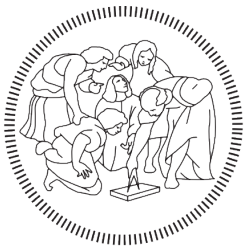
TRAVLENDAR+



POLITECNICO
MILANO 1863

Requirement Analysis and Specification Document

29/10/2017 - v1.0

Matteo Colombo Alessandro Perego Andrea Troianiello

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Matteo Colombo, Alessandro Perego, Andrea Troianiello
Version:	1.0
Date:	29-October-2017
Download page:	GitHub - ColomboPeregoTroianiello repository

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document is the baseline for project planning and for software evaluation of Travlendar+; it describes the system in terms of functional and nonfunctional requirements, analysing the needs of the customer in order to model the system.
Travlendar+ is a calendar-based application for people who have problems with scheduling working and personal appointments at various locations all across Milan. The application aims at simplifying the life of people by automatically computing travels and by organizing the daily schedules.

1.2 Scope

Travlendar+ is a calendar-based application whose purpose is to help people to easily schedule working or personal meetings around Milan. Travlendar+ is intended to all people that need to organize your daily events and movements. The system-to-be will allow users to plan travels and meetings without worrying of being late or to miss lunch and, thanks to its high customizability, people will be able to save a lot of time. Alerts will be given when meetings are created at unreachable locations and travel means will be computed depending on day hours and weather.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- Schedule: set of meetings of the same day.
- Calendar: set of the schedules and from this you can select a specific schedule.
- Meeting: event in the schedule of the user that has a location and a start and end hour.
- Break: event that has a custom time range in which it can be schedule, with a minummum duration of 5 minutes.
- Lunch: it's a particular break with no customizable range time and minimum duration of 30 minutes.
- Travel pass: represents all type of passes (daily, weekly, monthly and yearly pass) for public transportations.
- Level: pollution level of a travel travel mean.
- Event type: the type of a meeting (e.g. family, personal, work, etc).

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document.
- IEEE: Institute of Electrical and Electronic Engineers.
- API: Application Programming Interface.

1.3.3 Abbreviations

- [G.x]: the goal number x.
- [R.x.y]: the requirement number y of the goal x.
- [D.x]: the domain assumption number x.

1.4 Reference Documents

- Specification Document: “Mandatory Project Assignments.pdf”.
- IEEE Std 29148-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering.
- Alloy Specification:"Software Abstractation", Daniel Jackson.
- Alloy:"Alloy e l’Analyzer versione 4.0", "Alessandro Pagliaro"

1.5 Document Structure

This document is essentialy structured in four part:

- Section 1: Introduction, this part gives a little description of the project and the definitions about terms and acronyms used in the document.
- Section 2: Overall Description, gives more information about the software product in particular its functions, constraints and assumptions.
- Section 3: Specific Requirements, describes the complete requirements of the project. You can see a list of possible scenarios, the use cases and the sequence diagram of the specific actions.
- Section 4: Alloy Description, this part is the representation of the structure of the project in Alloy language. You can see the entire code and the representation of one generated world.
- Section 5: This section contains the details about the efforts of each member of the group and the information about the software used in the writing of the document.

2 Overall Description

2.1 Product Perspective

Travlendar+ will be developed from scratch and it will be a mobile application that will require the user to have a smartphone with an internet connection and an integrated GPS system.

The application will allow the user to digitalize his schedule and will easily manage it. When the user creates a new meeting, he is asked to provide the coordinates of the location, so that the application can compute the travel time to reach the meeting. In this way the application makes sure that the user isn't ever late, because it permits to create a meeting only if it is reachable in time.

The application proposes only the best route for each travel that complies with the user preferences; this is achieved by using some external services like maps, public transportation and weather forecast providers.

During the journey, the application will assist the users by showing them the correct route or noticing them when they have to get off from public services. Furthermore, through an external payment system, users are able to buy travel passes for their travels and visualize them directly in the application.

2.1.1 Class Diagram

The classes Event, Public Transportation, Travel Mean, Owned Vehicle and Shared Vehicle are abstract.
The classes that implement/extend Constraint, Level, Owned Mean, Shared Mean and Public Transportation are omitted for clarity.

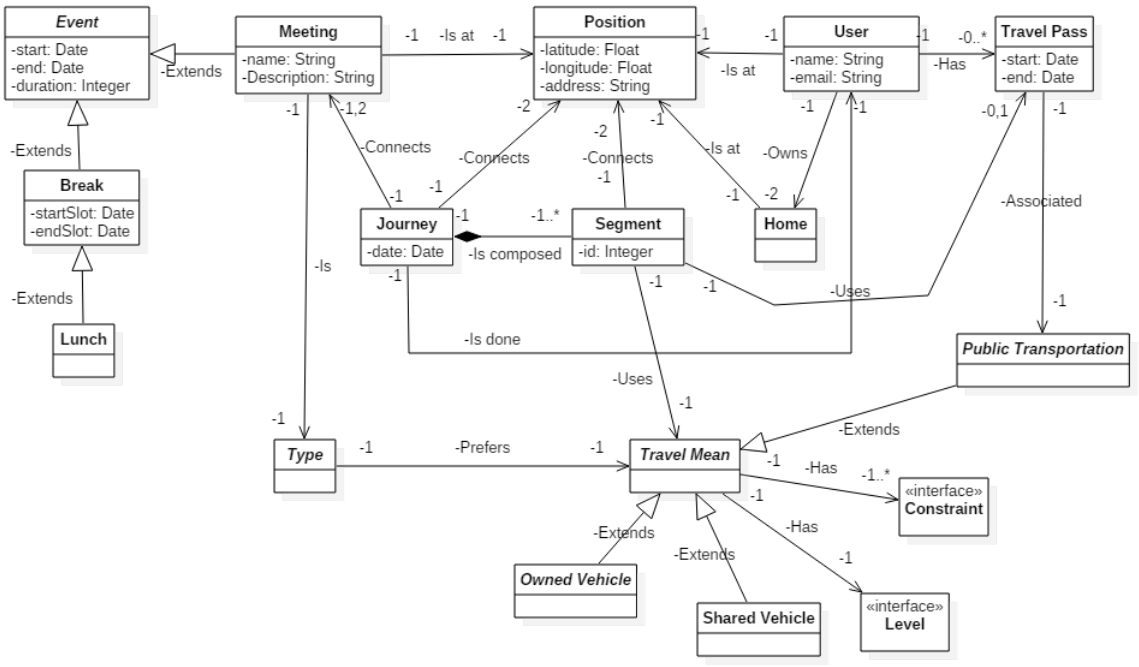


Figure 1: Class Diagram

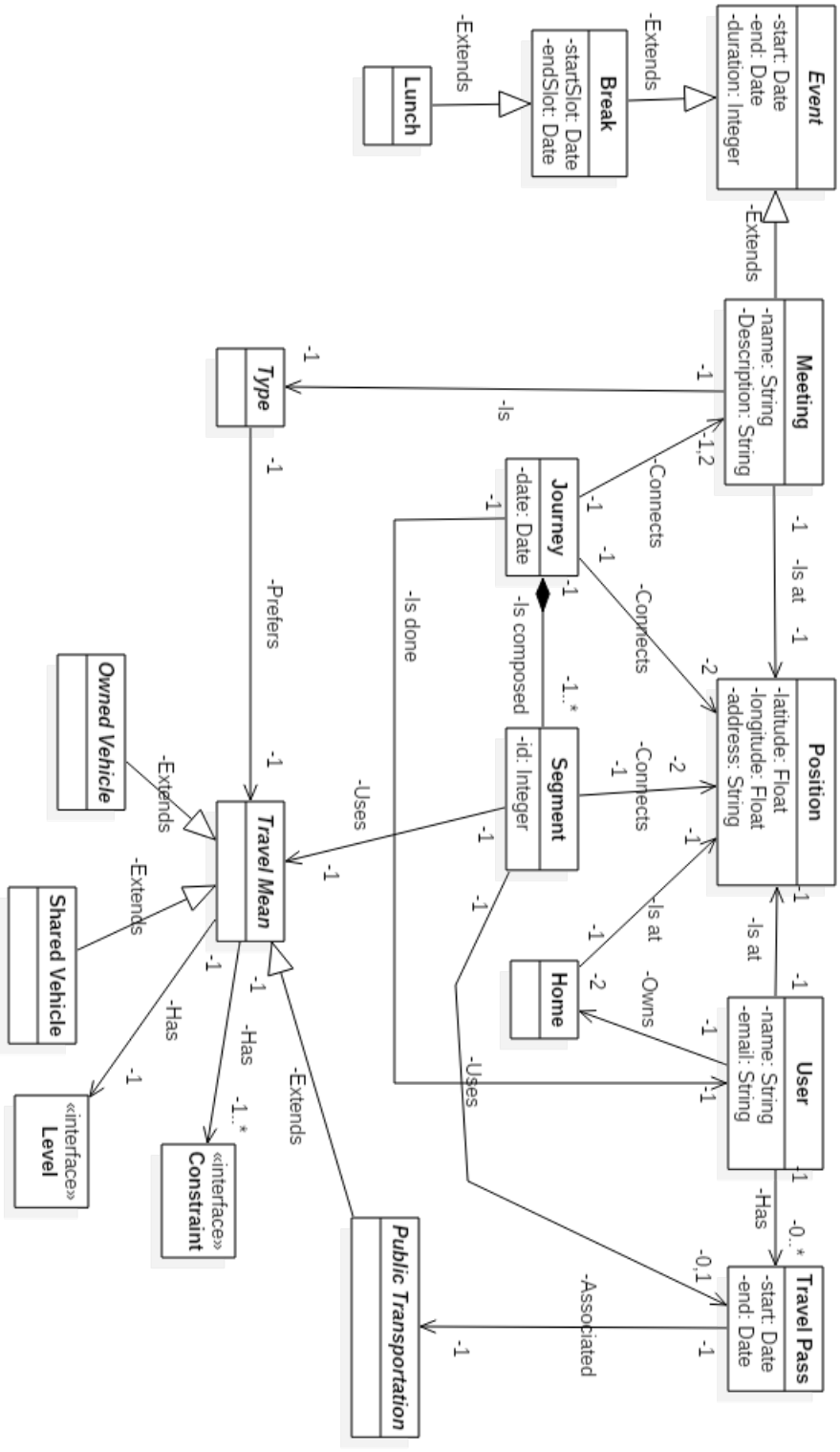


Figure 2: Class Diagram

2.2 Product Functions

[G.1] Visitors can create an account.

- **[R.1.1]** An account requires an username.
- **[R.1.2]** An account requires an email address.
- **[R.1.3]** An account requires a password.
- **[R.1.4]** The email address must be unique.

[G.2] Users can create new meetings.

- **[R.2.1]** The user must be logged in the application.
- **[R.2.2]** A meeting requires a location, which is either coordinates or an address.
- **[R.2.3]** A meeting requires a starting and ending hour.
- **[R.2.4]** A meeting requires a name.
- **[R.2.5]** A meeting requires a type.
- **[R.2.6]** A meeting may have a description.

[G.3] Warnings are shown in case a new meeting is unreachable or causes conflicts.

- **[R.3.1]** A warning should be given at the creation of a meeting if it is unreachable or if it is causes conflicts.
- **[R.3.2]** A warning should be given in case of exceptional events that make one or more meetings unreachable.
- **[R.3.3]** A warning forbids the user to force the creation of a meeting.

[G.4] Travel means are suggested depending on different constraints.

- **[R.4.1]** Different travel means should be suggested depending on the weather conditions.
- **[R.4.2]** Different type of meetings should be reached with different/specific travel means.
- **[R.4.3]** Different travel means should be suggested depending on the hour of the day.
- **[R.4.4]** Travel means can have constraints on the length of their journey section.

[G.5] Users can set their preferences and select between different options.

- **[R.5.1]** The application must support different travel means and transportation systems.
- **[R.5.2]** Users can select constraints on travel means.
- **[R.5.3]** The application must show results consistent with the users preferences.
- **[R.5.4]** Users should be able to specify the travel passes they own.
- **[R.5.5]** Users must specify their home address.
- **[R.5.6]** Users can disable travel means.
- **[R.5.7]** Users can enable the option to minimize air pollution.

[G.6] Lunch breaks are automatically scheduled by the application.

[R.6.1] The minimum length of a lunch break is 30 minutes.

[R.6.2] The lunch break must be scheduled between 11.30 am and 2.30pm.

[G.7] Users can add custom breaks that are automatically scheduled.

[R.7.1] The minimum length of a lunch break is 5 minutes.

[R.7.2] Users can set as many breaks as they want.

[R.7.3] Users can select the time slot in which they want their break to be scheduled.

[G.8] Users are assisted during their journey.

[R.8.1] The application should notify the user when he has to get off from a transportation system.

[R.8.2] The applications should give directions to the users when they are driving, cycling or walking.

[R.8.3] Users can buy travel passes through the application.

[R.8.4] The application must locate the nearest vehicle of the selected sharing system.

2.3 User Characteristics

2.3.1 Principal Actors

- Visitor: a person using Travlendar+ without being signed-up. He is able to proceed with registration or log-in.
- User: a person has successful login and can use the app services. He can manage his preferences and his appointments.

2.3.2 Secondary Actors

There are some secondary actors such as third party service providers, that are needed by the system to retrieve information, used to perform payments or to compute the travel options.

2.4 Assumptions and Dependencies

2.4.1 Domain Assumptions

[D.1] The time slot in which lunch can be scheduled is between 11.30am and 2.30pm.

[D.2] The minimum duration for a lunch break is 30 minutes.

[D.3] Breaks work in the same way of lunch, but their time slot can be customized by the user.

[D.4] The minimum duration of a break is 5 minutes.

[D.5] A meeting is always associated to a position.

[D.6] When a user adds a new meeting, if the application detects that it causes one of the following errors, it forces the user to edit the information of the meeting. Errors:

1. The meeting overlaps with another.
2. The meeting is unreachable.
3. The meeting makes another unreachable.
4. The meeting doesn't leave enough time for lunch.

[D.7] In case of a long travel that overlaps with the lunch break and doesn't give enough time to the lunch in (but there is enough time after the lunch slot), it may be splitted in two parts, separated by the break. The same applies for afternoon/morning breaks.

[D.8] The application gives higher priority at moving breaks at the beginning of their dedicated time slot.

[D.9] The application tries to maximize the duration of the breaks.

[D.10] The daily schedule starts at the user's home and ends at the last meeting of the day.

[D.11] An event starts and ends always in the same calendar day.

[D.12] GPS has a precision of 3m.

[D.13] When creating a meeting, the position typed by the user is used as meeting location to compute the travel time; while during the navigation, the application retrieves his real location through GPS and updates travels times with it.

[D.14] The user can select a preferred travel mean for each type of meeting, so that depending on the type, the application prefers different travel means. For example, if for family meetings "car" is the preferred travel mean, if it is available, it would be the suggested mean, even if better options exist.

[D.15] Travel means can be subjected to different type of constraints, some chosen by the user other by the application.

[D.16] Travel means are classified by their footprint level.Travel passes can be of different types:

1. Daily pass.
2. Weekly pass.
3. Monthly pass.
4. Yearly pass.

[D.17] A travel pass can be used only for one transportation mean. For example, a bus ticket can't be used for a tram journey.

[D.18] Cars and Bikes are of two types: shared and owned.

[D.19] At least one vehicle of a sharing system can always be found within a 10 minute walk.

[D.20] Each vehicle of a sharing system can be located through GPS.

[D.21] During a single journey, many different travel means can be used.

[D.22] Public transports are always on time.

[D.23] Strikes or public transportation fault are communicated on the website of the service provider.

[D.24] The system suggests to the user the best journey that complies with the users preferences.

[D.25] Meetings can be of three types: family, work and personal.

[D.26] The server is used for account related purposes and as backup.

[D.27] When a new meeting is created, the application synchronizes the schedule with the server.

[D.28] If when a new meeting is created the connection to the server isn't available, the meeting is created anyhow and the synchronization will be done when the connection returns.

[D.29] If a conflict is generated when the user tries to change the time slot dedicated to a break, a warning is shown and the change fordbidden.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The following images represent the mockup of some of the most important pages of the application.

Figure 3: Homepage

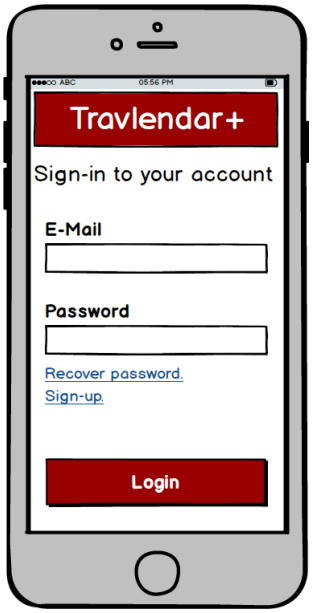


Figure 4: Login page

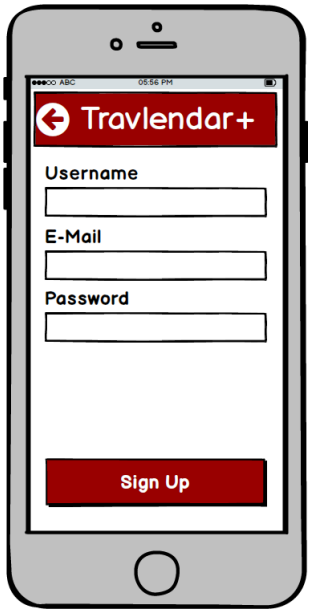


Figure 5: Sign up page

Figure 6: Navigation page

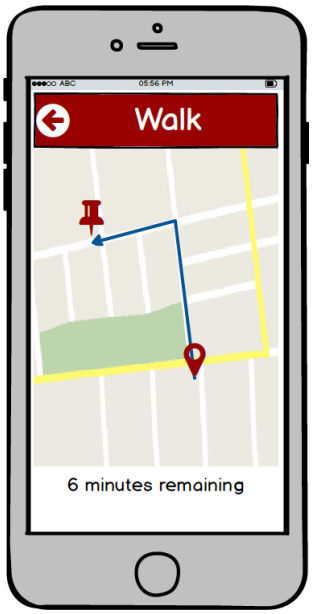


Figure 7: Walking assistant

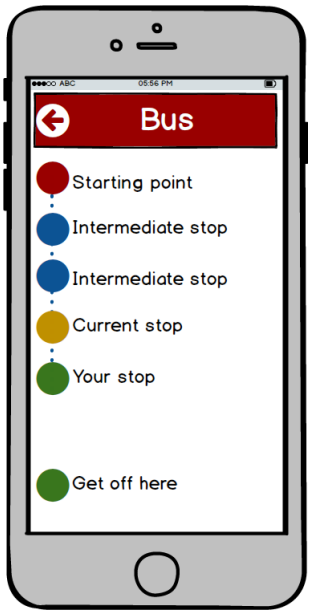


Figure 8: Bus assistant

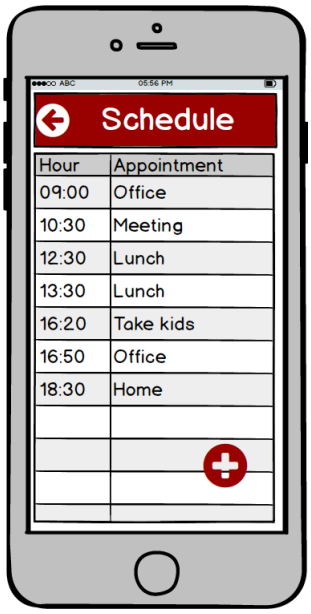


Figure 9: Schedule page

Figure 10: Meeting page

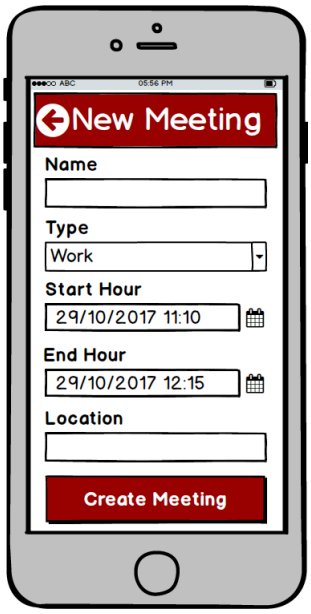


Figure 11: New Meeting

Figure 12: Warning Pop-up

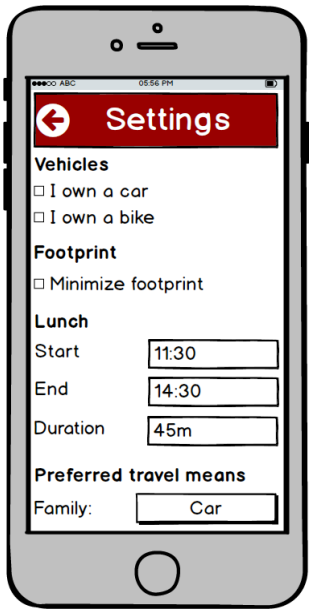


Figure 13: Preferences Page

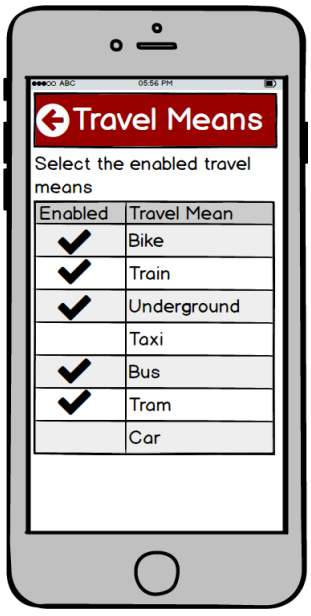


Figure 14: Travel Means

3.1.2 Hardware Interfaces

This project doesn't require any hardware interface.

3.1.3 Software Interfaces

- Database Management System (DBMS):
  - Name: MySQL
  - Version: 5.6.21
  - Source: https://www.mysql.com
- Operating System:
  - Name: Android
  - Version: 5.0 Lollipop or higher
  - Source: https://www.android.com/
  - Name: IOS
  - Version: 8 or higher
  - Source: https://www.apple.com/ios
- Application Server:
  - Name: Glassfish
  - Version: 4.1
  - Source: https://javaee.github.io/glassfish/

3.1.4 API Interfaces

For map visualization and to show the track of the journey we use the Google Maps API. This API provides the best and most complete maps system. We can retrieve real time information for road traffic and use these to compute a precise travelling time.

For the weather information we use the OpenWeatherMap API. This API gives access to current weather data for moreover than 200.000 cities on the world and it takes data from more than 40.000 wheater stations. The data is avaible in various formats: JSON, HTML and XML. We use the 3 hours weather forecasts. For more information see the OpenWeatherMap website (http://openweathermap.org/api).

To retrive the information of the schedule and anomalies of train, bus, tram and underground we have an agreement with the companies that provide the services. With this agreement we can use their data to calculate the best track for our users and an advise them when for some reason a service is not avaible.

3.1.5 Communication Interfaces

For communications we use the TCP protocol on port 80 for HTTP,port 443 for HTTPS, port 3306 for MySQL database. The data from the server to the application arrive in a JSON format.

3.2 Scenarios

3.2.1 Scenario 1

Mario is the CEO of a manufacturing company in Milan, his company collaborates with many shops located around in the region. Once a month he must visit each shop to get a report of administrative and management activities. To organize and take into account the travel time for the appointments, he uses Travlendar+. After having downloaded the application and registered his data, Mario has the possibility to create a meeting into the app for every appointment that he has in his personal calendar. During the meeting creation, he needs to specify the following information: location for the meeting, starting and ending time, name and type of the appointment. The app automatically calculates if the time between the end of the first meeting and the start of the second one is enough for the journey; in case the time is too short and the second meeting wouldn't be reachable in time, a warning is shown.

3.2.2 Scenario 2

Luca is a young architect of Milan, and in addition to using the latest graphic system to realize his project, he usually produces a demonstration model for his customers scattered around the city. The great capabilities of Luca permits to him to have a lot of appointments, so he uses Travlendar+ to manage the events. Luca has also set the preferences to use car sharing systems for work appointments, in this way he can carry his models around the town without risking of breaking them. The preferences system implemented in the application shows to the user the best itinerary that respects settings and user's preferences.

3.2.3 Scenario 3

Travlendar+ is the perfect application for Giovanni, father of family and financial advisor. With the application Giovanni can easily manage his travel time to reach every type of appointment. Travlendar+ always suggests to him the perfect way to reach a meeting, in case of work meetings the primary suggestion is public transportation, instead in case of family meetings the primary suggestion is his own car. When Giovanni chooses the public transportations for his movements he can also buy the ticket directly on the app, in a simple way he can select the payment system and in a few taps he can buy it. Furthermore in case of strike the app sends a notification alert to the user and it will recalculate the travel based on the new options.

3.2.4 Scenario 4

Antonio is an event organizer and meantime he is a food lover; this is the reason why in his calendar is always presents a lunch in one of the best restaurant of Milan. Antonio is a daily user of Travlendar+ for this reason, the app in fact force a mandatory break of at least 30 minutes for lunch. In this way Antonio can have his lunch break in a period of time between 11.30 am and 2.30 pm and he can arrive in time for the next appointment. Over the lunch break, Antonio usually plans a break in the afternoon, with Travlendar+ he can do this and the minimum time for the break is 5 minutes

3.2.5 Scenario 5

Alex is a olympic champion of foot race, when he is not training he is often invited to attend some sport conferences. Alex always suggests that it is really important to do physical activity not only during the training sessions but also when we move around the city. For this reason Alex can not do to use Travlendar+, indeed he chooses the bike for his movements and also selects the option to minimize the carbon footprint. Furthermore the application notifies Alex when the weather conditions are not optimal and suggests him to use another travel mean to reach his desiderated location.

3.3 Functional Requirements

3.3.1 Use Case Diagram

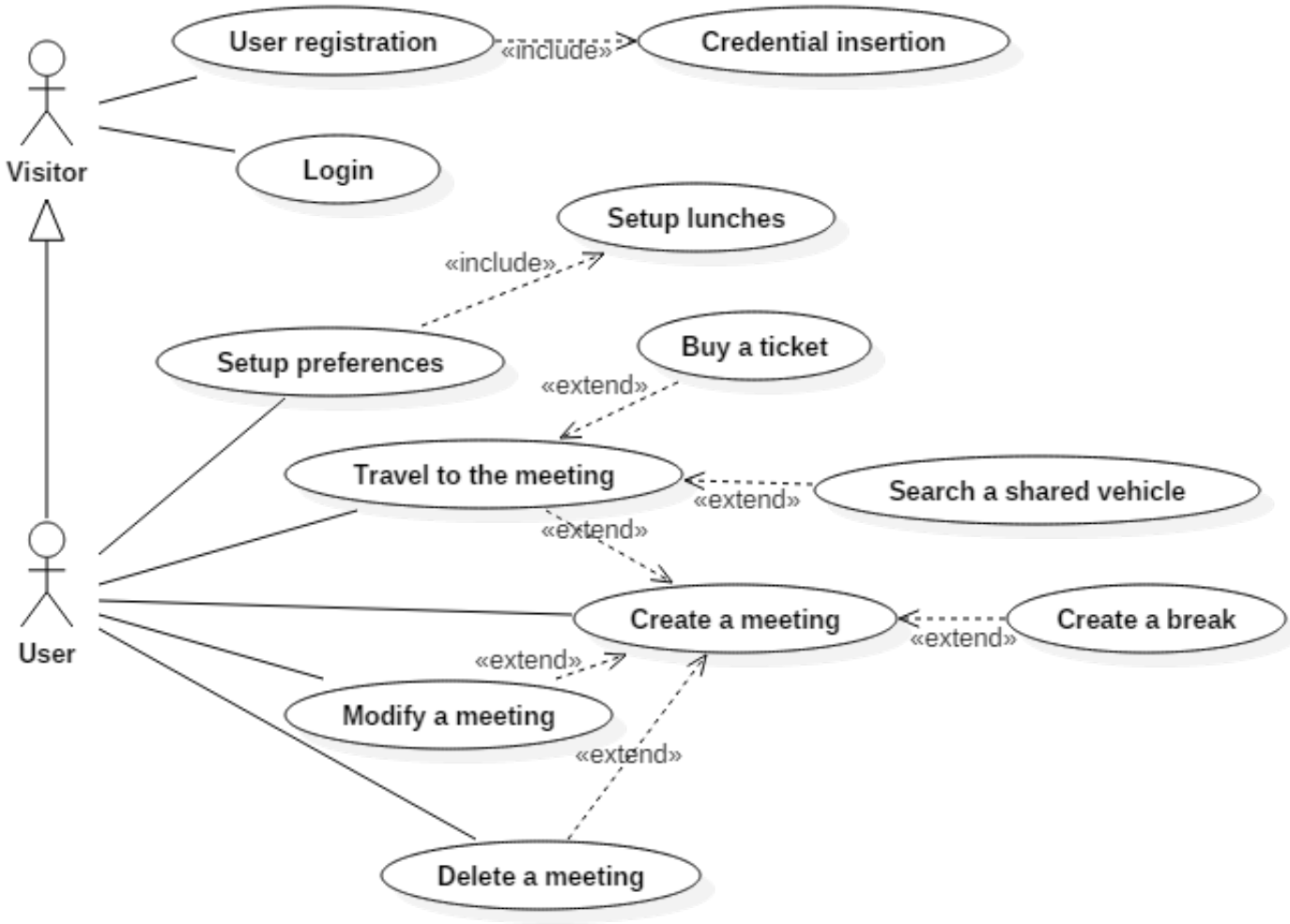


Figure 15: Use Case Diagram

3.3.2 Use Case Descriptions

In this section are listed some common or significant use cases derivable from the Use Case diagram.

Visitor's registration	
Actors:	Visitor
Goals:	G.1
Input conditions:	The visitor is on the home page.
Event flow:	1. The visitor clicks any button and is redirected to the login page. 2. The visitor clicks on the "Sign Up" button to start the registration process. 3. The visitor fills all the mandatory fields. 4. The visitor clicks on the "Sign Up" button. 5. The system send the data to the server. 6. The system sends a confirmation email to the new user.
Output conditions:	The visitor successfully ends the registration process and is redirected to the login page. From now he can login to the application and start using that.
Exception:	1. The visitor is already an user. 2. The visitor inserts not valid information in one or more mandatory fields. 3. The visitor chooses an email that is associated with another user. 4. The server is unreachable. All exceptions are handled notifying the issue to the visitor and taking back the Event Flow to the point 2.

Table 1: Visitor's registration

User's login	
Actors:	User
Goals:	G.1
Input conditions:	The user is on the login page.
Event flow:	1. The user inserts his credentials into the "email" and "password" fields. 2. The user clicks on the "Login" button in order to access.
Output conditions:	The user is successfully redirected to the home page.
Exception:	1. The user inserts a not valid email. 2. The user inserts a not valid password. 3. The server is unreachable. All exceptions are handled notifying the issue to the user and taking back the Event Flow to the point 1.

Table 2: User's login

Create a meeting	
Actors:	User
Goals:	G.2, G.3
Input conditions:	The user is already logged into the system and is into the schedule page.
Event flow:	1. The user clicks on the "New meeting" button to start the creation process. 2. The user inserts the information into the fields and the type. 3. The user clicks on the "Create meeting" button.
Output conditions:	The user is successfully redirected to the meeting page.

Continued on next page

Exception:	1. The name isn't valid. 2. The start hour doesn't precede the end hour. 3. The start hour and end hour precede the current date. 4. Exists an another meeting with the same hours. 5. Travel mean is unavailable. 6. Meeting is unreachable. All exceptions are handled notifying the issue to the user and taking back the Event Flow to the point 2.

Table 3: Create a meeting

Travel to the meeting	
Actors:	User
Goals:	G.4, G.8
Input conditions:	The user is already logged into the system and is into the schedule page.
Event flow:	1. The user clicks on the desired meeting and is redirected into the meeting information. 2. The user clicks on "Navigate" and goes into navigate page. 3. The navigate page helps the user with indications. 4. When the user arrives at the desired location, the application notifies him.
Output conditions:	The user is successfully redirected to the schedule.

Continued on next page

Exception:	
	1. The GPS is unavailable. 2. Internet connection isn't working. 3. The location isn't found. All exceptions are handled notifying the issue to the user and taking back the Event Flow to the point 1.

Table 4: Travel to the meeting

Setup preferences	
Actors:	User
Goals:	G.5, G.7
Input conditions:	The user is already logged into the system and is into the home page.
Event flow:	1. The user clicks on "Setting" button and is redirected into setting page. 2. The user inserts his preferences about travel means and lunch timetable. 3. The user confirms the new information with "Confirm" button.
Output conditions:	The user stays in the setting page.
Exception:	1. The user doesn't insert numbers in the lunch fields. All exceptions are handled notifying the issue to the user and taking back the Event Flow to the point 2.

Table 5: Setup preferences

3.3.3 Sequence Diagrams

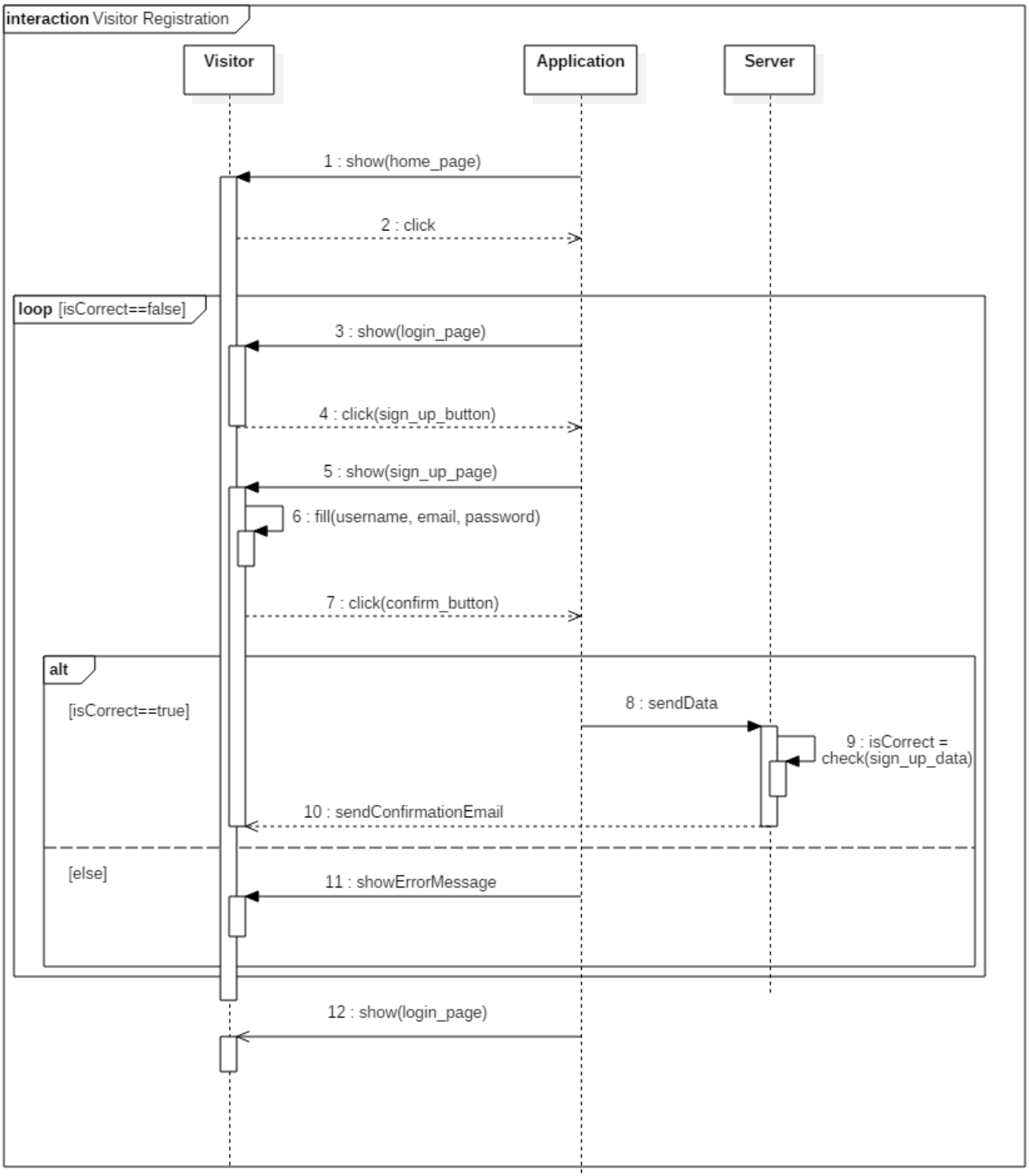


Figure 16: Visitor Registration Diagram

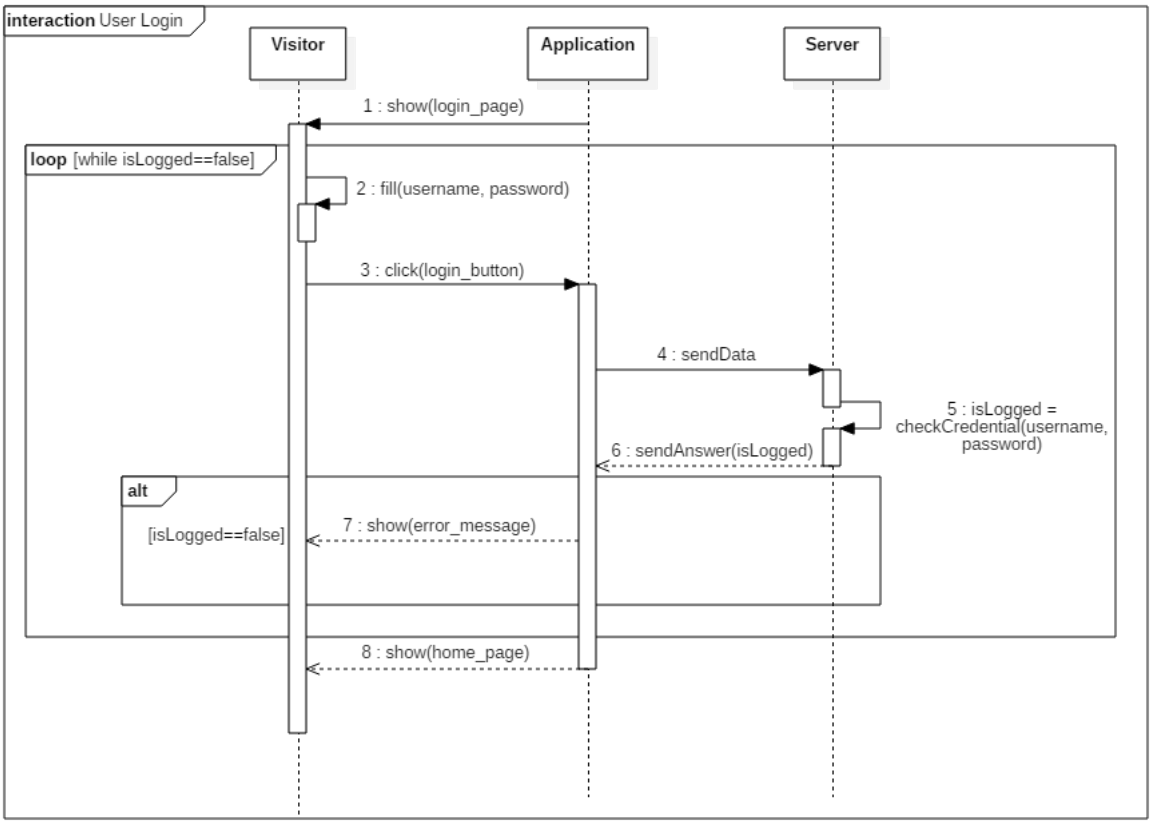


Figure 17: User Login Diagram

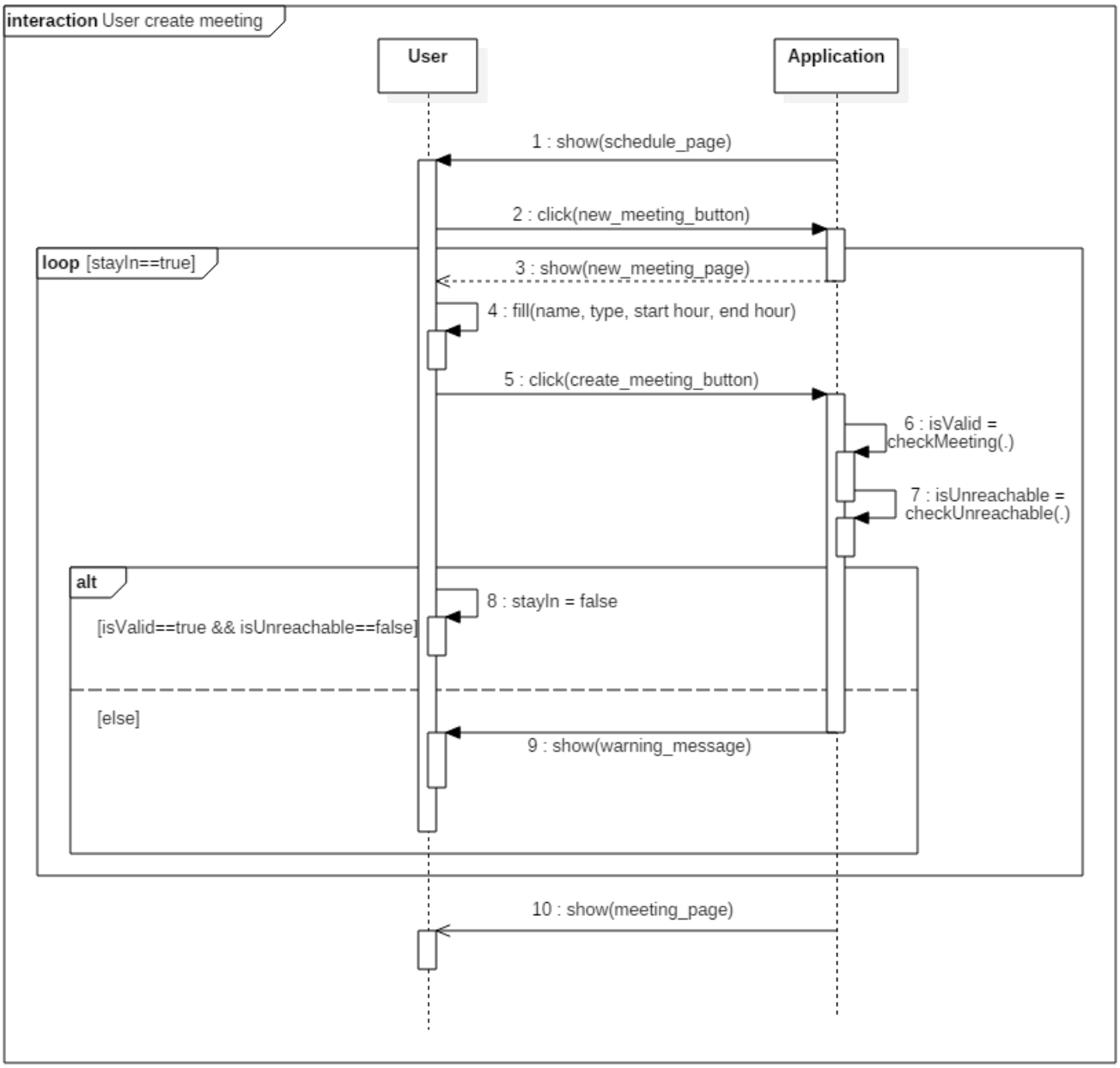


Figure 18: Create Meeting Diagram

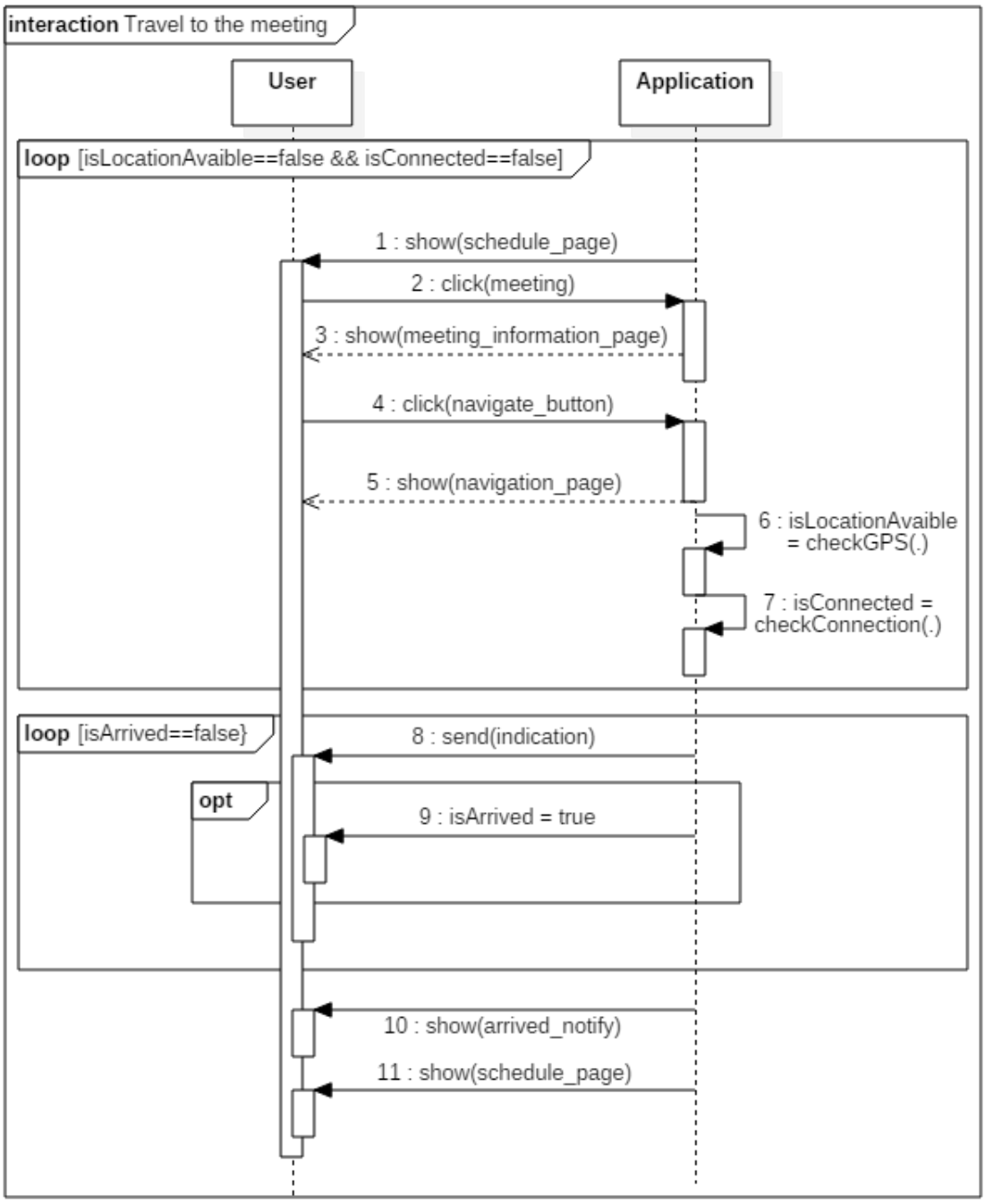


Figure 19: Travel To The Meeting Diagram

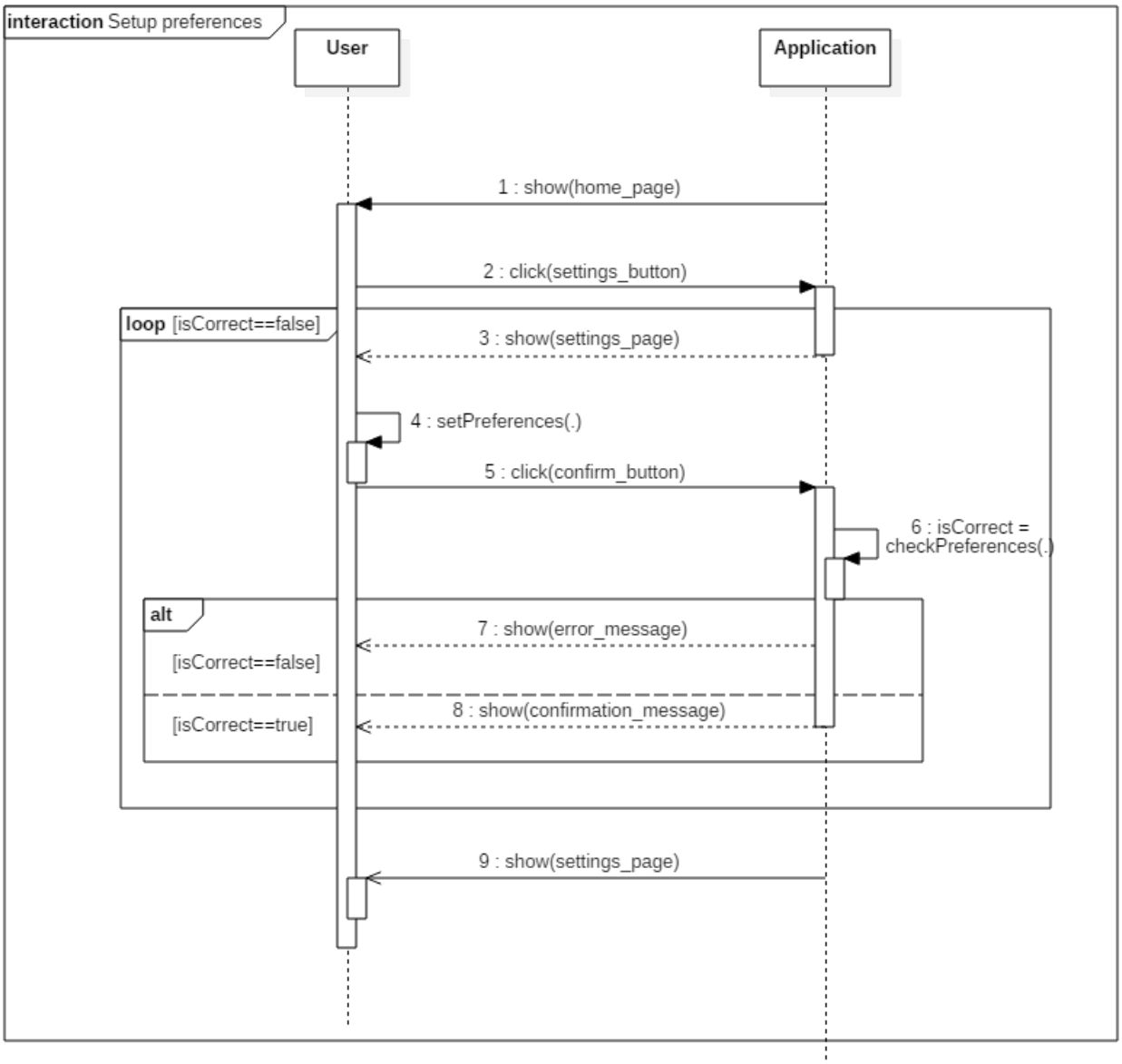


Figure 20: Setup Preferences Diagram

3.4 Performance Requirements

1. 95% of the requests should be processed within 5 seconds.
2. 100% of the requests should be processed within 20 seconds.
3. There is no limit to the total number of the registered users.

3.5 Design Constraints

3.5.1 Hardware Constraints

1. To use the application, the user must have a smartphone; either Android or iOS.
   For Android the minimum supported OS version is Android 5.0 Lollipop.
   For iOS the minimum required OS version is iOS 8.
2. The application requires an Internet connection.
3. The application requires that devices have an integrated GPS system.

3.5.2 Software Constraints

1. If the users decide to use the bike sharing systems, they must install the systems applications on their phones.

3.5.3 Safety Constraints

1. The system must guarantee that users data are stored in a safe way and that are used only within the application.

3.5.4 Regulatory Constraints

1. The system must ask for the users permission to acquire, store and elaborate their location.
2. The system must complain with the local laws.
3. It is responsibility of the user to complain with the rules and the local laws.

3.6 Software System Attributes

3.6.1 Reliability

The system is designed to run on a single server and its reliability depends on the server one. The server is required only for logins an backups of the users' schedules. In case of downtime, as long as the user is logged in the application, the fault wouldn't affect the functions and backups would be carried out when the service returns available.

3.6.2 Availability

The system is required to have a 99% uptime. Most scheduled downtimes must occur in the weekends at night time.

3.6.3 Security

All the communications between the clients and the server will be encrypted and protected by using the SSL protocol. All attempts of establishing an unsecure communication channel with the server must be refused. All the information will be stored on the server and security will have high priority; users' passwords will not be stored in plain text.

3.6.4 Maintainability

The code should be documented using JavaDoc in order to enable other developers to easily understand and edit it. The system must provide a configurable logging function for debugging purposes. The development of the software will follow the object-oriented Model-View-Controller pattern and the principles of separation of concerns. The source code management should be done through a Version Control System (e.g. git).

3.6.5 Portability

The backend will be developed in Java so that it will be possible to run the application on every machine that supports Java Virtual Machine. The frontend will be developed in Java for Android and in Swift for iOS and the application must support at least the last three version of each operative system.

4 Formal Analysis Using Alloy

4.1 Alloy

```
sig Position{}
sig Text{}

/*
Time represents a date.
day: number of days passed since the default day
minute: number of minutes passed since the default day
The control to check if  the number of minutes is coherent with the day
isn't actually performed, but it is supposed to be always true
*/
sig Time{
    timestamp: one Int,
    day: one Int
}
{
    day > 0
    timestamp > 0
}

// An event is composed by two times: start and end.
abstract sig Event{
    start: one Time,
    end: one Time
}
{
    start.timestamp < end.timestamp
    start.day = end.day //An event starts and ends in the same day
}
```



```
/*
The slot is the range of time in which the Break can be scheduled.
The slot must be in the same day of the start and end dates
*/
sig Break extends Event{
   slotstart: one Time,
   slotend: one Time
}
{
   slotstart.timestamp <= start.timestamp
   slotend.timestamp >= end.timestamp
   slotstart.timestamp < slotend.timestamp
   slotend.day= slotstart.day
   slotstart.day = start.day
}

sig Lunch extends Break{}

sig Meeting extends Event{
   name: one Text,
   desc: one Text,
   position: one Position,
   type: one EventType
}
```



```
/*
For each Event Type there is a preferred travel mean
*/
abstract sig EventType{
    prefers: one TravelMean
}
one sig Family extends EventType{}
one sig Work extends EventType{}
one sig Personal extends EventType{}

one sig User{
   home: one Home,
   currpos: one Position, //The current position of the user
   passes: set TravelPass
}

one sig Home{
   pos: one Position
}

sig Journey{
   user: one User,
/*
   The starting meeting of the journey, null if the user starts from home
*/
   startmeet: lone Meeting,
   endmeet: one Meeting, // The ending meeting of the journey
   start: one Position, //The starting position
   end: one Position, // The end position
   startseg: one Segment,
   endseg: one Segment,
   intseg: set Segment,
   day: one Int //The day in which the journey happens
}{
```



```
 start != end //The journey must connect two different positions
 end = endmeet.position
 startmeet != endmeet //The two meetings must be different
 /* If the user starts from home, the start position is the home position, otherwise
 it is the position of the starting (previous) meeting */
 #startmeet = 1 implies start = startmeet.position && (startmeet.start).day = (endmeet.start).day
                                  && (startmeet.start).timestamp < (endmeet.start).timestamp
              else start = (user.home).pos



    day = (endmeet.start).day
  /*
  Concatenation of segments, checks that the path is linear and that there are no repeated segments
  It also checks that the starting position of the first segment is equal to the starting position of the journey
  and that the ending position of the ending segment is equal to the ending position of the journey
  */
  startseg.start=start
   endseg.end = end
   no x: intseg | x=startseg || x=endseg
   lone x: intseg | x.start = startseg.end
   lone x: intseg | endseg.start = x.end
   no x: intseg | startseg.start = x.end
   no x:intseg | x.start= endseg.end
   all x: intseg | x.end = endseg.start
      or one xl: intseg | x.end = xl.start
   all x: intseg | x.start = startseg.end
      or one xl: intseg | x.start = xl.end
   #intseg = 0 implies (startseg = endseg or startseg.end=endseg.start)
}

sig Segment{
  id: one Int, //Unique
  start: one Position,
  end: one Position,
```



```
  mean: one TravelMean,
  pass: lone TravelPass, // null if no travel pass is used
  journey: one Journey
}
{
  id >0
  start != end
  /* The travel pass must be valid */
  #pass=1 implies mean = pass.mean && journey.day>= (pass.start).day
            && journey.day <= (pass.end).day
}

abstract sig TravelMean{
  constr: set Constraint,
  pollution: one Level
}
abstract sig PublicTransport extends TravelMean{}
one sig Bus extends PublicTransport{}
one sig Train extends PublicTransport{}
abstract sig SharedVehicle extends TravelMean{}
one sig Bike extends SharedVehicle{}
abstract sig OwnedVehicle extends TravelMean{}
one sig Car extends OwnedVehicle{}
one sig Walk extends OwnedVehicle{}

abstract sig Constraint{}

abstract sig Level{}
one sig Level1 extends Level{}
one sig Level2 extends Level{}
one sig Level3 extends Level{}
```



```
sig TravelPass{
   start: one Time,
   end: one Time,
   mean: one PublicTransport
}
{
    start.day <= end.day
}

/* ------------------------------------------------
FACTS
*/ ------------------------------------------------

//All events must be disjoint and not overlapping
fact allEventDisjoint{
   all disj e1,e2: Event | ((e1.start).timestamp) > ((e2.end).timestamp)
                        || ((e2.start).timestamp) > ((e1.end).timestamp)
}

//The ID of segments must be unique
fact noSegWithSameId{
   no disj s1,s2: Segment | s1.id = s2.id
}

 //A segment  to exist must belong to a journey
 fact eachSegBelongsToOneJourney{
    all s: Segment | one j:Journey | (s in j.intseg || s=j.startseg || s=j.endseg)
    all s: Segment |one j:Journey | s.journey = j
 }

//We make sure that if on a day there is a meeting, there is also one and only one lunch
fact oneLunchADay{
   all disj l1,l2: Lunch | (l1.start).day != (l2.start).day
   all m:Meeting | one l:Lunch | (l.start).day = (m.start).day
```



```
}

//We make sure that a journey is created together with a meeting
fact aMeetingNeedsAJourney{
  all m: Meeting | one j: Journey | j.endmeet = m
  all m: Meeting | lone j: Journey | j.startmeet = m
}

//The user must own a travel pass to use it
fact travelPassMustBeOwned{
  all p: TravelPass | one u:User | p in u.passes
}

//We make sure that only one journey a day starts from home
fact atMostOneJourneyAtADayStartsFromHome{
  all disj j1,j2: Journey | (j1.day=j2.day && #j1.startmeet = 0) implies #j2.startmeet = 1
}

/*The number of journeys is always equal to the number of meeting
  In this ways we make sure that the user every day makes a journey from home*/
fact journeyNumber{
  #Journey = #Meeting
}
```



```
/*-----------------------------------------------
  ASSERTIONS
*/ -----------------------------------------------

//Checks that all events are disjoint
assert allEventsDisjoint{
  no disj e1,e2: Event | (e2.start.timestamp > e1.start.timestamp
                        && e2.start.timestamp < e1.end.timestamp)
                        || (e2.end.timestamp > e1.start.timestamp
                        && e2.end.timestamp < e1.end.timestamp)
}

//Checks that if on a day there is a meeting, there is a lunch
assert ifThereIsAMeetingThereIsLunch{
  no m:Meeting | all l:Lunch | (l.start).day != (m.start).day
}

//Checks that the lunch is always scheduled in the range
assert lunchIsAlwaysInTheLunchSlot{
  no l:Lunch | l.start.timestamp < l.slotstart.timestamp
             || l.end.timestamp > l.slotend.timestamp
}

//Checks that the journey that leaves from home is the first of the day
assert journeyFromHomeFirstOfTheDay{
  no disj j1,j2: Journey | j1.day=j2.day && #j1.startmeet = 0 && #j2.startmeet = 0
}

//Checks that tickets used in a segment are valid
assert usedTicketsAreValidDuringTheSegment{
  all s:Segment | #s.pass = 1 implies s.pass.start.day <= s.journey.day
                                   && s.pass.end.day >= s.journey.day
}
```



```
/*
  PREDICATES
*/
/*This generated a simplified world,
  with the schedule of one day in which there are 2 meetings and three segments
   along two journeys
*/
pred show(){
  (Meeting.start).day=3
  #Meeting = 2
  #Segment = 3
  #Position = 4
  #TravelPass = 1
}

check lunchIsAlwaysInTheLunchSlot for 6 but 5 int
check ifThereIsAMeetingThereIsLunch for 6 but 5 int
check allEventsDisjoint for 6 but 5 int
check journeyFromHomeFirstOfTheDay for 6 but 5 int
check  usedTicketsAreValidDuringTheSegment for 6 but 5 int
run show for 6 but 5 int
```

6 commands were executed. The results are:

#1: No counterexample found. lunchIsAlwaysInTheLunchSlot may be valid.
#2: No counterexample found. ifThereIsAMeetingThereIsLunch may be valid.
#3: No counterexample found. allEventsDisjoint may be valid.
#4: No counterexample found. journeyFromHomeFirstOfTheDay may be valid.
#5: No counterexample found. usedTicketsAreValidDuringTheSegment may be valid.
#6: **Instance found.** show is consistent.

4.2 Generated World

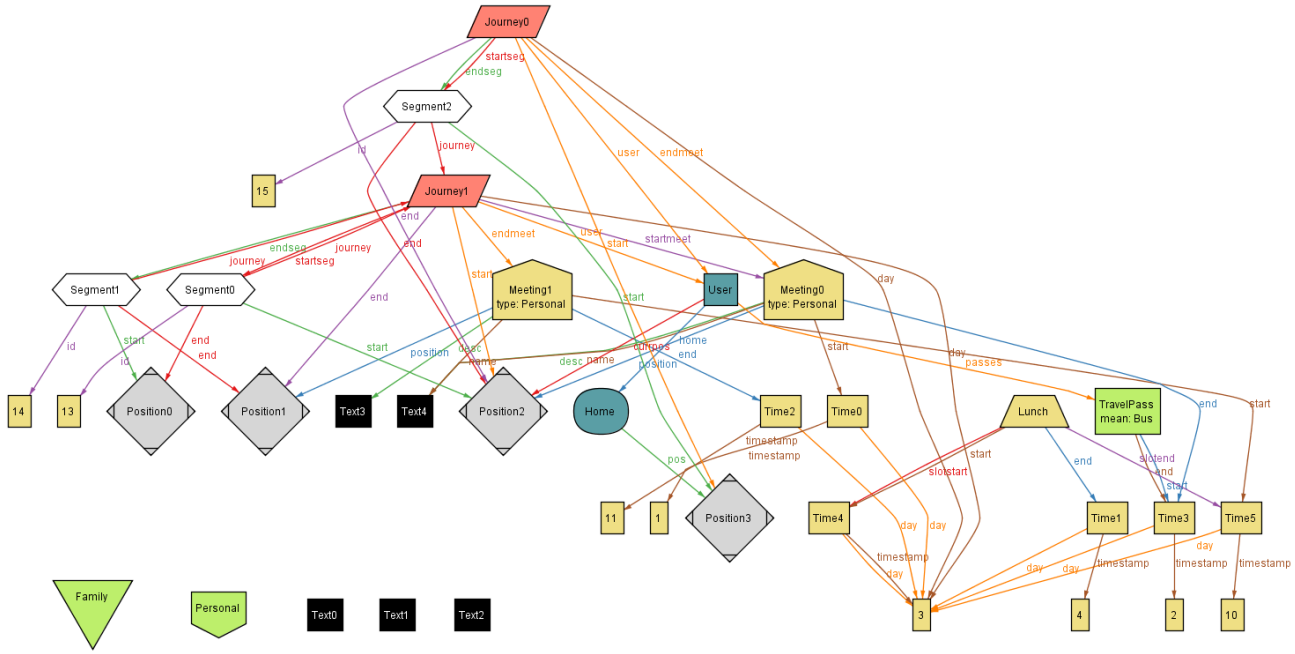


Figure 21: Generated World Magic Layout

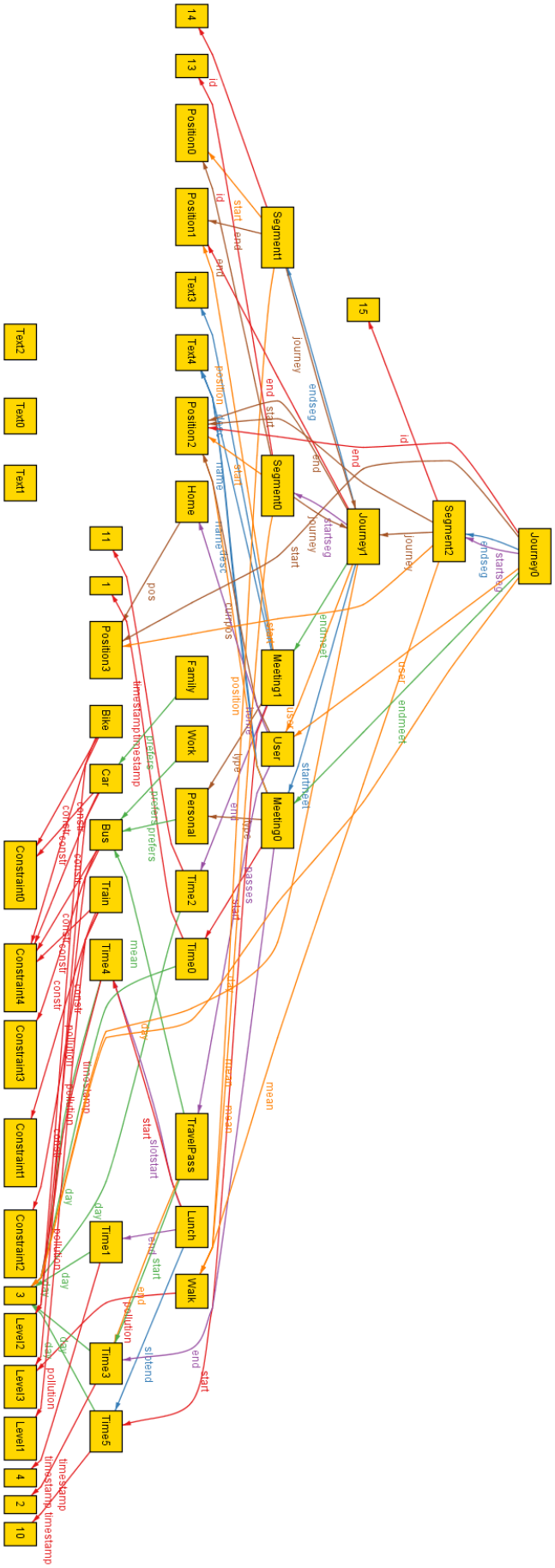


Figure 22: Generated World

5 Effort Spent

5.1 Time Spent

Member	Hours
Colombo Matteo	45
Perego Alessandro	45
Troianiello Andrea	45

Table 6: Time Spent

The commits on GitHub are not completely representative of the work done, a good part of it has been done in group.

5.2 Used Tools

1. TexMaker for the editing of the LaTeX document.
2. Star UML for the creation of UML diagrams (Class,Use Cases and Sequence diagrams).
3. Balsamiq for the creation of the Mockup.
4. Alloy Analyzer 4.2 for alloy analysis.

