# ModeChooser

## Program Documentation

Developed by

Bing Mei

Triangle Regional Model Service Bureau
ITRE @ NCSU

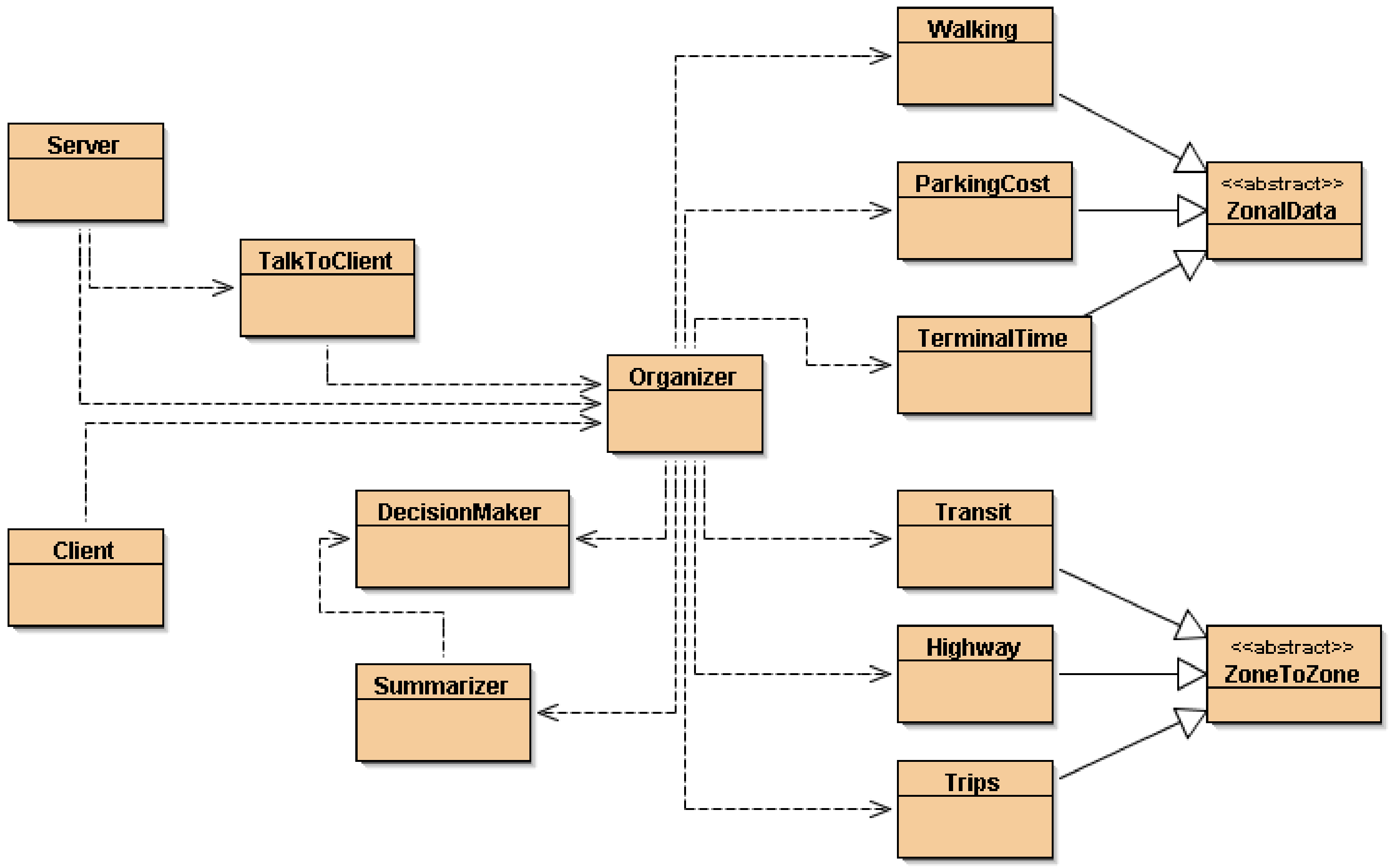
Raleigh, NC

## Table of Contents

# INTRODUCTION

The ModeChooser application is developed using Java for experimenting in the Triangle Regional Travel Demand Model (TRM). Basically, ModeChooser simulates urban travelers' behavior in traffic mode choice, based on a few factors such as personal and household socioeconomic characteristics and highway and transit system attributes. The ModeChooser application utilizes a distributed computing algorithm to make use of multiple computers in the network to compute simultaneously to improve computational efficiency.

## CLASS CHART OF MODECHOOSER



# CLASS DESCRIPTION

## Organizer

1) Class responsibilities:

- construct a GUI where the user is allowed to specify the directory of the model scenario to be run and then start the model run by clicking the "OK" button.
- define the input and output files with the directory specified by the user; input and output files are one for each combination of trip purposes (5) and times of day (2).
- loop thru all the trip purposes and times of day – run the model for each of them and write model results from Summarizer to output files
- respond to the server for data taking and sending, once requested.
- run the model for the client, once requested by the client, in a different way from the one described above (which is for the server). The method for model run on the client side is titled "runModelAsHelper(File dir, int[] statusToRun)"

2) Main methods:

- public void go(): construct a GUI for the user to specify model scenarios and execute the model.
- private void setInputFiles(File dir): define on the GUI input files from the directory chosen by the user.
- private void setOutputFiles(File dir): define on the GUI output files from the directory chosen by the user.
- private boolean runModel(): load all input data and execute the model for a certain trip purpose and a certain time period as organized by the Organizer. An instance of class Summarizer and many instances of class DecisionMaker are created by this method. This method returns true if model execution is successful; otherwise false.

The following methods are involved in the interaction between Server and Client for distributed computing.

- public void runModelAsHelper(File dir, int[] statusToRun): this method is only used by the Client. After the Client gets the directory of the model scenario that is running on the Server, it uses the directory to define all the input and output files on the Client side, and starts the model run for a new trip purpose and/or a new time of day.
- public File getDir(): this method is only used by the Server to get the directory of the model scenario that is currently running on the Server side. It will be passed to the Client by TalkToClient.
- public int[] getStatus(): get the GUI status on the Server side for sending an new status to the Client.
- public void setStatus(int[] newStatus): this method is only used by the Server to update the status on its own side, after communicating with the Client.

The following methods load input data.

- private boolean loadParameter(): load all the parameters used by the mode choice models;
- private boolean loadWalk(): load walk access distances to transit buffers for each TAZ;
- private boolean loadParkingCost(): load auto parking cost (in $)for each TAZ, if there is any;
- private boolean loadTerminalTime(): load auto access and egress terminal time (in minutes) for each TAZ;
- private boolean loadLocalSkim(int originZone): load local bus travel times and fares for a specified origin TAZ;
- private boolean loadExpressSkim(int originZone) : load express bus travel times and fares for a specified origin TAZ;
- private boolean loadDrive1Skim(int originZone) : load drive-alone auto travel time and distance for a specified origin TAZ;
- private boolean loadDrive2Skim(int originZone): load 2-person shared-ride auto travel time and distance for a specified origin TAZ;
- private boolean loadDrive3Skim(int originZone): load 3+ person shared-ride auto travel time and distance for a specified origin TAZ; and
- public boolean loadTrips(int originZone): load trips by mode (auto drive-alone, auto 2-person shared-ride, 3+ person shared-ride, local bus, and express bus) for a specified origin TAZ.

## DecisionMaker

1) Class responsibilities:

- estimate the probability of a traveler choosing each of the available travel modes from a specified origin zone to a destination zone, based on his/her socioeconomic characteristics (SE strata) and the traveling characteristics (such as travel time, transit fare, parking cost, travel distance, etc.)

2) Main methods:

- public void go(double auto_market, double[] coeff, double[][] econst, double[] lbusSkim, double[] ebusSkim, double[] drv1Skim, double[] drv2Skim, double[] drv3Skim, double[] zPark, double[] zWalko, double[] zWalkd, double[] zTermiTo, double[] zTermiTd): this method takes all the input data passed over from the Organizer and computes probabilities of a traveler choosing each of the available traffic modes for his/her trip for a OD pair. The arguments are:
    - auto_market: percent of auto trips in total number of trips
    - coeff: coefficients used by the mode choice model
    - econst: bias constants used by the mode choice model
    - lbusSkim: local bus skim data as loaded by the Organizer
    - ebusSkim: express bus skim data as loaded by the Organizer
    - drv1Skim: drive-alone auto skim data as loaded by the Organizer
    - drv2Skim: 2-person shared-ride auto skim data as loaded by the Organizer
    - drv3Skim: 3+ person shared-ride auto skim data as loaded by the Organizer
    - zPark: zonal parking cost as loaded by the Organizer
    - zWalko: walk access time for transit modes as loaded by the Organizer
    - zWalkd: walk egress time for transit modes as loaded by the Organizer
    - zTermiTo: walk access time for auto modes as loaded by the Organizer
    - zTermiTd: walk egress time for auto modes as loaded by the Organizer

The following methods are getters that get probability values from DecisionMaker.

- public double[][] getprobLB(): get the probability for local bus;
- public double[][] getprobEB(): get the probability for express bus;
- public double[][] getprobRL(): get the probability for rail;
- public double[][] getprobTRN(): get the probability for transit as a whole;
- public double[][] getprobSR(): get the probability for the shared ride mode;
- public double[][] getprobAUT(): get the probability for the auto mode as a whole;
- public double[][] getprobNM(): get the probability for non-motorized modes.

## Summarizer

1) Class responsibilities:

- compute zone-to-zone number of trips for each travel mode and SE stratum, based on the zone-to-zone total number of trips passed from Organizer and probabilities calculated by DecisionMaker
- summarize zone-to-zone trips for each travel mode over SE starta
- summarize regional trip totals for each travel mode and each SE stratum
- respond to requests from other classes thru getters and setters

2) Main methods:

- public void computeZZtrips(double[] market, double[] totalTrips, DecisionMaker dMaker): this method takes input trip data from Organizer, gets probabilities from DecisionMaker, and then computes zone-to-zone trips by traffic mode and by SE stratum. It also summarizes trips.

The following methods are getters and setters that are used by Organizer to get data from Summarizer to output to files on hard drive and reset data to zeros once the model runs for a new set of trip purpose and/or time period.

- public double[] getZZTrips(): get zone-to-zone trips by travel mode;
- public double[][] getRegionalTrips(): get regional trip totals by travel mode;
- public void setZZTrips(): set zone-to-zone trips by travel mode;
- public void setRegionalTrips(): set regional trip totals by travel mode.

## Server

1) Class responsibilities:

- start a thread for talking to the client with the job defined in TalkToClient; and
- start the main thread, which runs the model on the server side

2) Main methods:

- public static void main(String[] args):  start the execution of the program on the server.
- public void go():   start a thread for talking to the client with the job defined in TalkToClient; and starts the main thread, which runs the model on the server side.

## TalkToClient

1) Class responsibilities:

- receive the status of model run from the client side, i.e., the trip purpose and the time of day that the client has just finished and passed over.
- check to see if there is any new tasks for the client to "help" with.  If yes, pass the task to the client and update the status on the server side.  If no, pass (99, 99) to the client.

2) Main methods:

- public void go():  set up networking & make the server application listen for client requests on port 4410;  get model run status from the client; get model run status from the server; and return the client a new status and update the status on the server.

Functionality of the following getters and setters is described as follows.

- private int[] getServerStatus():  get server status;
- private void setServerStatus(int[] s):  set server status;
- private File getServerDir():  get the directory on the server where model is executed.

## Client

1) Class responsibilities:

- establish connection with the server
- tell the server which trip purpose (pi) and time of day (tj) itself has just finished
- ask the server for a new task (i.e., a new set of values of pi and tj) and therefore gets input/indication from server
- if the new task is valid (i.e., pi <= 4 & tj <= 1), start the model run on the client side
- loop through the four items above until getting pi = 99 or tj = 99.

2) Main methods:

- public static void main(String[] args):  start the execution of the program on the client.
- public void go():  connect to the server, send the current model run status on the client itself to the server, receive a new status from the server, and run the model if the status is valid. Otherwise, do nothing.

## ZonalData

1) Class responsibilities:

- serve as the superclass to classes Walking, ParkingCost, and TerminalTime. It reads zonal data from a text file that is passed over from each subclass of it.
- format of input text files:
  - 1st column: zone *i*
  - Starting from 2nd column: each column stores an attribute

2) Main methods:

- public boolean readData(File file, int nfields): read zonal data from a text file that is passed over from each subclass of it, and check for errors. It returns true if data is read successfully; otherwise, return false;
- public int[] getID(): get zone IDs;
- public double[][] getAttrs(): get zonal attributes.

## Walking

1) Class responsibilities:

- extend class ZonalData and read zonal walk-related data from the specified walk file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get data from the walking data file by calling the readData() method in superclass ZonalData. It returns true if data is read successfully; otherwise, return false.

## ParkingCost

1) Class responsibilities:

- extend class ZonalData and read zonal parking cost data from the specified parking cost file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get data from the parking cost file by calling the readData() method in its superclass ZonalData. It returns true if data is read successfully; otherwise, return false.

## TerminalTime

1) Class responsibilities:

- extend class ZonalData and read zonal terminal time data from the specified terminal time file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get data from the terminal time file by calling the readData() method in its superclass ZonalData. It returns true if data is read successfully; otherwise, return false.

## ZoneToZone

1) Class responsibilities:

- serve as the superclass to classes Transit, Highway, and Trips. It reads zonal data from a text file that is passed over from each subclass of it.
- format of the input text files:
  - 1st column: origin zone *i*
  - 2nd column: destination zone *j*
  - Starting from 3rd column: each column stores an attribute

2) Main methods:

- public boolean readData(File file, int nfields, int zone): read zone-to-zone data from a text file that is passed over from each subclass of it, and check for errors. It returns true if data is read successfully; otherwise, return false.
- public int[] getOD(): get origin and destination zone IDs.
- public double[][] getAttrs(): get zone-to-zone attributes.

## Transit

1) Class responsibilities:

- extend class ZonalToZone and read zone-to-zone transit skims from the specified transit skim file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get data from the transit skim file by calling the readData() method in its superclass ZoneToZone. It returns true if data is read successfully; otherwise, return false.

## Highway

1) Class responsibilities:

- extend class ZonalToZone and read zone-to-zone highway skims from the specified highway skim file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get data from the highway skim file by calling the readData() method in its superclass ZoneToZone. It returns true if data is read successfully; otherwise, return false.

## Trips

1) Class responsibilities:

- extend class ZonalToZone and read zone-to-zone trips for 5 SE strata from the specified trip file, which is passed over from Organizer.

2) Main methods:

- public boolean getData(): get trips from the trip file by calling the readData() method in its superclass ZoneToZone. It returns true if data is read successfully; otherwise, return false.