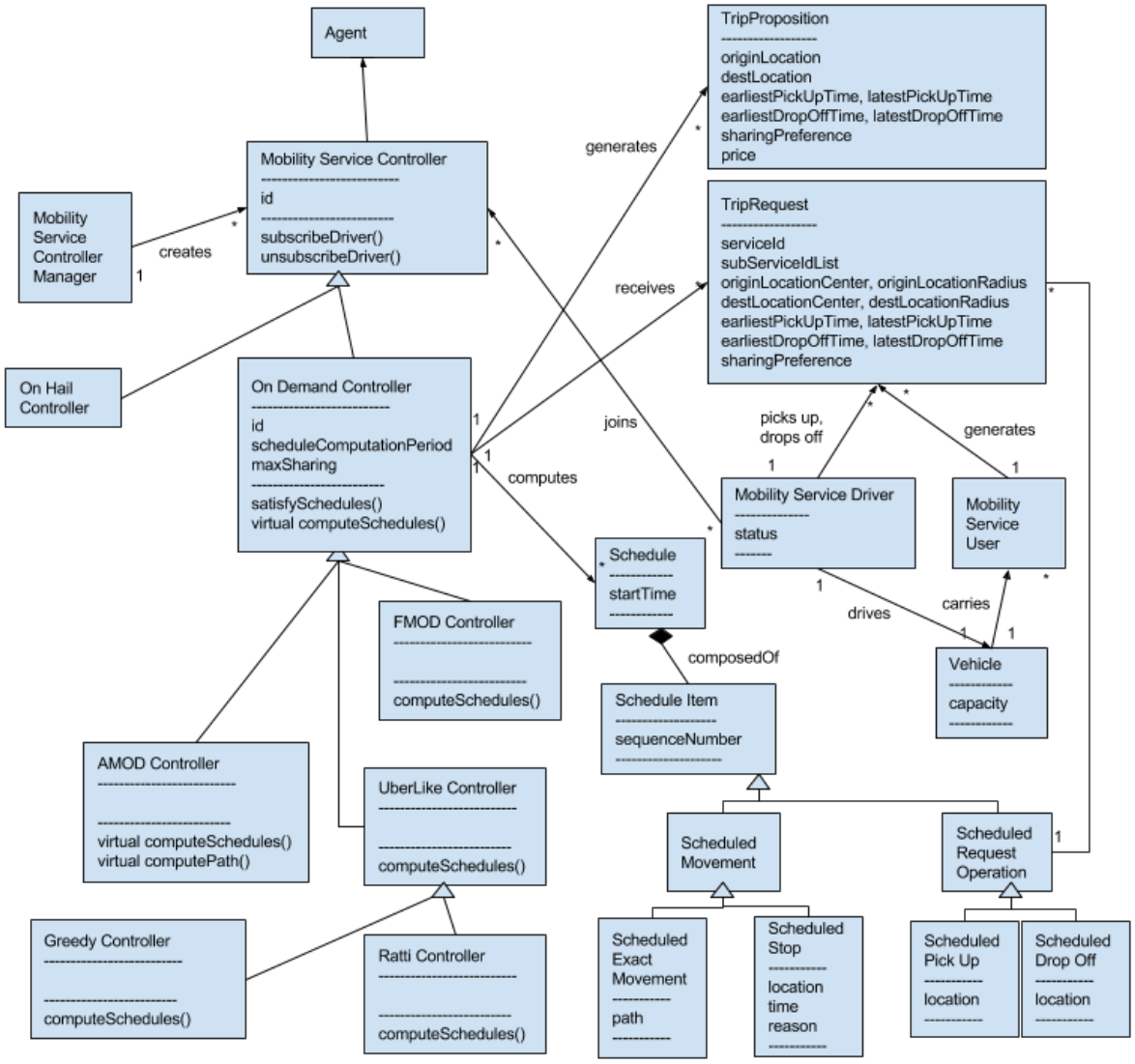


Figure 14: Code hierarchy of the mobility service controller class, its children, and its relationship to the rest of SimMobility

The greedy controller takes a list of trip requests and matches them to available mobility service drivers by finding the closest available driver to each trip's origin location. This controller can be compared to an Uber-like controller and can serve as an autonomous Uber-like controller if

mobility service drivers must follow the controller's instructions (rather than allowing them to reject propositions).

The shared mobility service controller takes a list of trip requests and determines whether trips can be shared while satisfying the relevant users' trip conditions (e.g., latest pickup time and latest drop-off time). Specifically, we do this by using a maximum matching algorithm proposed by Paolo Santi [6]. We begin by creating a graph, where each node is a trip request that is waiting to be processed. Then, we add an undirected, unweighted edge from one node to another if the two trip requests can be shared, that is, if both passengers' trips can overlap (they are in same vehicle at the same time) while satisfying both trips' conditions.

Finally, we perform maximum matching on the graph, which allows us to create as many shared trips as possible. Specifically, a matching M of a graph $G=(V,E)$ is a set of pairwise non-adjacent edges (no two edges share a common vertex) and a maximum matching is a matching that contains the largest possible number of edges. We use Jack Edmonds' blossom algorithm [7] to perform maximum matching, which is an efficient choice, since it runs in polynomial time and is already implemented using the Boost library in C++ [8]. The construction of shared trips is depicted in Figure 15, and we see that this algorithm can be extended to include more than two overlapping trips.

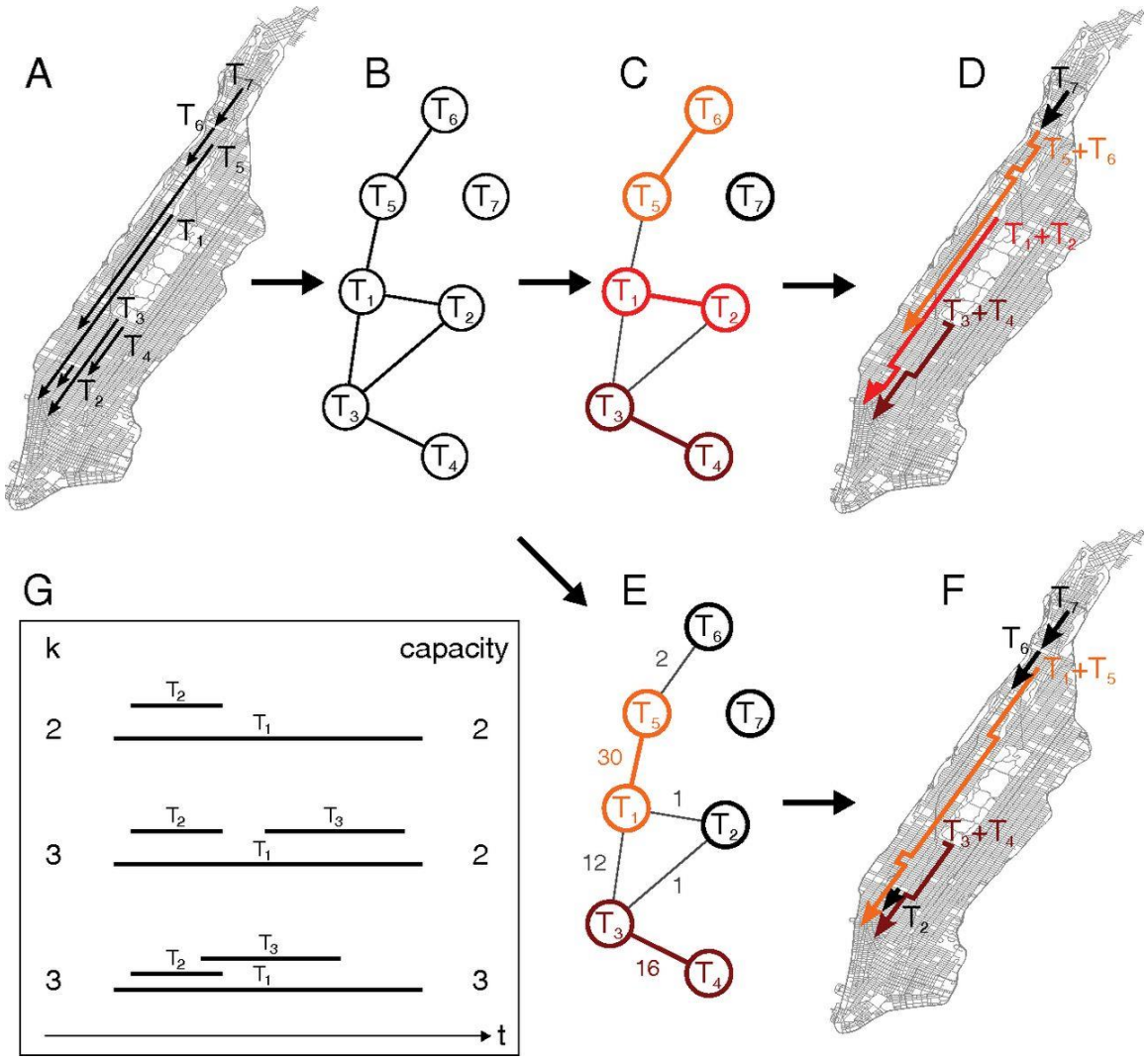


Figure 15: An example of computing trip shareability using Santi's maximum matching framework [6]

When implementing both controllers in SimMobility, we create a generic abstract mobility service controller class that extends Agent (since it is a source and destination of communication and is updated every frame tick). This abstract class contains methods that are common to all its children, including *subscribeDriver()*, which adds the driver that sends a DriverSubscribeMessage to its list of drivers, and *unsubscribeDriver()*, which removes the driver

that sends a DriverUnsubscribeMessage from its list of drivers. This generic class is what the mobility service manager keeps track of, as it does not need to know the specific ways the generic class' children override its methods. For the same reason, mobility service drivers and mobility service users send and receive messages from the generic class, rather than specific implementations of the class.

From this abstract mobility service controller class, we implement two controllers – an on-hail controller, which does not override or implement any methods, since it does not send instructions to mobility service drivers, and an on-demand controller. The on-demand controller is also an abstract class and represents smart mobility controllers that can take on-demand requests from users. Specifically, we introduce three types of on-demand controllers: an autonomous mobility on-demand (AMOD) controller, a flexible mobility on-demand (FMOD) controller, and an Uber-like controller. The principal difference between the AMOD controller and the others is that in our implementation in SimMobility, the AMOD controllers send a schedule and a route to the driver (in this case, an autonomous vehicle) and the driver always accepts and always follows the route given.

As mentioned before, we focus on implementing two types of Uber-like controllers: a greedy controller and a shared controller. We notice that most methods between these controllers are common, and the only method that is changed between them is *computeSchedules()*, since the greedy controller assigns requests to the nearest available drivers, and the shared controller performs maximum matching to determine which trips can be shared. The common methods are implemented in the generic mobility service controller class and are not overridden by these controllers.

3b. 4) Smart Mobility Results

We test our two controllers in SimMobility for correctness, robustness, and scalability. These tests are performed on an exemplary virtual city with no modes of transportation other than taxis and walking, and no types of passengers other than taxi-hailers and on-demand users. In these tests, there are 235 taxi vehicles and drivers, and a variable set of hailing and on-demand passengers.

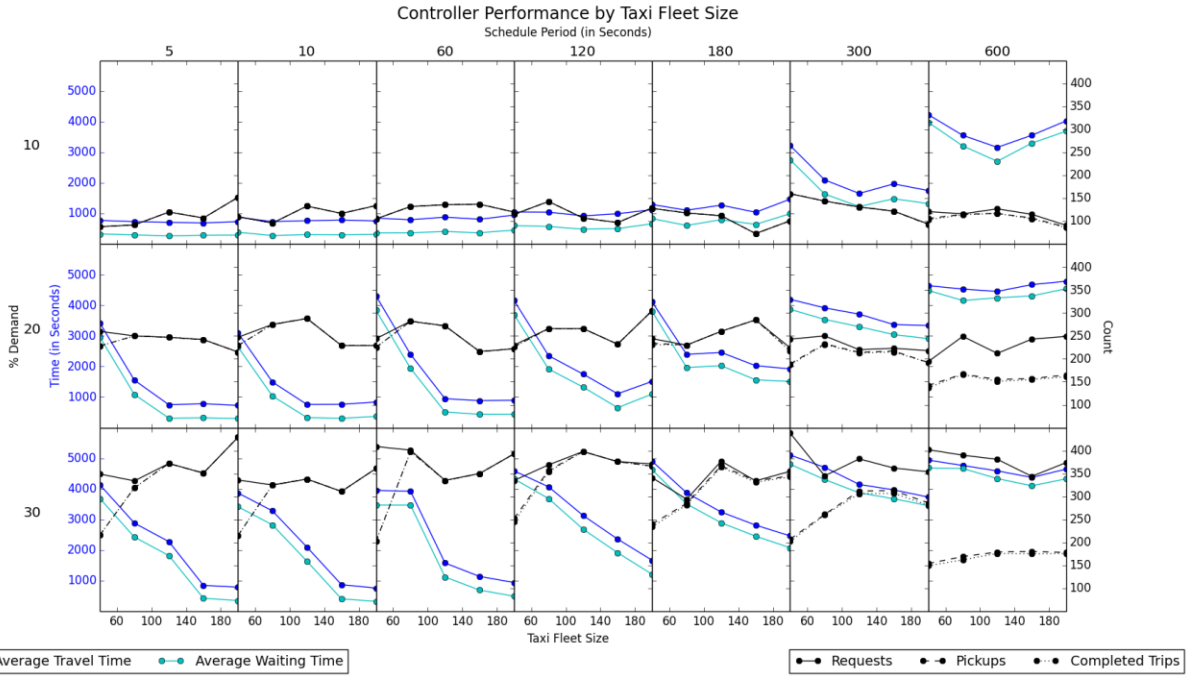


Figure 15: Performance of the greedy mobility service controller with respect to taxi fleet size for a simulation time of 1.5 hours in an exemplary city with only mobility service controller users and drivers. Each column represents the value of the schedule computation period (which is how frequently the controller services requests). Each row represents the percentage of the total trip-requesting demand in the simulation that is serviced by the controller. For each graph, the x-axis is the taxi fleet size used to generate the data, and the y-axis for the colored lines (average travel time and average waiting time) is time in seconds, and the y-axis for the black lines (number of requests, pickups, and completed trips) is a count.

First, we begin by testing the greedy mobility service controller. As shown in Figure 15, we test the controller with various schedule computation period values (which is how frequently the controller services requests), demand, and taxi fleet sizes. We first analyze the plots by row – when the schedule computation period increases, we see that the average travel time and average waiting time increases. This makes sense, since increasing the time between servicing requests causes users to wait longer before being picked up by a driver. Likewise, we see that increasing the schedule computation period can also cause fewer pickups and fewer completed trips (as in the row with 30% demand). This is because increasing the time between servicing requests can cause some requests to not be satisfied by the end of the simulation run.

We now analyze the plots by column – when demand increases, we see that the number of requests increases, as expected. Furthermore, we see that the number of pickups and completed trips also increases until a threshold is reached (as in the column with a schedule computation period of 300 seconds). This threshold is a combination of a limited taxi fleet size and the limitations imposed by a high schedule computation period. We also see that an increase in demand can result in an increase in average travel time and average waiting time after a certain threshold. This threshold is again a combination of a limited taxi fleet size and the limitations imposed by a high schedule computation period.

Finally, we analyze how fleet size affects the performance of the controller. Specifically, we see that the average travel time and average waiting time decreases with an increase in taxi fleet size (accounting for variance). This makes sense, since a larger fleet size can service more requests – fewer mobility service users have to wait for a mobility service driver to become available when there are more drivers in the simulation. Furthermore, we see that a change in taxi fleet size has no effect on the number of requests (accounting for variance). This makes sense, since the number of requests only depends on demand. Conversely, we see that an

increase in taxi fleet size does increase the number of pickups and completed trips, since a higher fleet size can service more requests, up to the total number of requests.

In terms of scalability, we check the performance of using one greedy controller in the simulation and compare it with the performance of the simulation without any controllers. Doing this, we see that the controller adds a ~2x runtime overhead, which is reasonable given the nature of processing and matching requests with available mobility service drivers. Therefore, we conclude that our implementation of a greedy mobility service controller is scalable.

We then test the shared mobility service controller. Immediately, we observe that creating a shareability graph for a list of requests and performing maximum matching on this graph results in a significant performance overhead in the simulation's runtime (at times, the shared mobility controller increased the simulation runtime by 250x). Therefore, since we determine that the current implementation of the shared mobility service controller is not scalable, we choose to not analyze this controller further. Going forward, we aim to replace the current shareability algorithm with one that uses heuristics to create a controller that is more scalable, sacrificing maximizing the number of shared trips in the process.

Once we conduct the previously mentioned FMS pilot in the Greater Boston Area, we can use the data obtained from FMS to update supply and demand parameters in SimMobility to simulate smart mobility solutions in the Boston area. Specifically, we will use the preference data obtained from the FMS pilot to determine client choice as depicted in T.02 in Figure 8. This will allow us to simulate smart mobility options as realistically as possible, since we already have a simulation of Boston's Central Business District in SimMobility, as shown in Figure 16. With these input parameters, we will be able to simulate and collect output from viable smart mobility solutions, which can then be used to analyze the effects, both positive and negative, of smart mobility's role in the future.

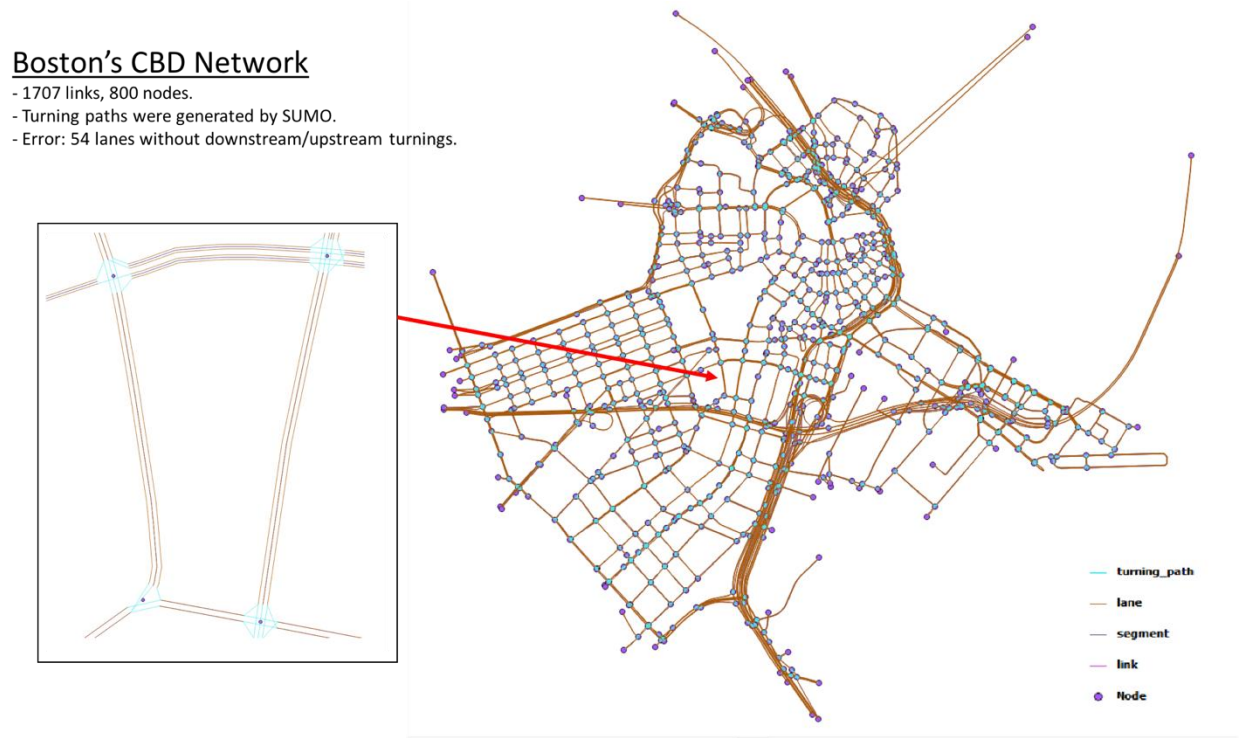


Figure 16: Implementation of Boston's Central Business District in SimMobility

4. Conclusion

In this project, we aim to analyze smart mobility solutions through the Mobility of the Future and Tripod projects. Specifically, this research focuses on two enhancements to tools used in these projects:

1) Using FMS to collect data and creating predictive models using this data to obtain travel preferences in the presence of future fuel/technologies in the transportation sector
2) Extending SimMobility to accommodate future mobility specifications in the transportation environment

These enhancements are crucial for researchers to accurately analyze the effects of new, disruptive transportation solutions. Likewise, they are vital for mobility-service providers to determine how different smart mobility solutions affect their business models, so they can prepare and react accordingly. Overall, collecting behavioral data and simulating mobility solutions are highly relevant and impactful in an industry that affects everyday life and is constantly changing.

5. Acknowledgements

The author would like to thank Carlos Azevedo, Andrea Araldo, Arun Akkinepally, Mazen Danaf, Jimi Oke, and Bilge Atasoy for advice and direction on the work described in this document. Additionally, the author would like to thank Kakali Basak, Huai Peng Zhang, and Neeraj Deshmukh for assistance on the software development described in this document.

6. References

[1] Benevolo, Clara, Renata Paola Dameri, and Beatrice D'Auria. "Smart mobility in smart city." *Empowering Organizations*. Springer International Publishing, 2016. 13-28.

[2] "MIT Energy Initiative." 13 December 2016. <https://energy.mit.edu>

[3] Zhao, Fang, et al. "Exploratory analysis of a smartphone-based travel survey in Singapore." *Transportation Research Record: Journal of the Transportation Research Board* 2.2494 (2015): 45-56. Web.

[4] Cottrill, Caitlin, et al. "Future mobility survey: Experience in developing a smartphone-based travel survey in Singapore." *Transportation Research Record: Journal of the Transportation Research Board* 2354 (2013): 59-67. Web.

[5] Adnan, Muhammad, et al. "SimMobility: A Multi-scale Integrated Agent-Based Simulation Platform." (2016). Web.

[6] Santi, Paolo, et al. "Quantifying the benefits of vehicle pooling with shareability networks." *Proceedings of the National Academy of Sciences* 111.37 (2014): 13290-13294.

[7] Edmonds, Jack. "Maximum matching and a polyhedron with 0, 1-vertices." *Journal of Research of the National Bureau of Standards B* 69.125-130 (1965): 55-56.

[8] Siek, Jeremy G., Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.