POLITECNICO DI MILANO

TRAVLENDAR+

LIFE, SIMPLIFIED

SOFTWARE ENGINEERING 2

Requirements Analysis and Specification Document

Authors:
Alessandro ALFONSO
Fabrizio CARSENZUOLA
Filippo CREMONESE

Supervisor:
Elisabetta DI NITTO

October 29, 2017

Contents

1 Introduction

1.1 Revision history

Version	Date	Authors	Summary
1.0	October 29, 2017	Alessandro Alfonso Fabrizio Carsenzuola Filippo Cremonese	Initial release

1.2 Purpose of the project

We're tasked to design Travlendar+, an application that helps its users schedule appointments and arrange for travel between them. The killer feature of Travlendar+ is it's ability to account for travel time between appointment locations. This allows to suggest the best travel means for any specific appointment, based on many factors such as time required, cost, weather, CO2 emissions, and so on. The app offers many features as it tries to be as convenient as possible for its users (hence the motto "Life, simplified"), such as:

- in-app automatic reservations for sharing mobility services (car, bike) and taxis
- in-app ticket shop for public transportation (bus, metro, train, ...)
- the possibility of setting constraints such as "maximum 2km of walking for any single travel"
- the possibility of scheduling flexible events (see 1.7 on the following page)

Profitability

Travlendar+ could be considered a venture capital backed project that has no need to be profitable in its development phase. For this reason we don't focus on how the system might earn a profit and make the assumption that it will be free to use for the time being, while avoiding decisions that will make difficult to include billing into the system. An obvious way to make a profit would be to earn a percentage of what the users pay to sharing mobility services and taxis reserved through the app.

1.3 About this document

This document analyzes the functional and non-functional requirements of the system to be developed. Requirements are based on the needs expressed by the stakeholders, and are subject to constraints imposed by the external world. Common scenarios are included to document the main interactions between the users and the system. This document mainly targets the software engineers and developers who will have to develop the service here described, but also the project owner, prof. Di Nitto.

1.4 Scope

Travlendar+ will be available on all the most used platforms (mobile and desktop). Other apps that partially cover Travlendar+ functionality already exist:

- calendar apps that provide weather information

- public transport assistants (e.g. Citymapper. See references 7.4)
- public transport ticketing apps
- navigation apps (e.g. Google Maps. See references 7.4)
- taxi apps (e.g. MyTaxi, Uber. See references 7.4)
- sharing mobility services applications (e.g. Enjoy, Mobike. See references 7.4)

Altough applications such as Citymapper or Google Maps already offer an excellent user experience, none of those apps allow to seamlessly schedule an appointment while keeping track of weather and travel times. Moreover, ticketing apps for public transportation are often poorly built, which increases the need for the in-app ticket purchase functionality in Travlendar+. The existing sharing mobility services don't offer the possibility to automatically reserve the nearest vehicle at a certain date and time.

The system will need to interact with external entities and events beyond of our control (public transportation, sharing mobility services, weather) which can disrupt user experience. The specifications of the service are based upon the assumption that most of the time those external entities will behave as expected (documented in section 2.5). When one or some external services won't behave as expected, the application will work the same but in a dregradated state. If it is possible the system should notify the user that the application will work in a limited way (and tells him which services are unaviable).

1.5 Stakeholders

The main stakeholder in this project is our professor, Elisabetta Di Nitto, which we'll consider a paying customer that has tasked our development team to design and eventually build Travlendar+. The specifications and requirements for the project are coming from her.

1.6 Reference documents

- project assignment ("Mandatory Project Assignment.pdf")

1.7 Definitions, acronyms, abbreviations

Application Depending on context we use the term application (app) to refer not only to mobile applications but also to the web version.

Flexible event An event which does not have fixed start and end times, but has a duration, start and end time constraints and optional location. The app will try to keep a free spot in order to accomodate for the flexible event. It may repeat. An example would be "30 minutes for lunch between 11:30 AM and 14:00 PM, every day from monday to friday".

MVP Minimum Viable Product

2 Overall description

2.1 Product perspective

Travlendar+ will consist of three major components:

2.1.1 User applications

Travlendar+ users will need to be able to use the service from their computers and mobile devices. We propose to build a desktop web app and Android and iOS applications. Users will be able to use Travlendar+ on all their devices and have their calendars always available thanks to automatic synchronization. For the mobile apps basic functionality (e.g. adding an event to the calendar) will be available even if the device is temporarily offline, but many features will need internet connectivity in order for the app to be able to query the backend and sync data across devices.

The web frontend will be targeted at modern browsers with support for HTML5, CSS3 and recent JS APIs. Shims will be used to ensure the best possibile compatibility with older browsers, altough not all features will be available to legacy clients. Mobile applications will need to support 95% of the user base of iOS and Android when the app will be released, which at the time of writing this document means supporting Android 4.4+ and iOS 10+. A native desktop application is not a strict requirement, but we note that with some appropriate implementation decisions the web application should be easily converted into a desktop application. The decision on whether to build a desktop version is left to the design phase, where we'll evaluate the costs and benefits.

2.1.2 Administrative application

An administrative web application will be needed to provide customer support and to manage the system. From there it will be possible to consult usage statistics, monitor service status and so on. The backend will be accessible only to authenticated Travlendar+ staff. In this document we don't analiyze how the administrative application will do or how it will work.

2.1.3 Backend

Most of Travlendar+ advanced functionality will depend on server side services for:

- map data and travel routes
- public transport information such as timetables and realtime status
- public transport tickets and passes
- sharing mobility services information and functionality (vehicle position, automatic reservations, ...)
- taxi services information and functionality (prices, automatic reservations, ...)
- calendar storage and synchronization between devices
- push and email notifications

Many of these functionalities will rely on data and services provided by external entities. Implementation details such as which specific service to use for weather data are left to the design document.

Data sources and external services
This data will come from external sources:

- public transport information
- weather forecast
- maps
- traffic information

This services will be provided by or depend on external services:

- sharing mobility services (find and reserve vehicles)
- public transportation tickets
- taxi reservations

This services may be provided by external services (the decision to implement in-house or not is left to the design phase):

- route planning
- mobility sharing and taxis payment

2.2 Main product functionality

Here we describe the main functionality that Travlendar+ will provide. Detailed functional and non functional requirements are described in section 3.

2.2.1 Appointments managment

Users will be able to add, edit and remove appointments to their calendars. Multiple calendars can be used (e.g. a user can have a work and a personal calendar). They will be able to share single events and entire calendars with other people (even not registered to Travlendar+) by generating a sharing link or sending an email. When inserting or editing an event the user can plan how to get to the event location using the app (see section 2.2.2 below).

2.2.2 Travel planning

Travlendar+ will help users plan for travel between appointments. They will have to provide the location of the appointment and the starting location and time (by default the ones of the previous event of that day, if present). They will receive suggestions compatible with the travel means they can use, and be able to see (when applicable):

- cost of the trip
- estimated time taken
- calories

- distance
- weather forecast
- CO2 emissions

A notification will inform the users in case of bad weather, strikes, accidents along the way and unavailability of sharing mobility vehicles. If it appears that there are no travel means that allow to reach the destination in time a warning is shown.

2.2.3 Automatic vehicle reservation

Users will be able to plan automatic reservation of a sharing mobility vehicle or taxi for their appointments. In case a sharing mobility vehicle is requested our service will start searching for a vehicle to reserve near the user location some time before the start of the trip. If no vehicles are available the users will receive a notification and alternative travel suggestions. The user must be able to log in (or signup if it is the first time) into the external service of sharing mobility vehicle or taxi. The payment of these services will be managed externally (directly in the sharing service or taxi system). The user before use for the first time these service must insert a valid payment method (credit card).

2.2.4 In-app tickets shop

Users will be able to buy tickets and passes for public transportation directly in the app.

2.3 Non functional requirements

Travlendar+ should be:

- fast (see section 3.2)
- multiplatform
  - desktop web app
  - mobile app (iOS and Android)
- available 24/7
- simple and user-friendly
- secure and privacy respecting (see sections 2.6 and 3.5.2)

2.4 User characteristics

Travlendar+ is most useful for users that know their schedule at least some hours ahead and live in cities with many travel options. Some scenarios are provided in section 4. The application is primarily designed to be used by registered, logged in users. Non registered users can interact with Travlendar+ on some occasions. Here we document the main characteristics and interactions of our users.

2.4.1 First time users

Users may start using Travlendar+ when:

- they access the web app (directly, from a search engine or otherwise)
- they download the app for the first time on a certain device
- they are sent an invite to try the app from a registered user

In each case the users are sent to an informational page (or application screen depending on the used platform) that shows the main features of Travlendar+ and invites them to sign up. Users that decide to sign up are asked an email and a password and will receive a confirmation email at the specified address.

It is called "Guest" in use case diagrams and sequence diagrams.

2.4.2 Unregistered users with a sharing link

Unregistered users (identified by an email address inserted by the owner of the event and other optional information) may receive a link to a shared event from a registered user or be invited to it. An email is sent to them as a reminder. The email contains all relevant information about the event such as name, date and time, location, weather forecast, and also a link to view the event in the web-based version of the app, which can be used to view updated information about the event and, if they are invited, allows them to leave a preference ("Accept", "Decline", "Maybe"). The preference will be visible to the event owner. The email also contains a link to the informational page to invite unregistered users to sign up.

It is called "Invited User" in use case diagrams and sequence diagrams.

2.4.3 Registered users

Registered users will be identified by their email and a password. Other information will be asked when appropriate:

- a dismissible "complete your profile" button will be visible in the main screen of the app
  - it will lead to the profile settings page/screen which offers to input and modify all account information
- when planning the first travel users will be asked
  - which transportation means they can and cannot use
  - credentials for sharing mobility and taxi services
  - the maximum walking distance for any single travel
- credit card information will be asked optionally on registration, otherwise on the first payment

The account settings page/screen will be accessible at any moment to edit the provided information.

It is called "User" in use case diagrams and sequence diagrams.

2.5 Assumptions and dependencies

Travlendar+ needs to interact with many entities and cope with events beyond our control, which can disrupt user experience. Here we document assumptions on which we'll rely for our design.

2.5.1 Data sources and external services

In order to provide our users convenient services such as automatic car sharing reservations or weather-aware travel planning we need to source data from external entities which cannot be expected to be perfectly reliable. The specifications of the service are based upon the assumption that most of the time those external services will behave as expected but also document what to do when this assumption does not hold.

Commercial agreements

Some of the external services (e.g. public transportation tickets, taxi and sharing mobility services) don't have a public API available, and an agreement with the companies providing the service will be needed in order to gain access and use their APIs. We assume these agreements will be made before the start of the design phase.

2.5.2 Area limit and app availability

We assume that appointments for which the user wants help scheduling his travels are limited to the same area (e.g. the metropolitan area of Milan). The app will be available only to users of certain supported cities, initially targeting the most populated Italian cities for the MVP. This constraint is necessary because:

- public transportation regulations, prices, ticketing systems and timetables (and associated data sources) vary wildly in quality, format and availability between cities
- time estimates for long travels — especially when involving public transportation — are not accurate

2.5.3 Other assumptions

We suppose these assumptions will hold:

- users will input correctly all relevant information about their appointments
- users will have a reliable internet connection, and will be offline only for short periods of time
- reserved taxis will show up (in time)
- reserved sharing mobility vehicles won't be broken or otherwise unavailable

2.6 Legal constraints

Travlendar+ will need to comply with the law of the states of the targeted users, which will include, but not limited to:

- European "Cookie Law" (Directive 2002/58/EC)
- European General Data Protection Regulation (2016/679)
- payment processing laws and tax codes

Consultation with a lawyer will be needed before the start of the development. Law compliance is not expected to be a major issue with regards to the requirements in this document.

3 Specific requirements

This section will analyze in detail functional and non-functional requirements that the developed system has to satisfy, when the domain properties previously denoted hold and referring to the declared goals.

3.1 Functional requirements

The functional requirements include the functionalities that the system must necessarily have and describe the interactions between the system developed and the external environment independently from the implementation. They are divided into categories as follows.

3.1.1 User account managment

The following operations must be supported:

- user registration
- user authentication (log in)
- user settings
  - allows to change user information such as email, password, etc...
  - allows to set constraints on the transportation means the user can employ
  - allows to provide credentials for sharing mobility and taxi services
  - allows to set some preferences on the route scheduling (e.g. set maximum km of walking in a single route, set the latest time at which to use public transportation)
- password reset
  - an email is sent to the user when he requests to reset his password
  - the email contains a link that allows to change the password
  - passwords must not be sent by email
- user can choose which transportation means to use, for example
  - choose whether to use shared vehicles or not
  - set maximun walking distance
  - insert details about which transportation passes they own
  - specify if they can drive or not

3.1.2 Calendar and events managment

- create, edit and delete calendars
- view calendars and events
  - calendars may be viewed by month, week or day

  - event details must be accessible from the calendar overview

- add an event to a calendar
- modify and delete and event
- share an event or an entire calendar with another person by email
- set and receive reminders (via push notifications and/or email) for specific events or all events in a calendar

3.1.3 Travel planning

When creating or modifying an event users will be able to specify starting and destination locations and time and schedule their travel. By default the starting point is the one of the previous event on the same day, if it exists.

- a warning must be shown if it appears that there's no suitable transportation mean to reach an appointment in time
- using the preferences and coinstraints set by the user, the application has to give a list that contains the best solutions
  - users can choose the preferred option from the list
  - the best options in terms of time, cost, distance, carbon emissions, calories to reach the destination must be shown
- if public transportation is chosen the app must offer to buy tickets or passes
  - users must be are able to by tickets or passes directly in the app
- if the user chooses to use a shared vehicle the app must offer to reserve one automatically
  - users can choose which vehicles they want to use and the app will search one to reserve from an hour before the scheduled departure
- if a taxi is chosen the app must offer to reserve one
- a notification must be shown if the scheduled trip should be changed
  - due to bad weather (and the user planned to walk or go by bike)
  - due to public transportation strike
  - due to unavailability of a sharing vehicle up to 20 minutes before the scheduled start of the trip. The app asks the user if it should continue looking for a shared vehicle or if the trip should be manually rescheduled (excluding shared vehicles)
- application must consider whether the user already has a passe when it asks them to use public transportation means

3.1.4 Flexible events

- schedule “flexible events”
  - users are able to set constraints on what they must do during the day (e.g. a user might want to have lunch between 12 am and 2 pm)
  - the app must notify users if they are going to schedule an event that goes against the set constraints

3.1.5 Customer service

Users (registered or not) could contact customer service through a form in the application. Users, if not logged in, must insert a valid email address on which a system administrator can answer the user.

3.2 Performance and responsiveness requirements

Users expect apps and websites to be fast and responsive, which entails both technical and UI requirements. We fix the following 95th percentile response times under normal conditions:

- 3 seconds maximum time for starting the app or loading the web app
- 1 second maximum response time for most pages and actions, including
  - showing the details screen for an event
  - inserting or modifying an event
  - updating profile information
- 15 seconds to load trip suggestions
- 1 minute maximum delay for receiving scheduled notifications

Trip suggestions should be loaded incrementally, so that information can be displayed to the user as soon as possible. In order to give hints to the user as to when the app is loading or processing spinners will be shown for each waiting period longer than 1 second.

3.3 External software interface requirements

Taxi services
In order to automatically reserve a taxi one or more external services must be integrated with our system. Those external services must provide the following functions:

- user authentication: depending on the agreement with the company providing the service the user must be required to register an account to the taxi service.
- reservation: service must allow to reserve a taxi at a given day, time and location
- cost estimation for a trip between two locations (including base fare)
- billing history (optional): for showing our users their trip details

Sharing mobility services
In order to automatically reserve a vehicle one or more external services must be integrated with our system. These services must provide:

- user authentication: depending on the agreement with the company providing the service the user must be required to register an account with them.
- vehicle positions: the service must allow to get a list of the available vehicles in a given area
- vehicle information: the service must allow to get details on a specific vehicle (remaining fuel, cost per minute or km, allowed passengers, ...)
- reservation: the service must allow to reserve a specific vehicle and cancel the reservation
- cost estimation (optional): the cost may be estimated by the external service or by us, given the starting and end locations

Weather services
In order to account for weather conditions for travel planning the system will need an external service providing :

- real time weather conditions at a given position
- weather forecasts at a given position, at least for the next 5 days

Public transport information service
In order to provide our users information about public transport the system will need an external service providing:

- information about which lines exist, their path, stops and timetables
- real time status about a specific line (optional)
  - vehicle position
  - strikes
  - accidents

Public transport ticketing service
In order to provide our users with an in-app ticket shop an external service which provides the following will be needed:

- information about which tickets/passes are available for a certain line
  - cost
  - duration/validity
- ticket and passes shop
  - payments must go through a payment processor which Travelendar+ can interact with directly via an API (e.g. Stripe)

Combined tickets rules (e.g. tickets valid for a single run in metro + train) might need to be hardcoded in the system, since they wildly vary between cities.

Payment processor service
In order to provide our users with an in-app ticket shop an external payment processor which allows the user to pay using his credit card (and possibly other means) will be needed. We choose to use an external payment processor to avoid having to be compliant with PCI/DSS and other regulations.

Map data and routing service
In order to help our users plan for travel an external service providing map data will be needed:

- map and points of interest
- routing service between two points, that accounts for the transport mean used (car, bike, walk)

Email service
In order to send messages for registration, password resets, reminders and event sharing the system needs an email service, which may be external. The following characteristics are required:

- supports secure protocols (STARTTLS or pure TLS)
- has good reputation in spam filters
- guarantees availability (99.9%+)

3.4 Design constraints

3.4.1 Standards compliance

Data interchange formats In order to allow interoperation with other calendar applications Travlendar+ will need to support exporting user calendars in iCalendar and CalDAV format. Another custom format such as JSON or XML may be used for exporting data that is not supported by the other two standard formats.

3.4.2 Hardware limitations

Our app doesn't have particular hardware constraints, but we expect that it will be run on modern computers, smartphones or tablets, so that it can satisfy the performance requirements (see section 3.2).

3.5 Software system attributes

3.5.1 Reliability and availability

Nowadays users expect all applications to always work. Even excluding faults to their devices or connections being 100% available and reliable is challenging, as these properties depend on our services and also on many external ones. Single points of failure must be avoided where possible.

Single points of failure Avoiding single points of failure can be achieved by technical design decisions, with techniques such as load balancing and separation of concerns (microservices). In the event an non-fundamental service (e.g. car sharing API) stops working the app should keep functioning, altough in a degraded state (unable to reserve cars, but still able to reserve other services). Of course a tradeoff must be made between reliability and cost/architecture simplicity. The design decisions regarding this are left to the design document.

Testing and development environment Test suites must be written where possible, and code changes must never be deployed if regressions are introduced. Testing of the software must be performed on a staging environment that will be as similar as possible to the production environment, with no access to production databases.

Backups and restore, disaster recovery Automated regular backups of code and data must be performed. Incremental backups of production data must be performed at least every 6 hours and snapshots must be performed at least daily. Data restore and disaster recovery procedures must be documented, automated where possible, and tested at least once every three months on a staging environment different from the production environment.

Infrastructure reliability The chosen hosting provider must give reliability guarantees both via technical means and contractual ones. The hosting location for the production environmente must have proper redundancies in place for power and connectivity in order to guarantee 99.9%+ service level agreement.

3.5.2 Security

The system must be developed following security best practices. This implies technology-agnostic requirements which are detailed here as well as technology-dependent ones which are a concern of the development team.

General reccomendations

- all software must be updated in short time after the release of security advisories
- the principle of least privilege must be followed
- regular security reviews must be performed
- a way to safely report security vulnerabilities must be present
  - a bug bounty program may be evaluated
- no security protocols or algorithms shall be created if a secure standard exists

Data security User data must be stored as securely as possible:

- databases must be password protected and not directly accessible from the public internet
- normal developers should not have direct access to production databases
- data should be stored encrypted at rest
- password must be stored after being hashed with a secure algorithm

Application security Common vulnerabilities must be avoided. This includes vulnerabilities from lists such as OWASP Top 10, or certificate validation errors. The development team must be knowledgeable about the possible pitfalls in the chosen software stack.

Transport security All network traffic must be encrypted with standard algorithms. The reccomendation is TLS with a modern configuration.

Operational security A small group of authorized people must be in charge of deployment and maintainance and only they should have access to the production environment. Where possible deployment procedures should be automated.

Physical security A hosting provider with proper physical security must be chosen. This includes 24/7 surveillance, strong authentication, authorization and accounting.

3.5.3 Maintainability

Code, database schemas and deployment procedures must be documented in order to allow new developers to gain familiarity with the system in a reasonable time.

3.6 User interfaces

Here we include some mockups for the main functionality of the application.

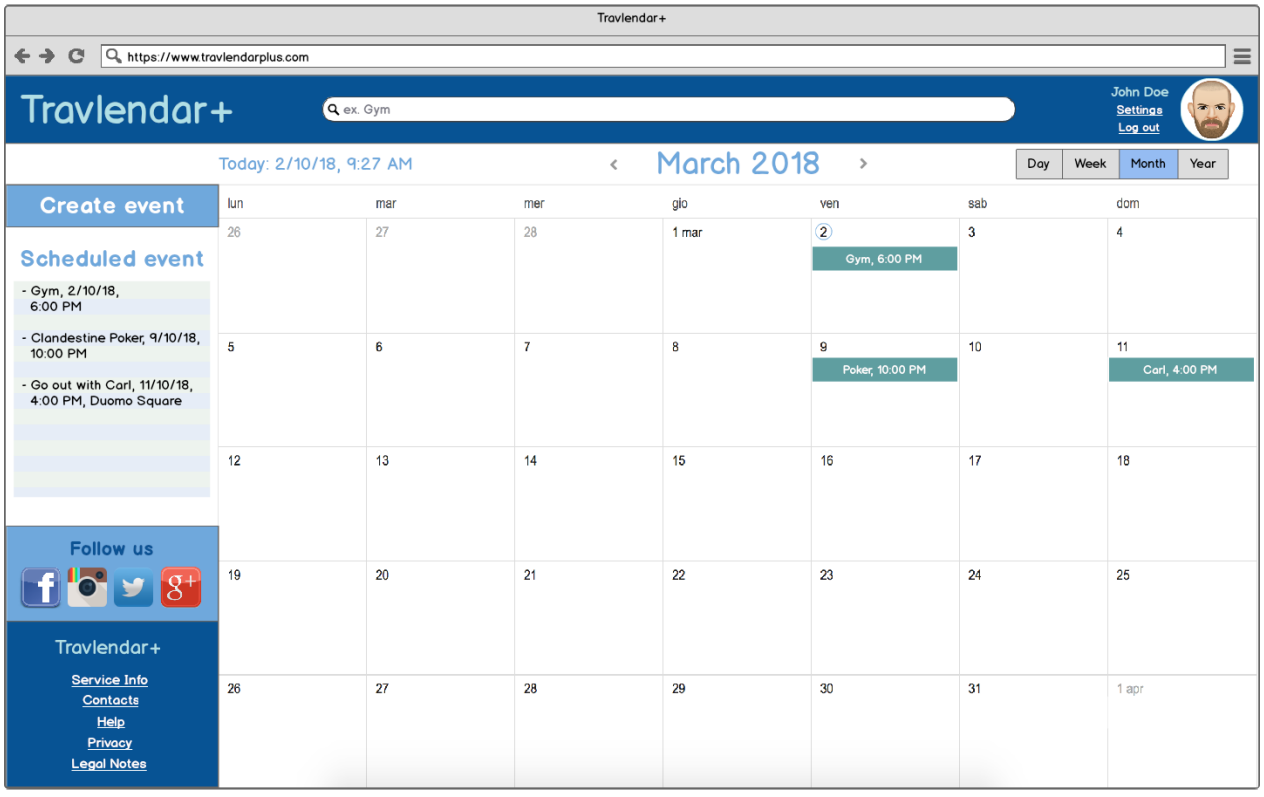

Figure 1: Desktop home page

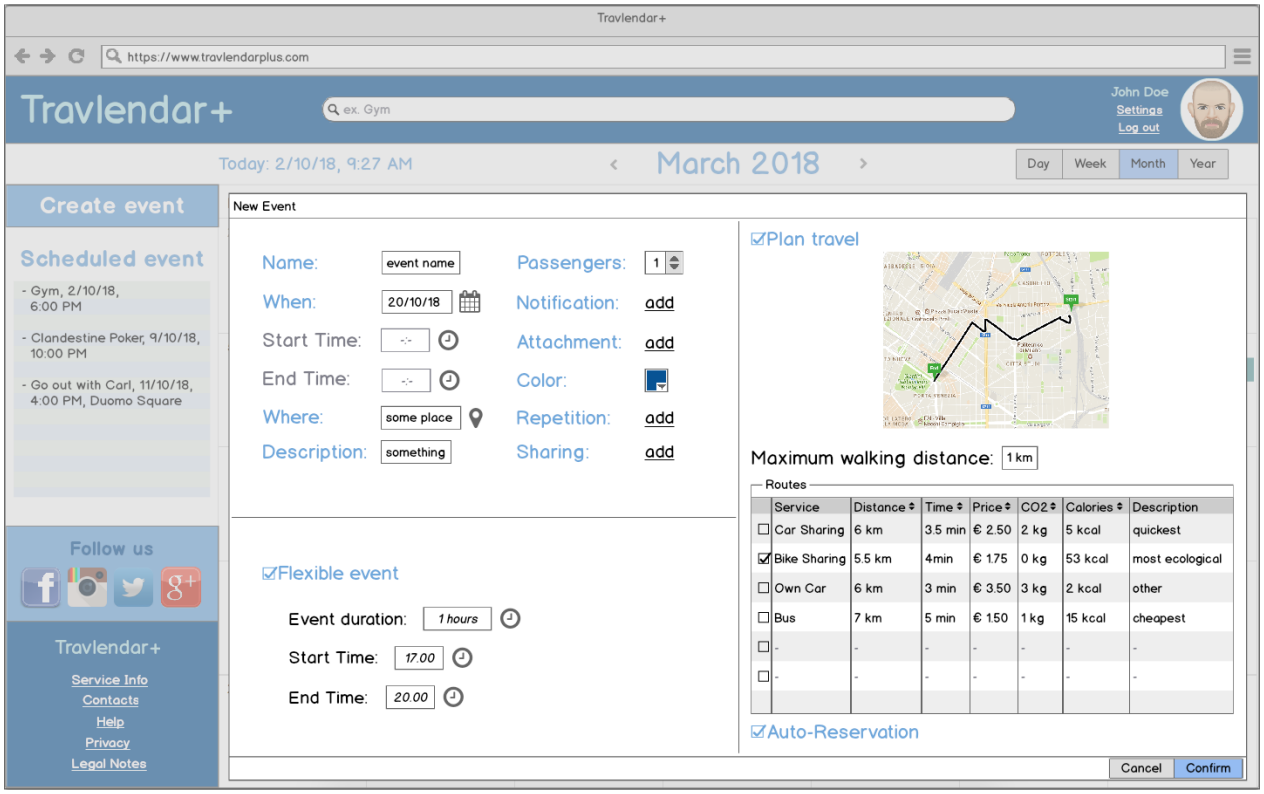

Figure 2: Desktop new event

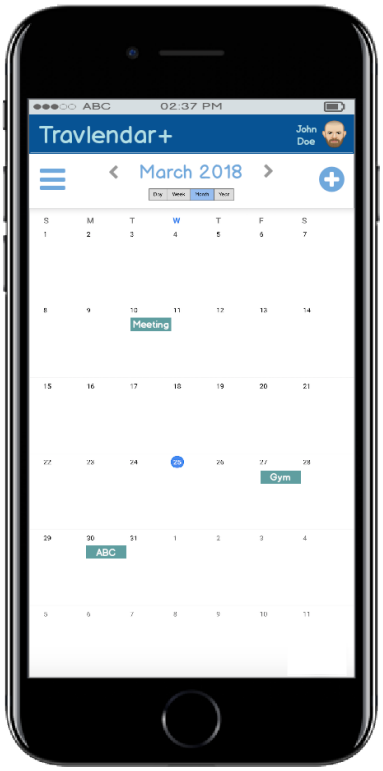

Figure 3: Desktop home page

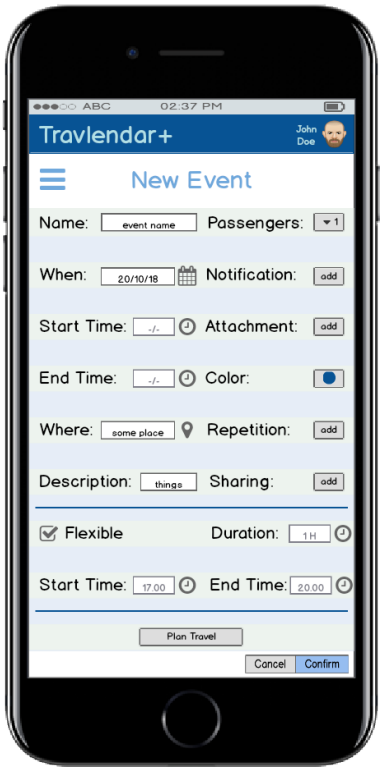

Figure 4: Desktop home page

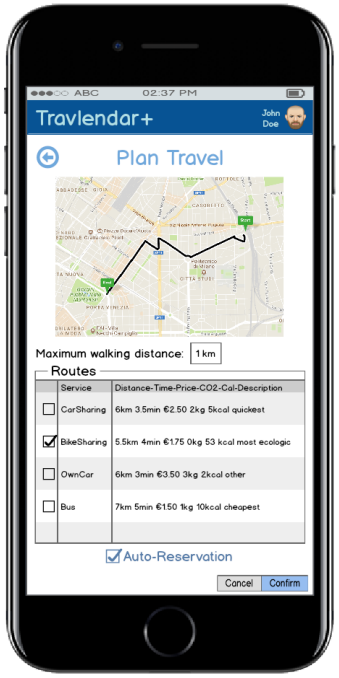

Figure 5: Desktop home page

4 Scenarios

This section will present some possible situations that may occur from the interaction between a user and the system developed.

4.1 Elisabetta is curious about Travlendar+

Lily tells Elisabetta about a new service called Travlendar+: Elisabetta, now curious, tries to register to Travlendar+. She visits the homepage, browses the site a little and then proceeds to the registration page. Elisabetta then enters all the data needed by the system to complete the process: name and surname, email address and password. Then the system sends her a confirmation mail with a verification code used to activate her new account. After she inserts the code, and to complete the sign-up she finally enters the billing information (optionally). She has now completed the registration, and she can log into the system and use all the services given by our app.

4.2 Filippo is getting married!

Filippo gets married and, to live with his wife, he has to move from his parents' house to the new one. So when he insert a new event doesn't want that the app calculates the route from his old house, because it is quite distance from the new one and the expected travel time would be incorrect. He can access to his personal private area (after having logged in) and, by clicking "Edit Personal Data", he inserts the new data in the system, and finally he saves the changes. Filippo can now use the app again and stay happy with his wife (having some children).

4.3 Alessandro is always drunk

Alessandro is a good boy, but he has one really bad habit: on Sunday he goes out with his friends and he drinks a lot; on Monday he has his duties to do but, having an hangover, he forgets some things. Fortunately, he has an account on Travlendar+, so on Friday he makes a note of all his appointments on the calendar. The application not only reminds him his scheduled events with a notification, but also it helps him to schedule his travel. So, on Monday Alessandro will arrive in time at work and his department boss won't fire him. Thanks Travlendar+!

4.4 Fabrizio, the sport addicted

Fabrizio is a sport addicted and he really belives in the problem of air pollution, so he prefers not to use car or motor vehicles. In his preference he can sets to select always the most ecologic route, so that he can runs or cycles around the city. He does sport and doesn't pollute the air! But if he is in hurry because he has an appointment with his girlfriend (and if he arrives late she will stay angry for the whole day), when he shedueles this event, he can choose the quickest route. An exception can be made for just one time: it is for a good reason!

4.5 Michele, the motobiker

Michele, is in love with his motorbike. He moves all around Milan with it, so that he can avoid traffic congestion because he is always in hurry. He is a very busy man especially near lunch time, but he wants to eat everyday between 12 am and 2 pm, so he schedules lunch as a flexible and recurrent event. The app day by day will manage Michele's lunch itself (choosing the best moment during the interval of time that Michele has set) in order to give him the time that he has set in his preference. If Michele doesn't have enough time between 12am and 2 pm to have lunch, the application will notify him this fact and then Michele can choose whether have a shorter lunch or delete (or modify) another event. Poor Michele!

4.6 Luca and his birthday

Luca is a very raver boy. He really like party and for his birthday he wants to invite a lot of people. Luca is looking for an application that lets him to share this event to his friends. Fortunatly, browsing on a search engin he finds out Travlendar+. He now can create a big shared party, maybe a lot expensive for him, but this birthday will never be forgotten. So he create a shared event invinting all his friends.

4.7 Andrea is offline

Andrea has a very bad memory and every months he forgot to top-up his phone, so if he finishes the credit he must remain offline for a day. When he has to add to Travlendar+ a new event, even if he is offline, he can do it (otherwise he would forget also the appointment!) The only limitation for Andrea is that the application when the device is offline won't calculate the route or tell him the weather forcast.

5 Models

In this section we are going to abstract from the previously seen scenarios in order to have a more high-level description. For this purpose we will use UML (Unified Modeling Language) diagrams.

5.1 Use cases model

From previously denoted scenarios and from the whole analysis we did in this document, we individuated the use cases of the system to be developed. In these pictures there are some use case diagrams which represent actors, their use cases and their interactions.

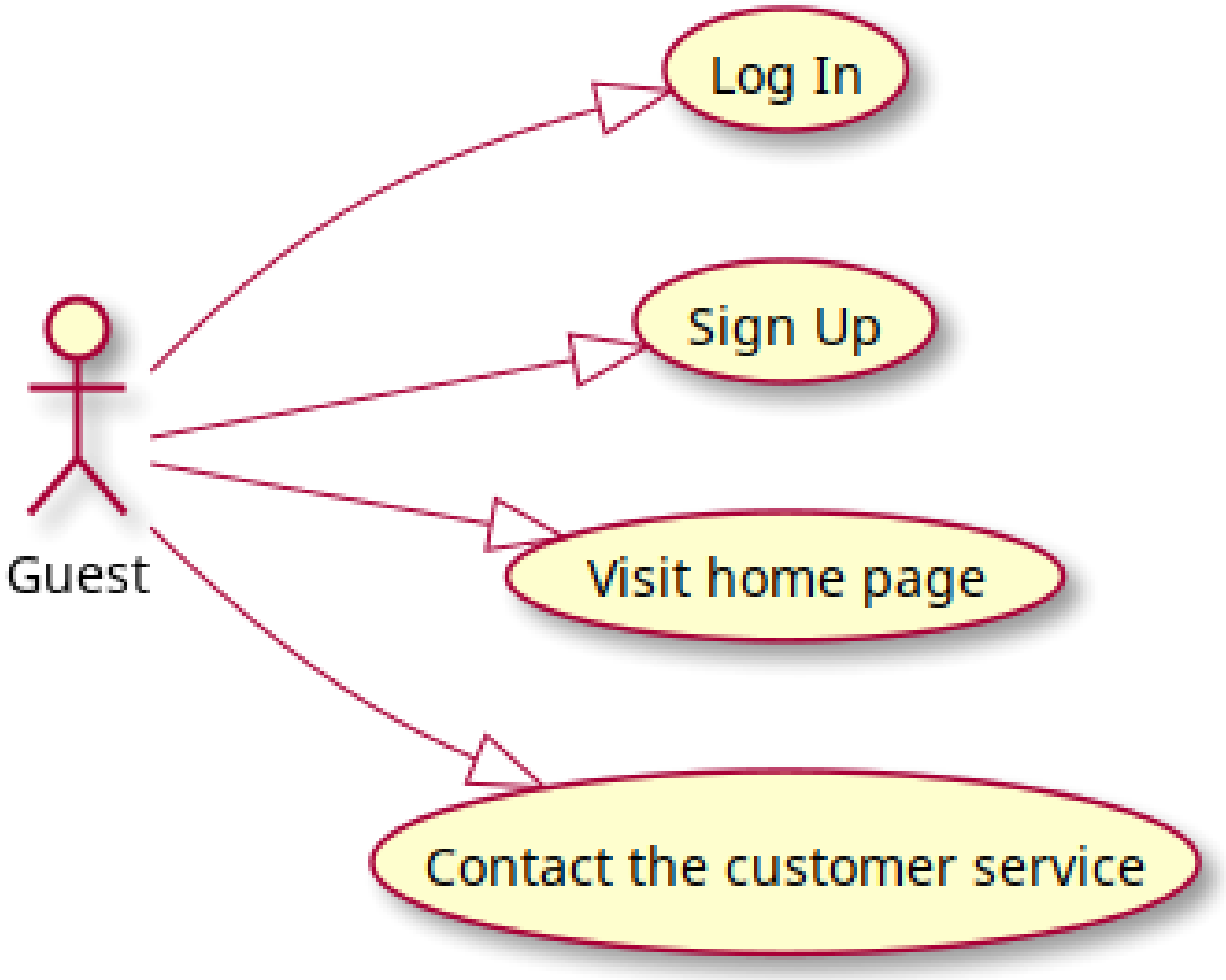

Figure 6: Guest Use Case

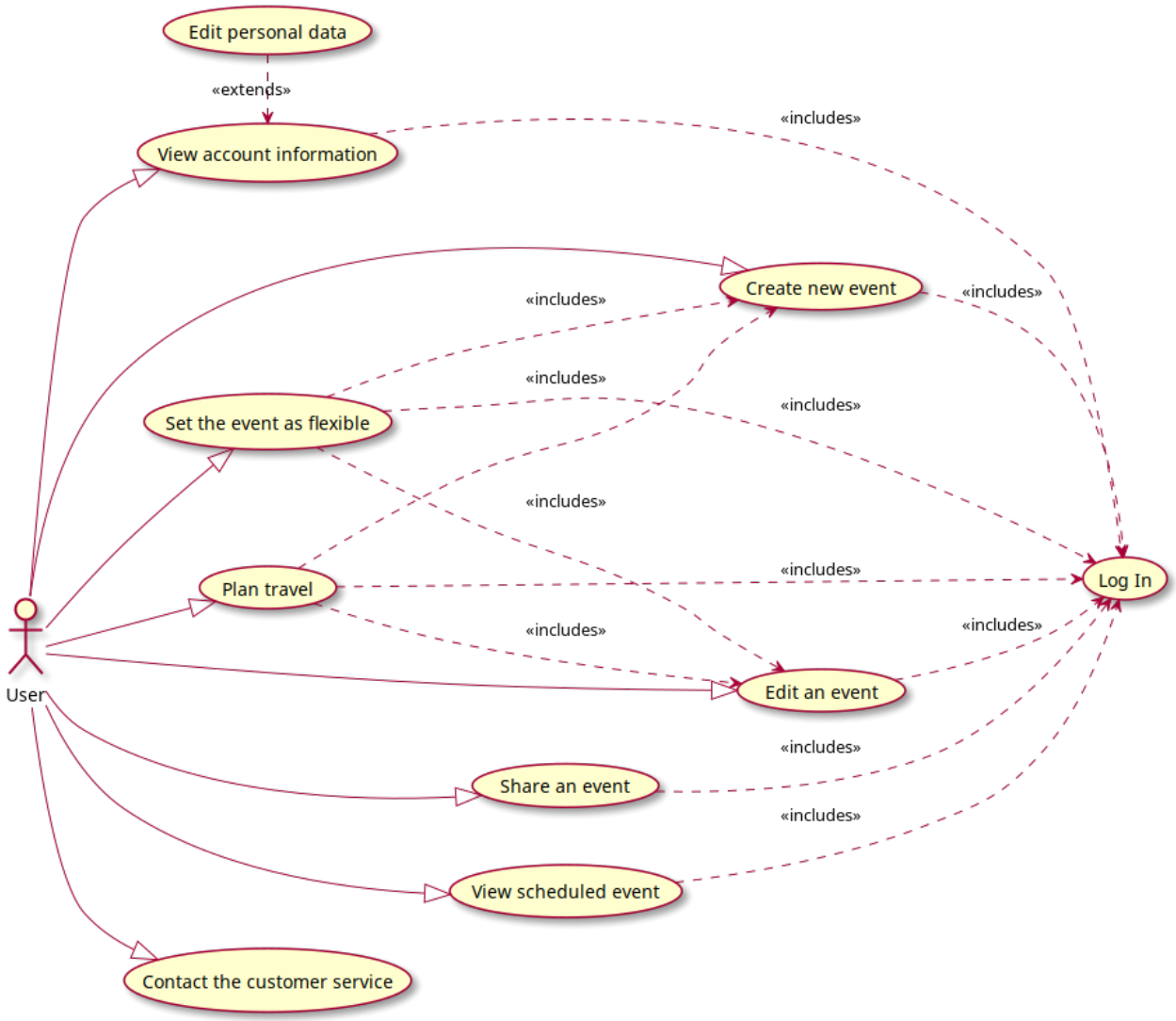

Figure 7: User Use Case

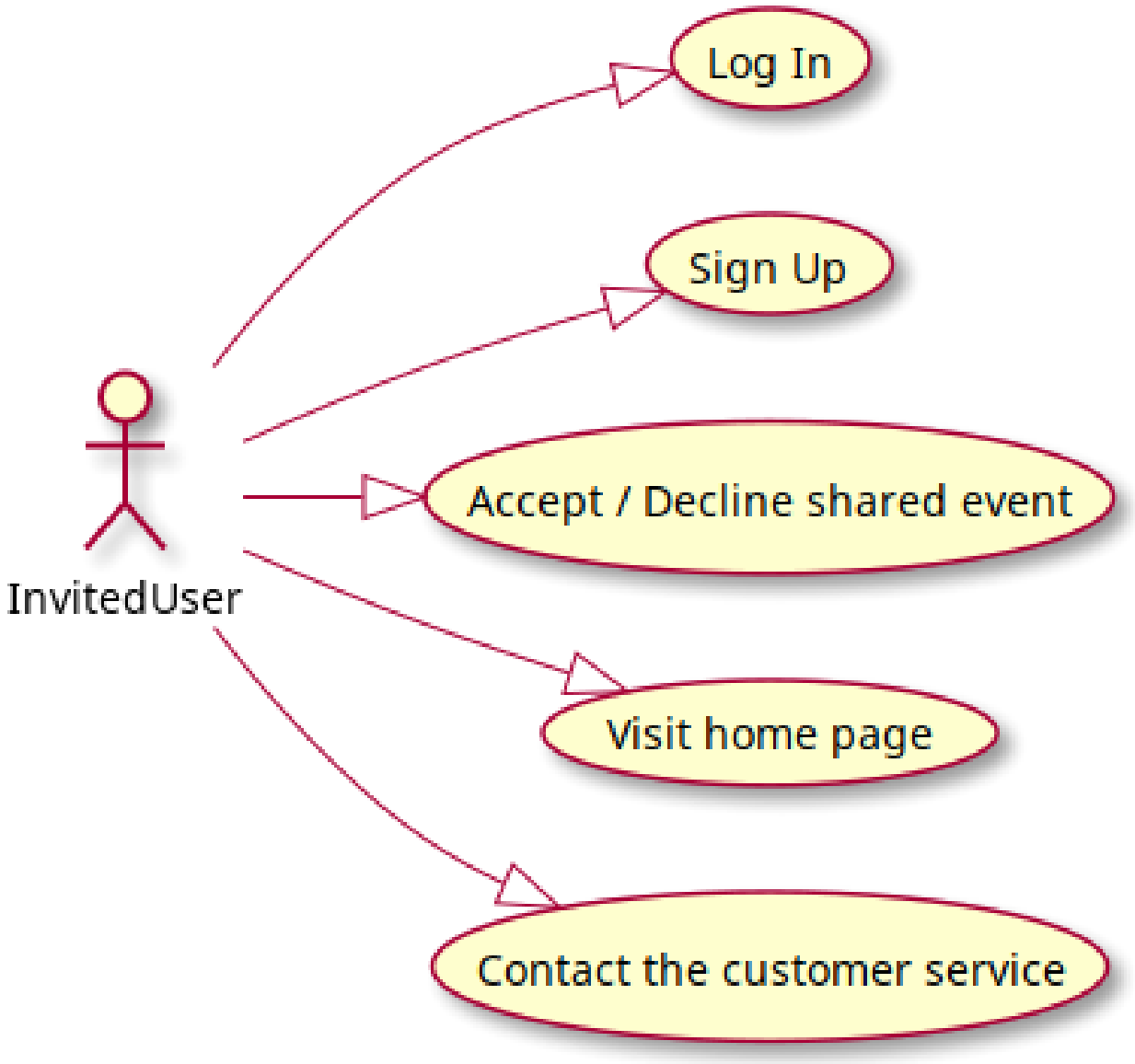

Figure 8: Invited User Use Case

5.2 Use case table

The following tables analyze the interaction between an actor and the application and how the system should work from the user perspective. For semplicity, here we assume that the actor is using the web application.

5.2.1 Signup

Actors	Guest
Preconditions	The guest has a working Internet connection and he has not registered an account yet.
Events	1. The guest reaches the registration page 2. The system requires the guest to enter all his/her personal information 3. The guest types the requested information and presses the 'Next' button 4. The system verifies the uniqueness of the email, the equality of the twice typed passwords 5. The system send an email to the specified address with a verification code 6. The user insert the verification code and press "Next" 7. The system shows to the guest a recap of the information already provided and requires him to confirm them, to read and accept the EULA and complete the registration 8. The guest ticks the "I read carefully, and I accept the general conditions" box and presses the "Confirm and Complete Registration" button 9. The system sends an email to notify the correct registration to the guest 10. The system reports the registration and redirects the user to the main page 11. The system ask the User if he want to be redirected to an external service web page to enter his billing information 12. If the User accept, the system wait to a confirmation from the external payment service about the correctness of the billing information and then notify him the conclusion of the process 13. Else the system notifies the user that he will need to insert the billing information the first time he will buy a ticket or pass in the app.
Postconditions	The user has signed up.
Exceptions	The email the guest typed has been already used. The second password does not match with the first one. One of the mandatory fields is empty. The user inserts a wrong verification code. The code sent by email arrives in the guest's spam inbox. The guest did not accept the EULA. In these cases the system notifies the error and cannot complete the registration. The external payment service didn't confirm the billing information. In this case the user can insert the billing information later.

5.2.2 Login

Actors	User
Preconditions	The actor has a working Internet connection and is already registered.
Events	1. The actor reaches the log in page 2. The system requires the actor to enter his email and password 3. The actor types the requested information and press the 'Log in' button 4. The system verifies the correctness of the email and password 5. The system redirects the actor to the main page
Postconditions	The actor is logged in.
Exceptions	The email or the password the actor typed are not correct. One of the fields is empty. In these cases the actor can't complete the log in. The system notifies the error and cannot complete the login.

5.2.3 Edit personal information

Actors	User
Preconditions	The actor has a working Internet connection and he is already logged into the system.
Events	1. The actor reaches his personal area 2. The actor clicks the “Edit personal information” button 3. The system allows the actor to change his settings including address, travel preferences, email or password (and to confirm it) 4. The user enters the new information and presses ’Save’ 5. The system verifies the correctness and the completeness of the information 6. The system shows the actor his updated information
Postconditions	The user has modified his personal information.
Exceptions	One of the mandatory fields is empty. The second password does not match with the first one. In these cases the system notifies the error and cannot complete the request.

5.2.4 Add/Edit travel preferences

Actors	User
Preconditions	The actor has a working Internet connection and he is already logged into the system.
Events	1. The actor reaches his personal area 2. The actor clicks the “Edit travel preferences” button 3. The system allows the actor to change his settings about usable vehicles, shared vehicles, maximum walking distance, transportation tickets/passes 4. The user enters the new information and presses ’Save’ 5. The system verifies the correctness and the completeness of the information 6. The system shows the actor his updated information
Postconditions	The user has modified his travel preferences.
Exceptions	Tickets or passes are invalid or expired.

5.2.5 Edit billing information

Actors	User
Preconditions	The actor has a working Internet connection and he is already logged into the system.
Events	1. The actor reaches his personal area 2. The actor clicks the “Edit billing information” button 3. The system notifies the actor that he is going to be redirected to an external service web page where he will be required to enter his new billing information 4. The system receives a confirmation from the external payment service about the correctness of the new billing information 5. The system shows the actor his updated information 6. The system sends an email as a notification of the changement
Postconditions	The user has modified his billing information.
Exceptions	The external payment service did not confirm the billing information. In this case the system notifies the error and cannot complete the request.

5.2.6 Buy a ticket/pass

Actors	User
Preconditions	The actor has a working Internet connection and he is already logged into the system.
Events	1. The actor reaches his personal area 2. The actor clicks the “Buy tickets / passes” button 3. The actor chooses what kind of ticket / pass he wants to buy 4. The system requires a final confirmation 5. The actor confirms the purchase 6. The system shows a summary page 7. The system sends an email as a notification of the purchase
Postconditions	The user has bought one or more tickets / passes.
Exceptions	The external payment service cannot complete the payment. In this case the system notifies the error and cannot complete the request.

5.2.7 Create a new event - Scenario 1 (Offline event)

Actors	User
Preconditions	The user is already logged into the system
Events	1. The actor clicks the new event button 2. The actor enters event name, hour, place and other standard information 3. The system verifies the correctness and the completeness of the information 4. The actor confirms the event creation clicking the “Confirm” button 5. The system shows a summary page 6. The application won’t let the user to choose the route or to share the event with other people
Postconditions	The actor has a new event in his scheduled events.
Exceptions	One of the fields of the basic information is empty. The starting hour is after the ending one. In these cases the system notifies the error to the user and he cannot create the event.

5.2.8 Create a new event - Scenario 2 (Standard event)

Actors	User
Preconditions	The user has a working Internet connection and he is already logged into the system
Events	1. The actor clicks the new event button 2. The actor enters event name, hour, place and other standard information 3. The system verifies the correctness and the completeness of the information 4. The actor confirms the event creation clicking the “Confirm” button 5. The system shows a summary page
Postconditions	The actor has a new event in his scheduled events.
Exceptions	One of the fields of the basic information is empty. The starting hour is after the ending one. In these cases the system notifies the error to the user and he cannot create the event.

5.2.9 Create a new event - Scenario 3 (Choose mobility sharing)

Actors	User
Preconditions	The user has a working Internet connection and he is already logged into the system
Events	1. The actor clicks the new event button 2. The actor enters event name, hour, address and other standard information 3. The system verifies the correctness and the completeness of the information 4. The actor clicks the “Plan travel” button 5. The system shows the travel suggestions 6. The actor chooses one of the proposed routes 7. The actor clicks the “Sharing transporation means” button 8. The actor clicks the “auto-reservation function” button 9. The actor chooses one or more bike or car sharing services through the buttons 10. The actor confirms the event creation clicking the “Confirm” button 11. The system shows a summary page
Postconditions	The actor has a new event in his scheduled events.
Exceptions	One of the fields of the basic information is empty. The starting hour is after the ending one. The address does not exist. In these cases the system notifies the error to the user and he cannot create the event.

5.2.10 Create a new event - Scenario 4 (Flexible event)

Actors	User
Preconditions	The user has a working Internet connection and he is already logged into the system
Events	1. The actor clicks the new event button 2. The actor enters event name, hour, place and other standard information 3. The system verifies the correctness and the completeness of the information 4. The actor clicks the “Flexible event” button 5. The actor inserts the event duration 6. The actor inserts the range of time 7. The actor confirms the event creation clicking the “Confirm” button 8. The system shows a summary page
Postconditions	The actor has a new flexible event in his scheduled events.
Exceptions	One of the fields of the basic information is empty. The starting hour is after the ending one. In these cases the system notifies the error to the user and he cannot create the event.

5.2.11 Accept/Decline a shared event with invitation

Actors	User, Invited User
Preconditions	The guest/user has a working Internet connection and he had received an email containing an invitation to a shared event
Events	1. The actor can click on the link given in the email to get further information about the shared event 2. The system ask to the actor to signup/login, to be able to add to the actor's calendar the event and to help him to plan the travel, but this is not necessary 3. The actor can click on "Accept", "Decline" or "Maybe" buttons (the default decision is "Maybe") 4. The system notifies the event creator about the actor's decision 5. If the actor chooses "Accept" or "Maybe" the system will send a reminder to the actor
Postconditions	The actor can choose how to do with an event shared whit him.
Exceptions	The actor fail signup/login. In this case the user can retry to do login/signup

5.2.12 Edit an event

Actors	User
Preconditions	The user has a working Internet connection, he is already logged into the system and he has at least one event in his scheduled event.
Events	1. The user clicks on a specific scheduled event 2. The system allows the actor to change event name, hour, place and the other standard information 3. The user enters the new information and presses 'Save' 4. The system verifies the correctness and the completeness of the information 5. The system shows the actor his updated information
Postconditions	The user has modified his personal information.
Exceptions	One of the fields of the basic information is empty. The starting hour is after the ending one. The address does not exist. In these cases the system notifies the error to the user and he cannot modify the event.

5.2.13 Delete an already scheduled event

Actors	User
Preconditions	The user has a working Internet connection, he is already logged into the system and he has at least one event in his scheduled event.
Events	1. The user clicks on a specific scheduled event 2. The user has to click on “Delete event” 3. The system shows a confirmation window 4. If the user accepts, the event will be deleted, else nothing will happened 5. If the event was scheduled as recurrent, the system will ask whether the user wants to delete only the current or all the events. 6. If the event was shared the system will notify via email to all the other users the deletion
Postconditions	The event selected by the user is deleted.
Exceptions	

5.2.14 Contact customer service

Actors	User, Guest, Invited User
Preconditions	The actor has a working Internet connection.
Events	1. The actor reaches the “Contact the Customer Service” page 2. The system requires the actor to enter his own name, surname, mobile phone number, email address and to explain his problem in less than 1000 words 3. The user types the requested information and press the ‘Contact the Customer Service” button 4. The system redirects the request of the user to the customer service.
Postconditions	The actor successfully contacted the customer service.
Exceptions	The actor uses more than 1000 words to explain his request. There is at least one empty field. In these cases the system notifies the error and cannot complete the request.

5.3 State Charts

In this section are presented the state charts diagram of the basic action (high level view from the users' perspective) that an actor can do. To see a more in depth view of what the application can do and how it works, see section 5.4

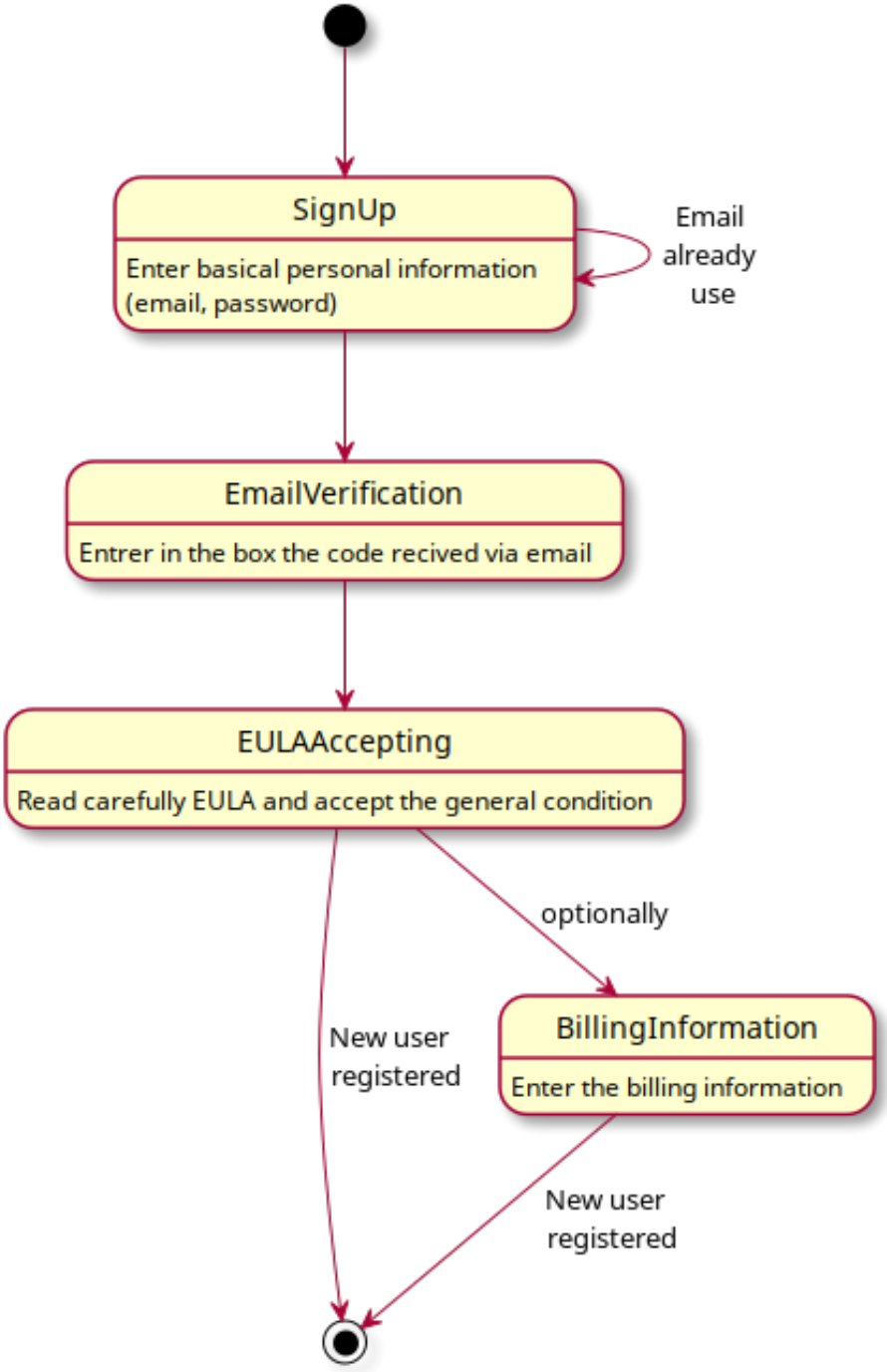

Figure 9: Signup

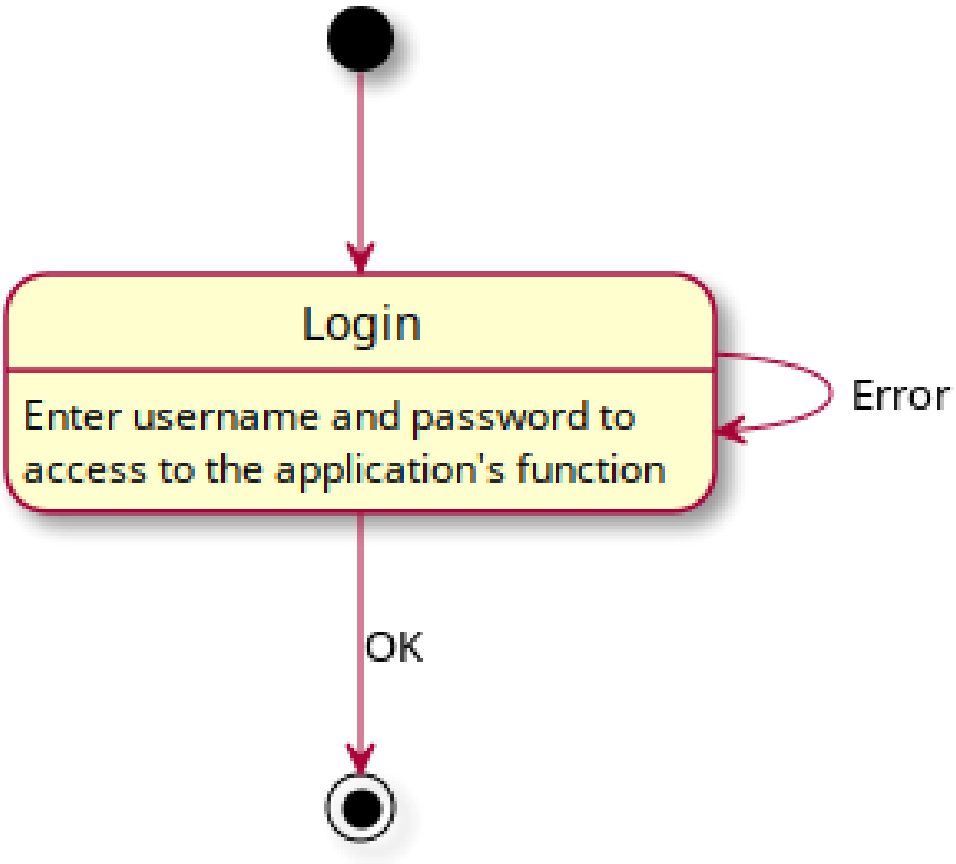

Figure 10: Login

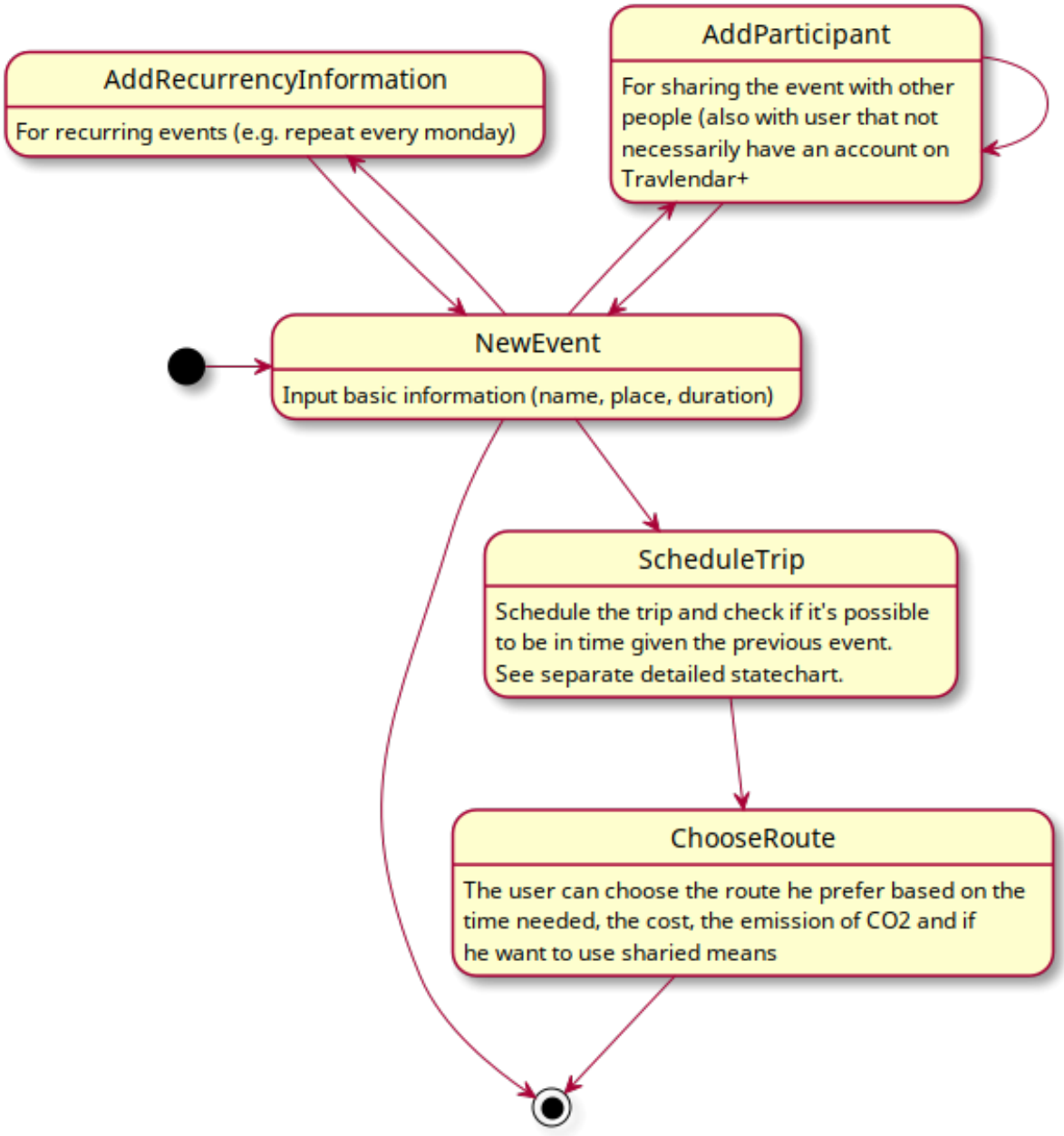

Figure 11: New event

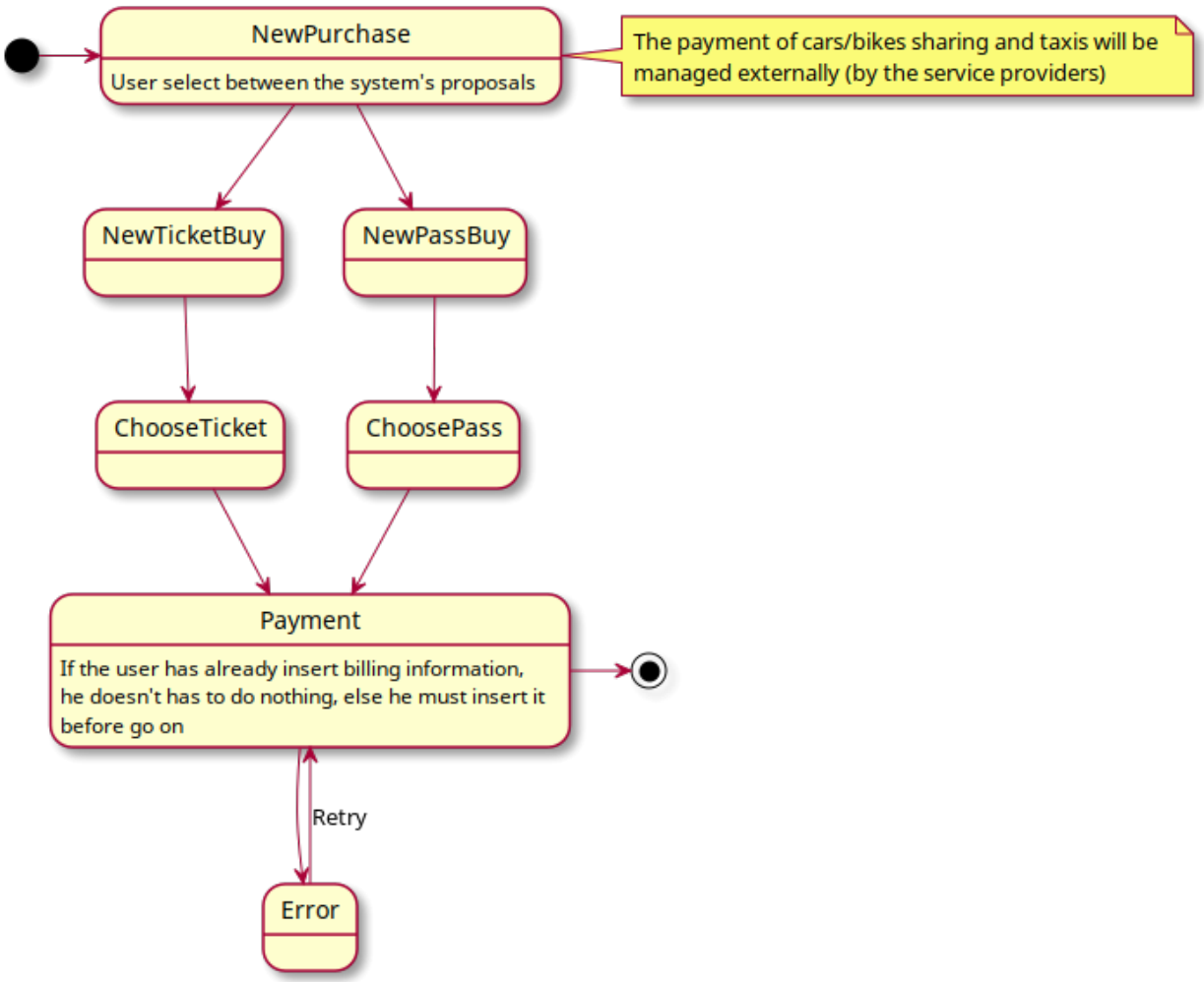

Figure 12: New purchase

5.4 Sequence Diagram

This section presents the sequence diagram of the most important interaction, in order to have a dynamic sight of the main entities too.

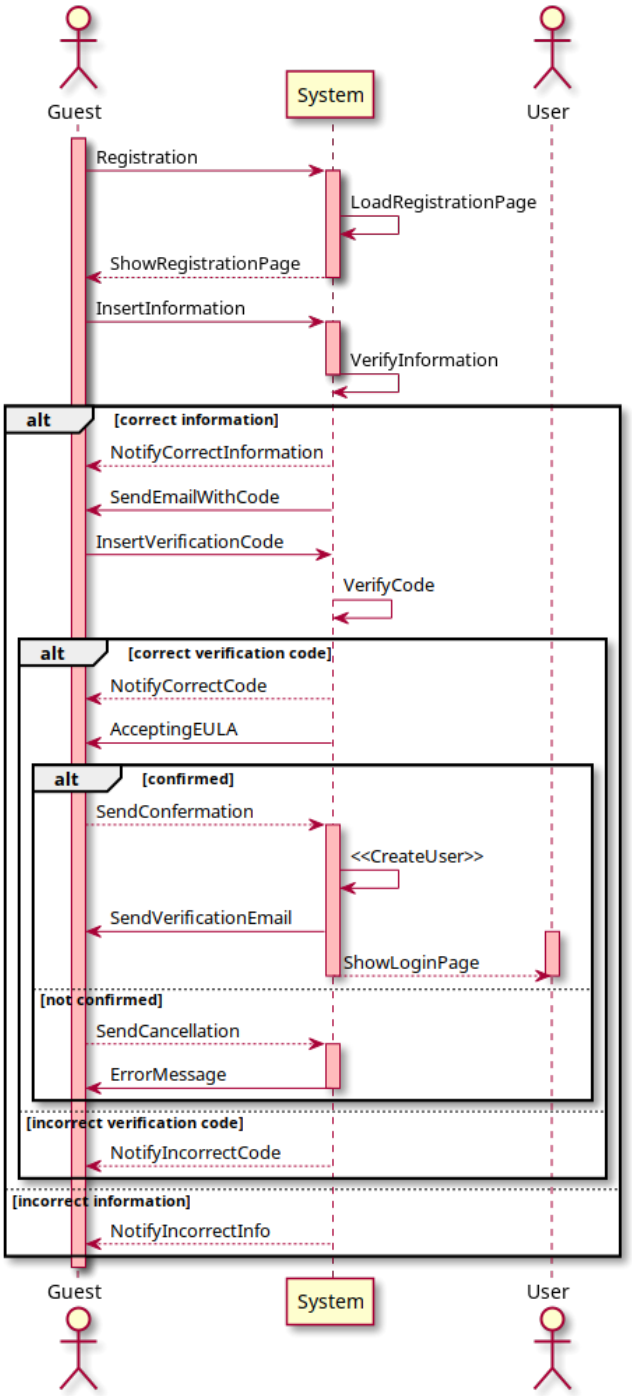

Figure 13: Signup

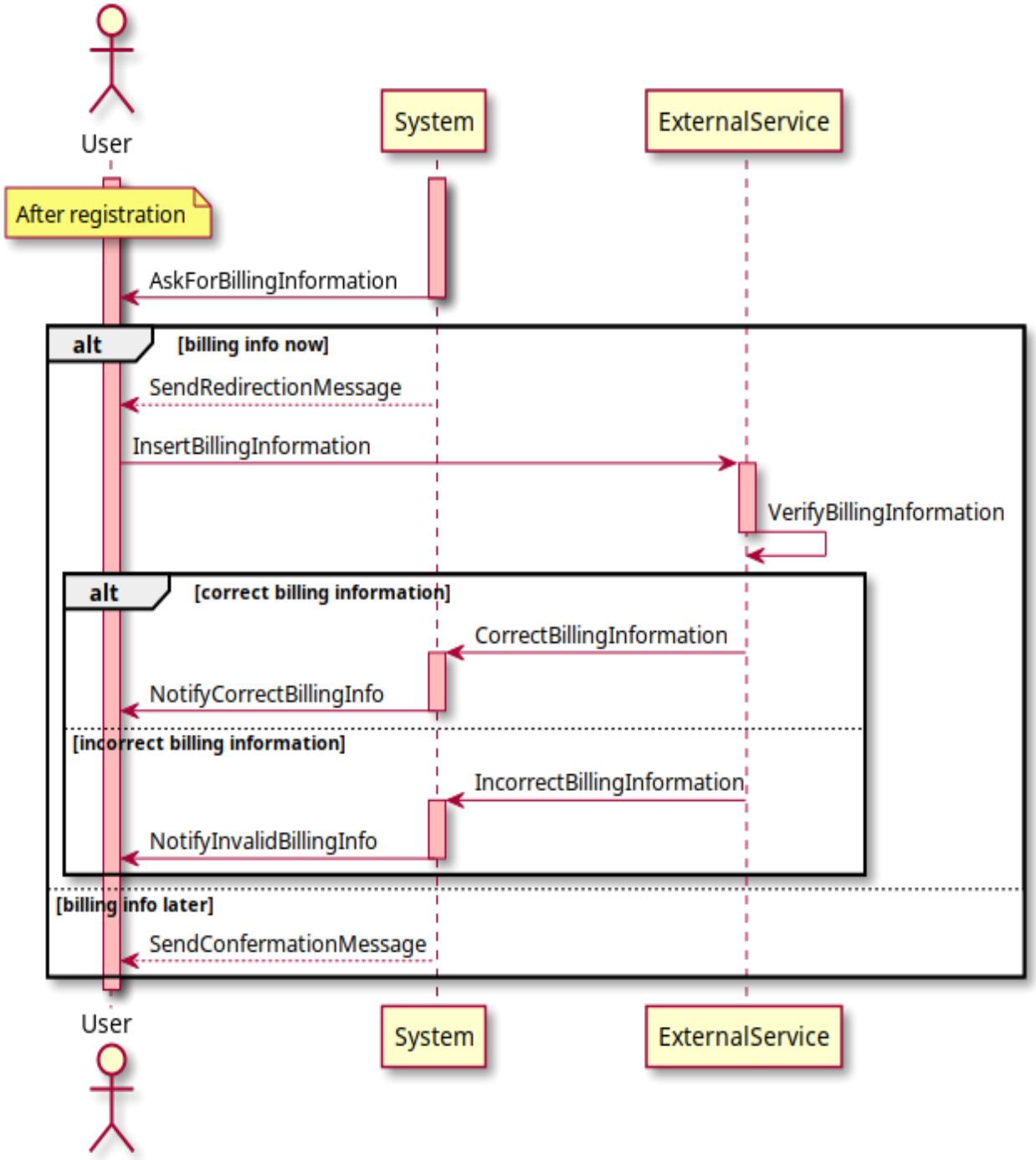

Figure 14: Insert billing information after signup

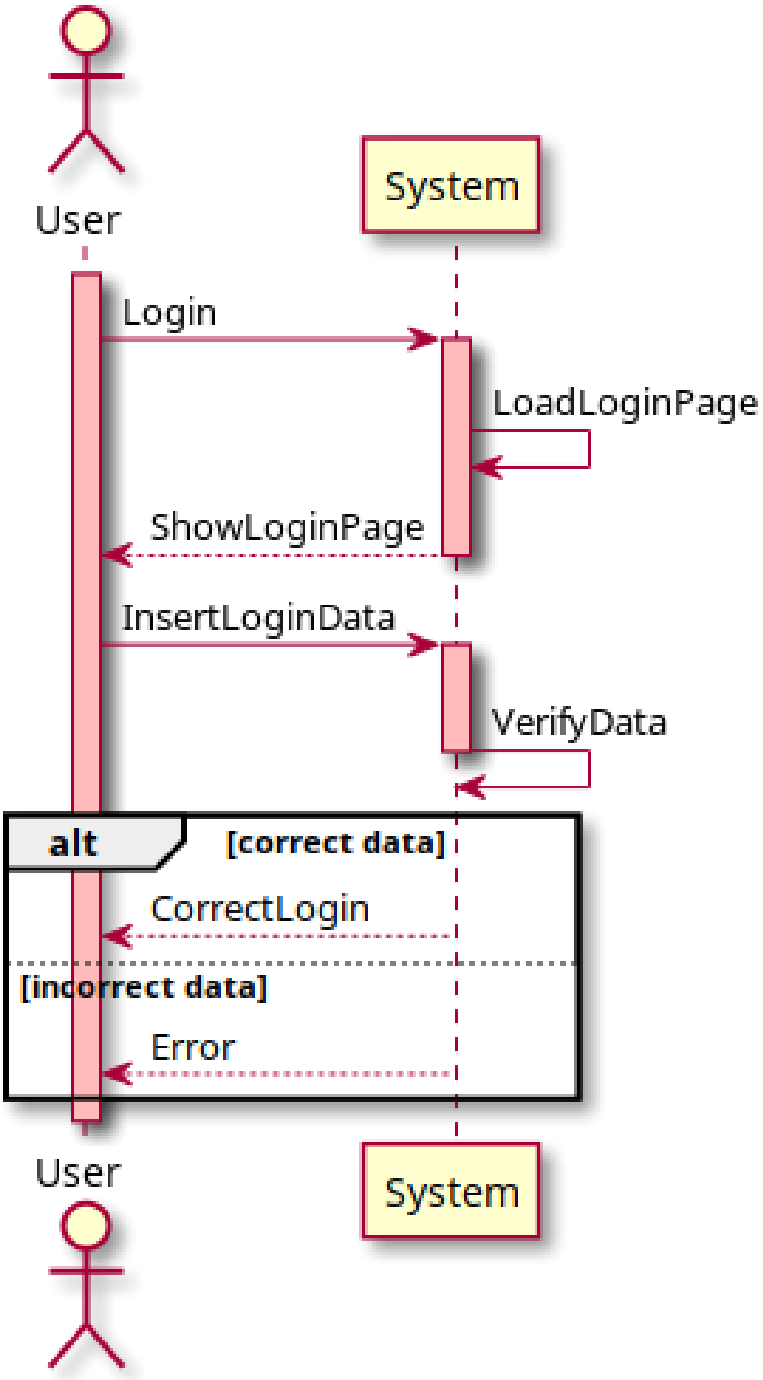

Figure 15: Login

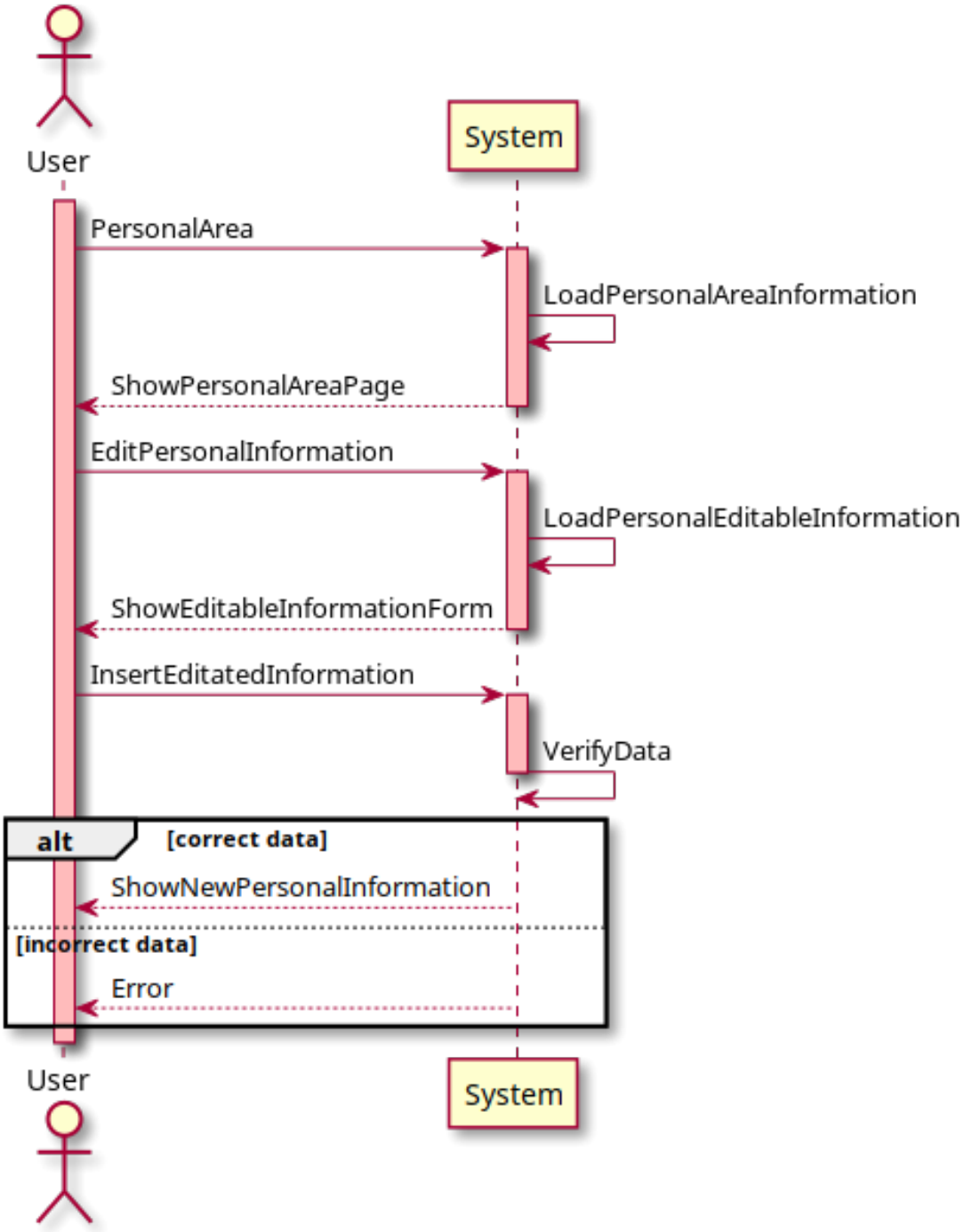

Figure 16: Edit personal information

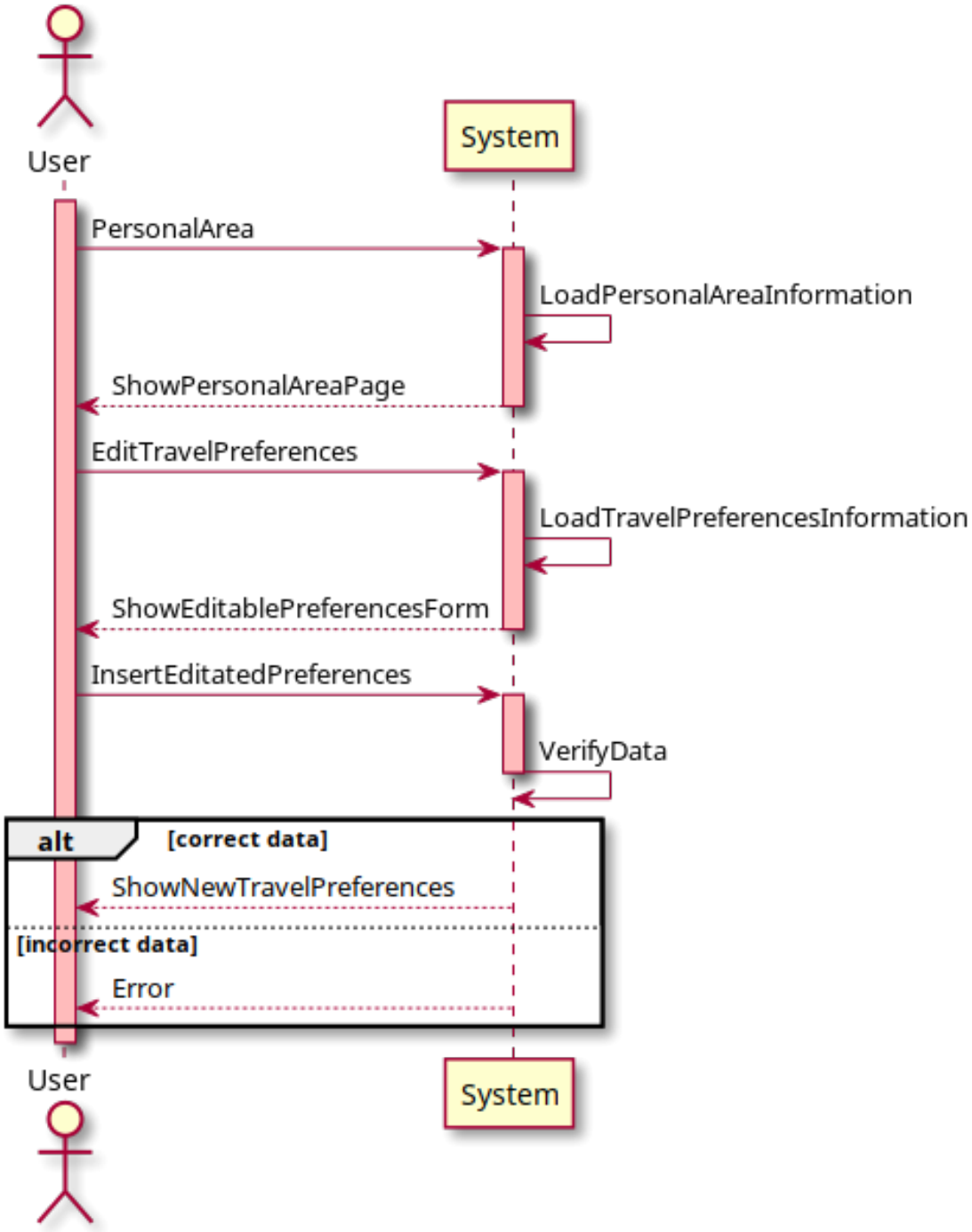

Figure 17: Add/Edit travel preferences

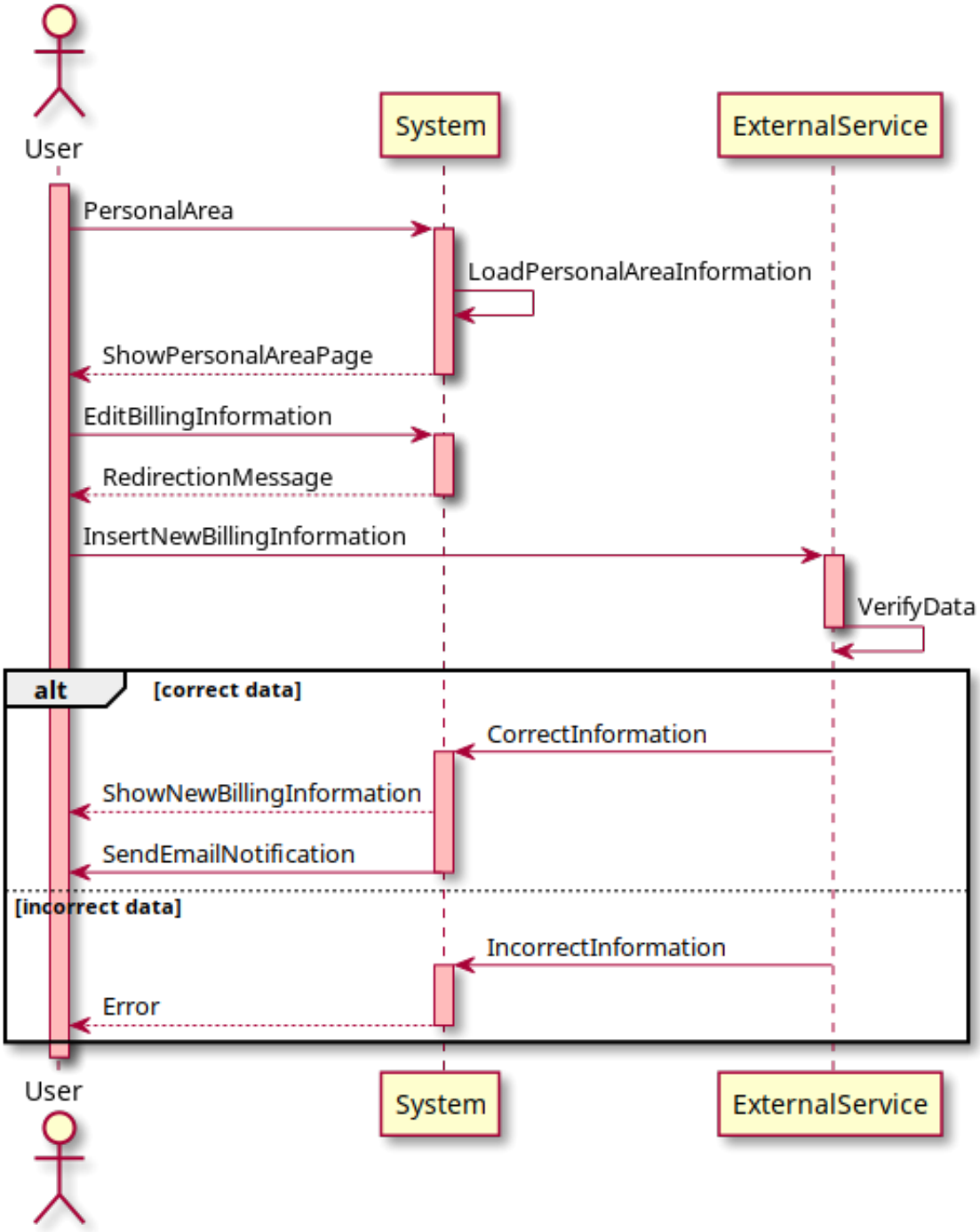

Figure 18: Edit billing information

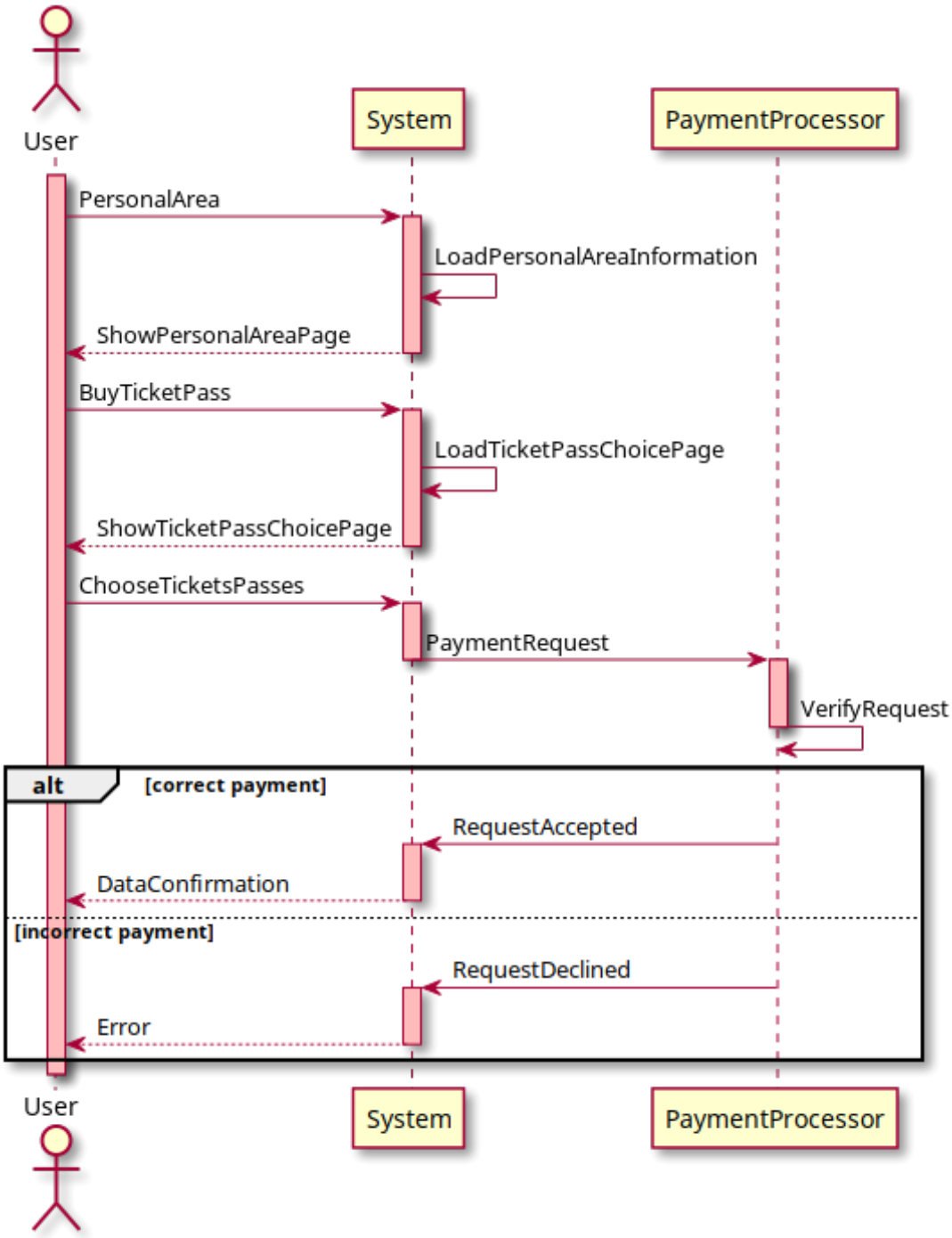

Figure 19: Buy Ticket/Pass

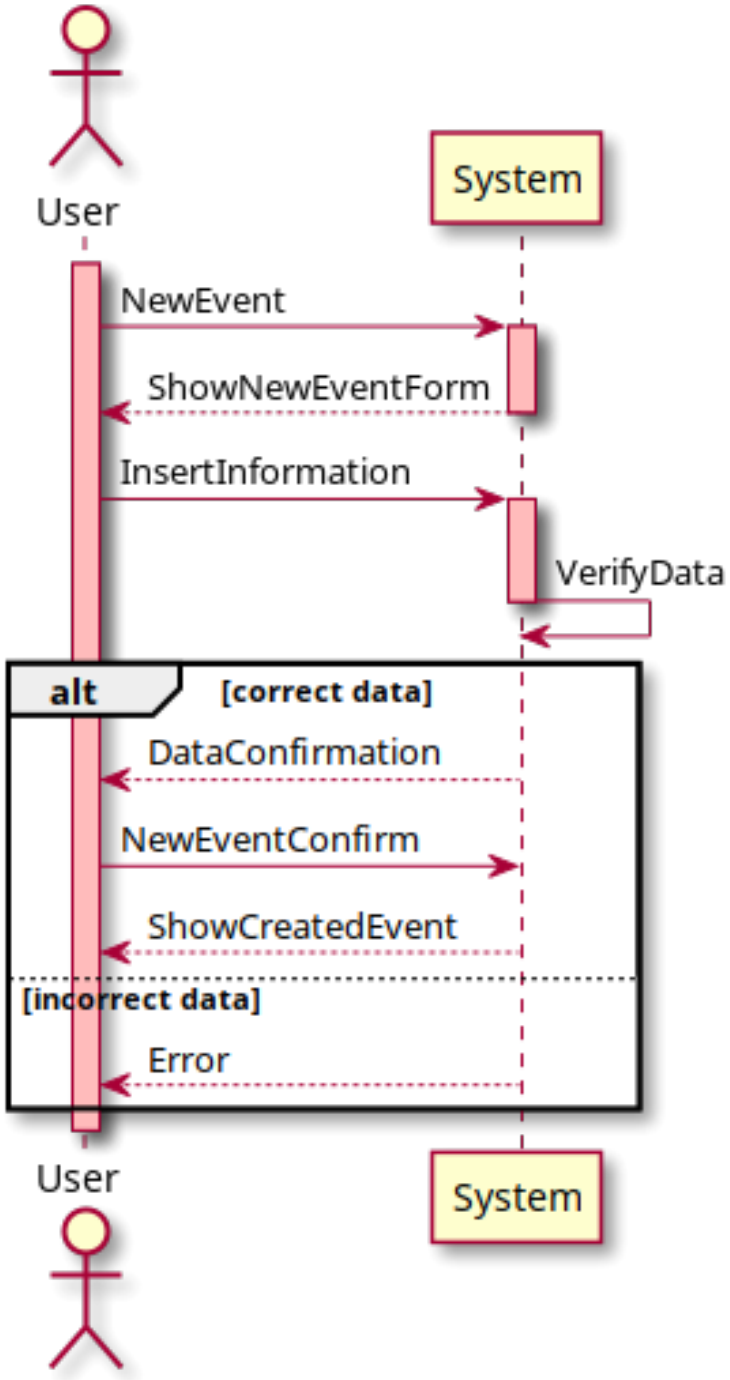

Figure 20: New standard event

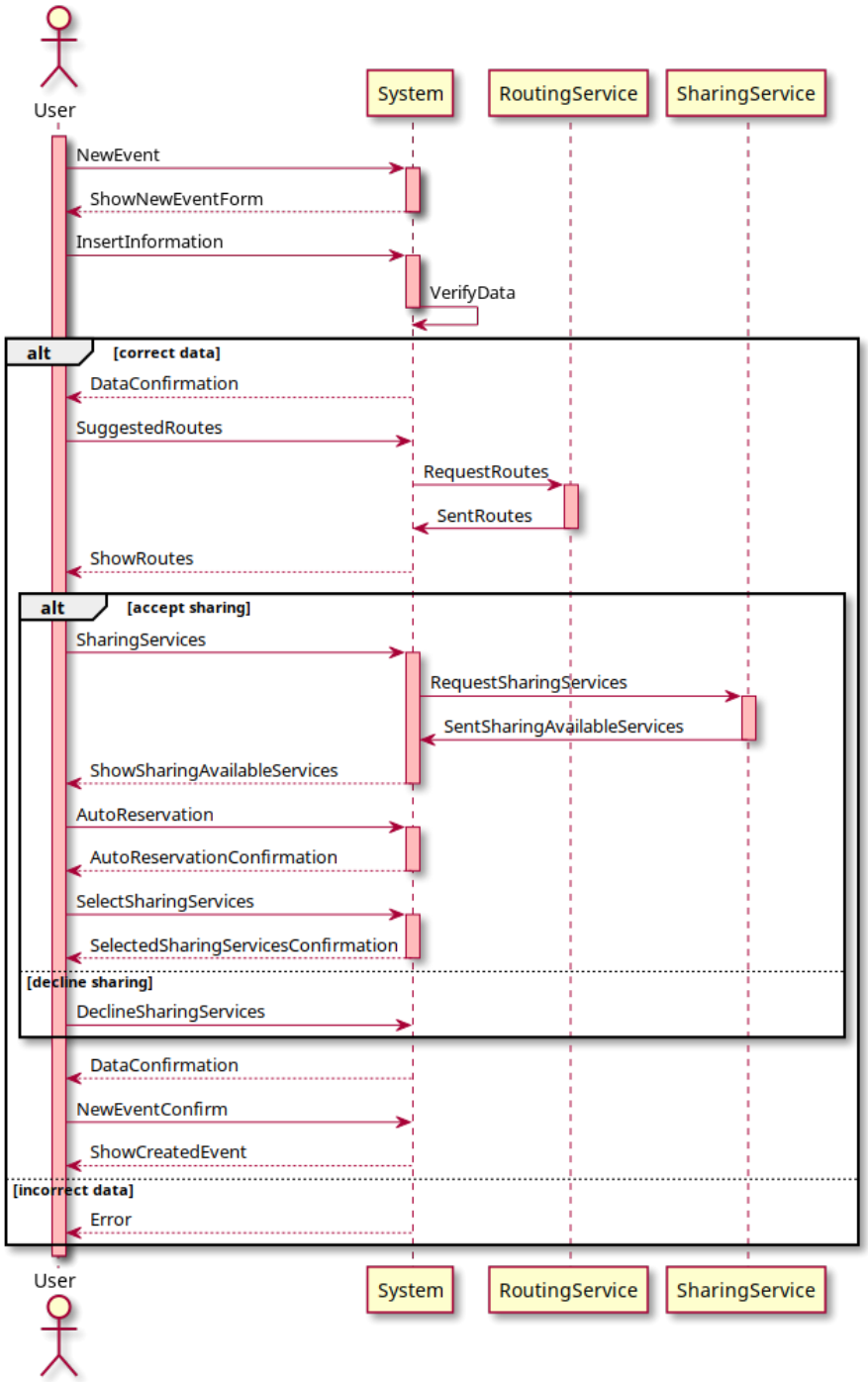

Figure 21: New event with sharing services

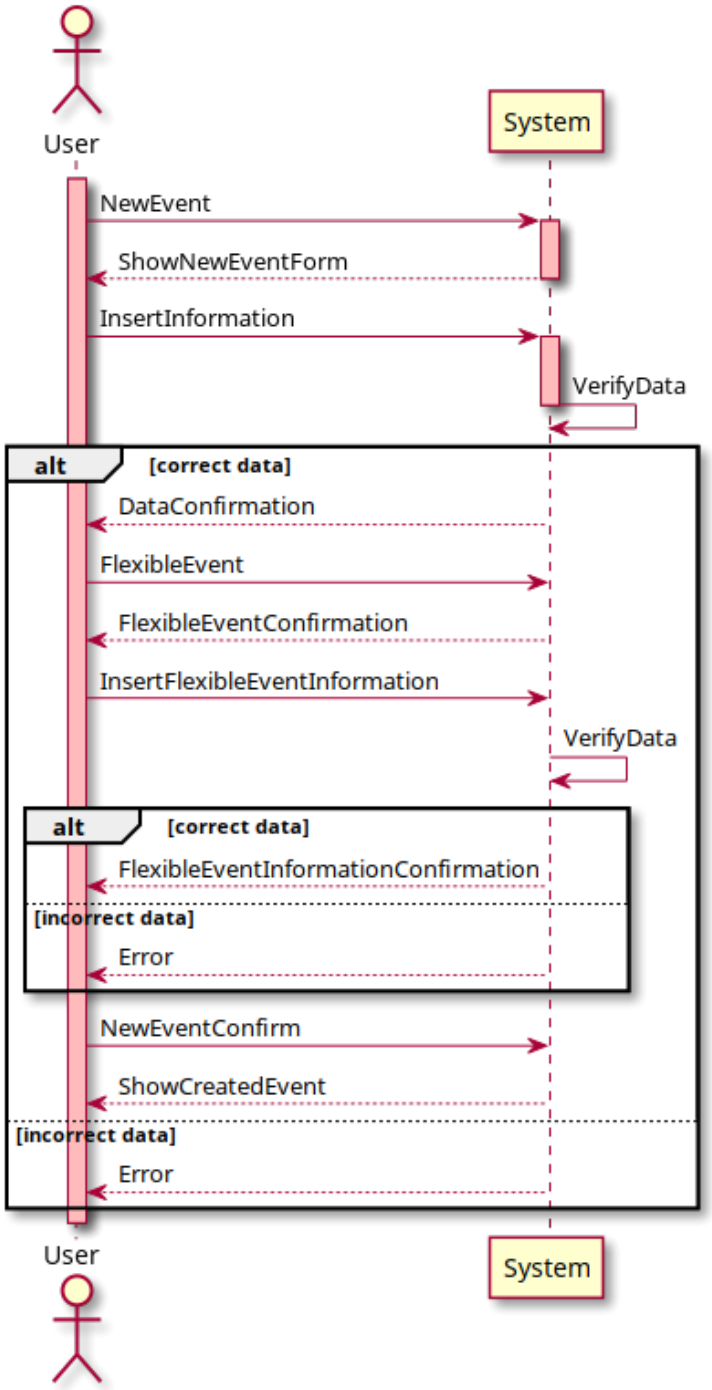

Figure 22: New event with flexible option

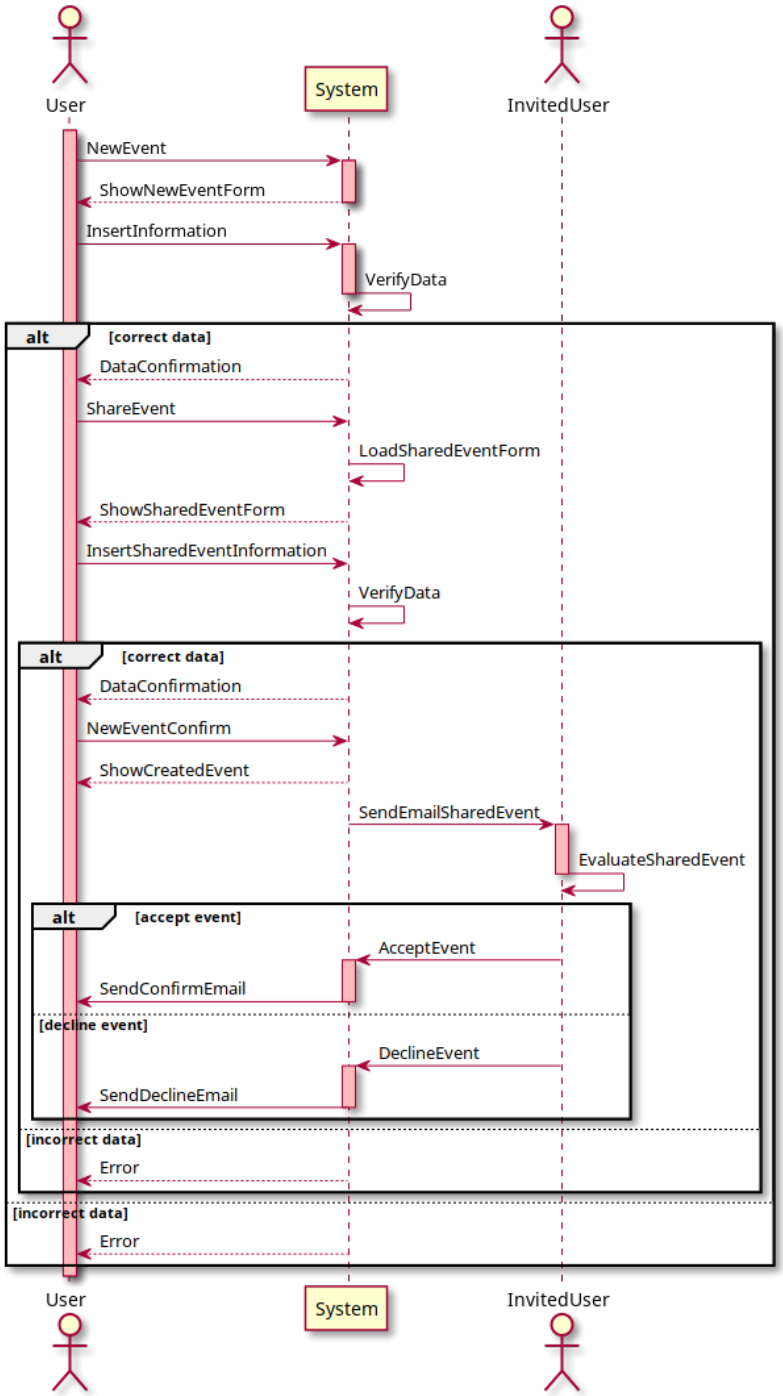

Figure 23: New shared event with invitation

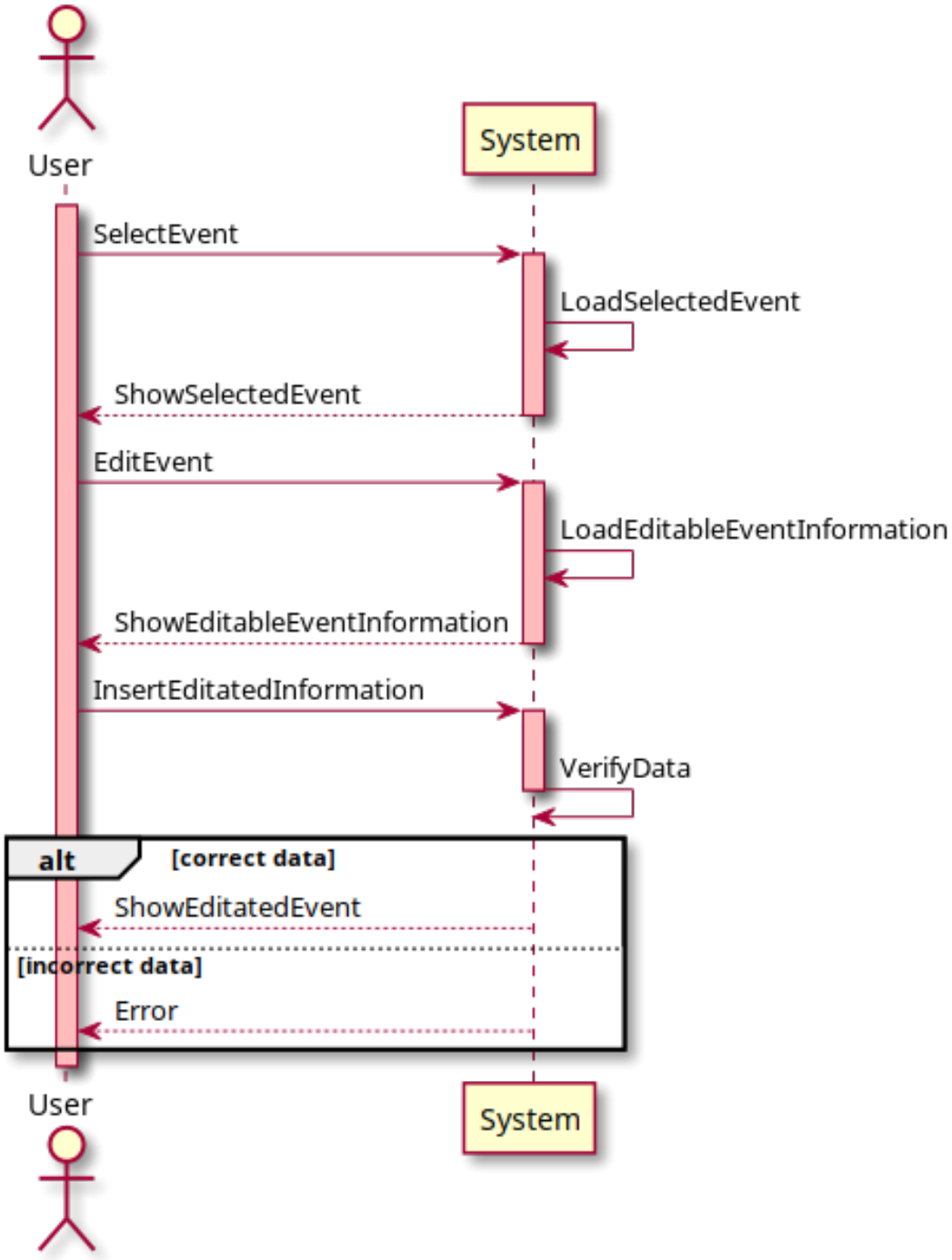

Figure 24: Edit event

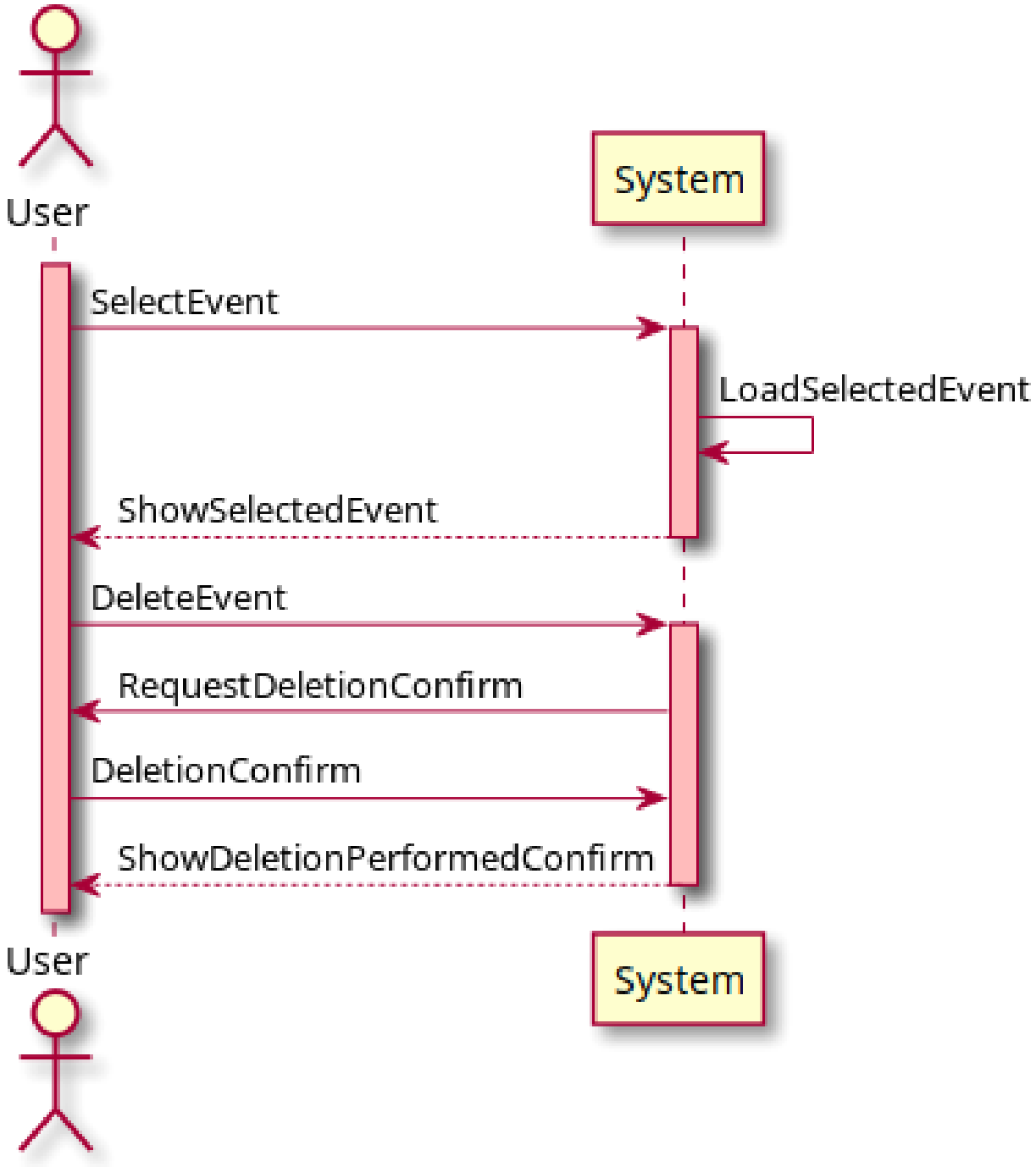

Figure 25: Delete event

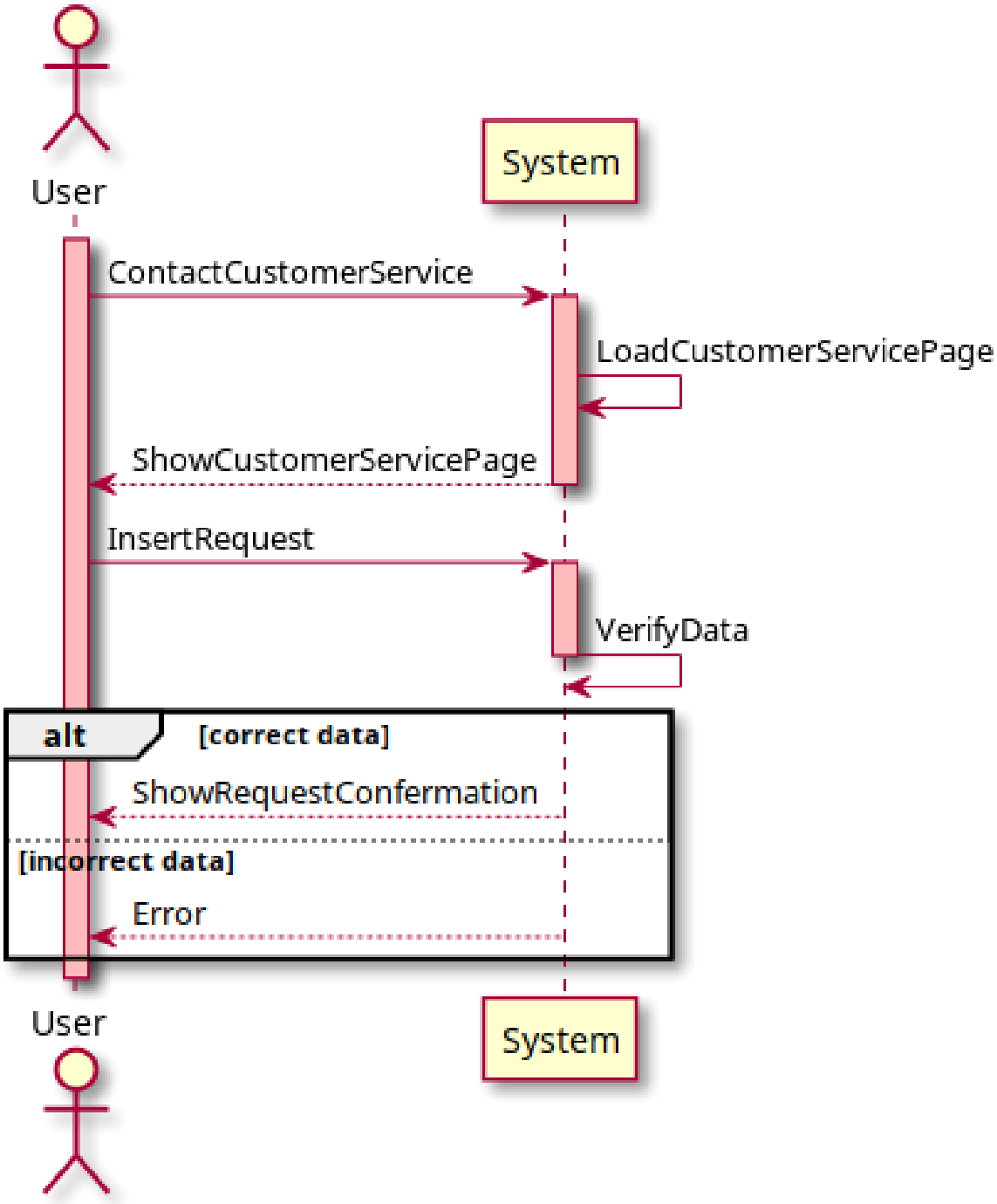

Figure 26: Contact customer service

5.5 Class Diagram

Here it is shown a class diagram in order to give a static sight of the main involved entities and of their relations. The diagram, whose aim is just to better specificy requirements, is shown in this picture.

This diagram is intended as a starting draft to be expanded further in the design phase: moreover, it doesn't represent the final class infrastructure which will be used in our system. Finally, this diagram doesn't include the classes that will be used in the Mobile/Web applications.

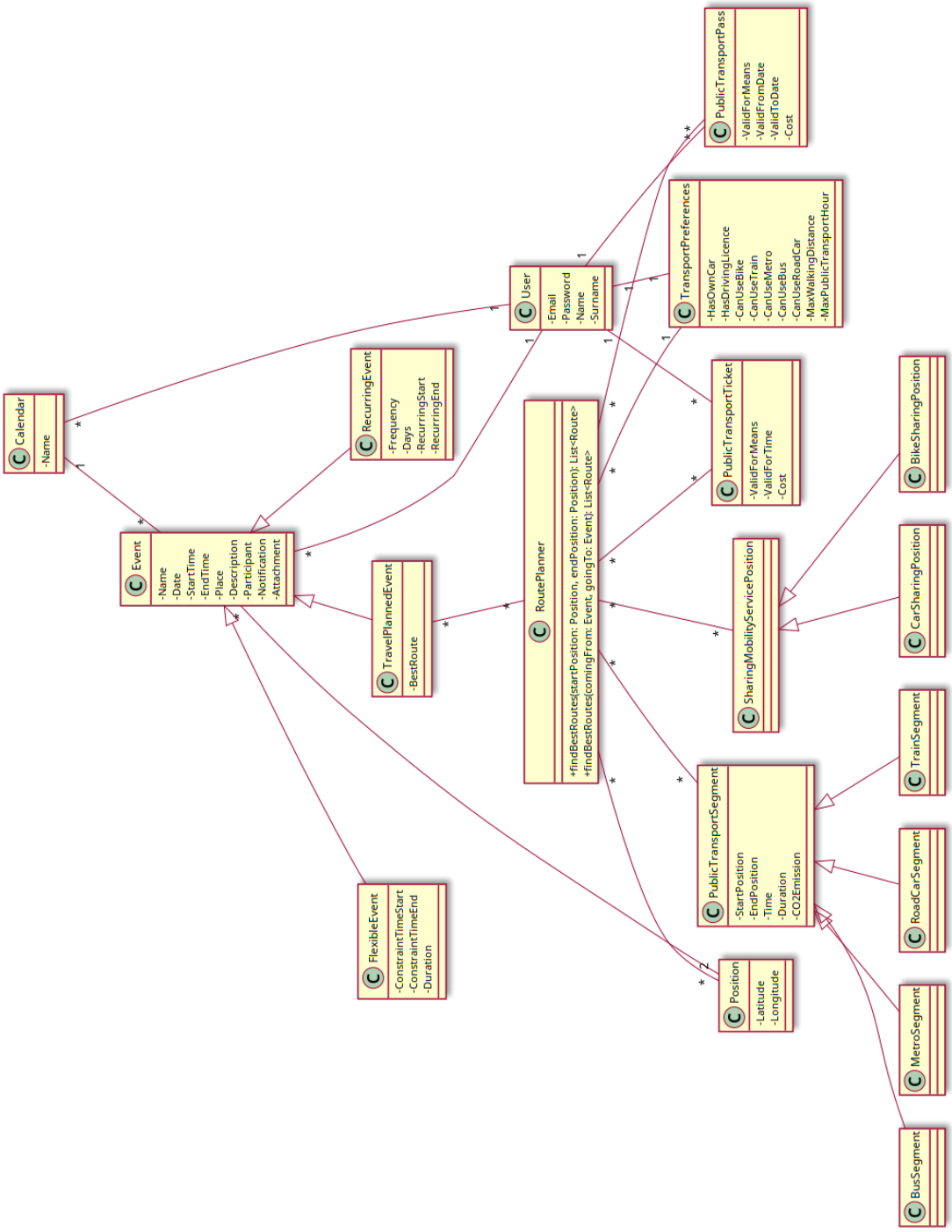

Figure 27: Class Diagram

6 Alloy

In this section we tried to verify the consistency of our class diagram. Here there is the code of the model of the system.

```
module travlendar

open util/integer
open util/ordering[Calendar] as calOrdering

/*
Notes about our model:
It tries to verify the objectives described below.
Only entities and properties relevant to the formal model are considered here:
it doesn't make sense to say that an event has a name for formal purposes.

There's one calendar (which evolves), and one user. This is reasonable because
a user's calendar is writable only by that user.

Objectives:
 - if the user inserts an event which overlaps with another a warning is generated
 - if an event is inserted and there's no way to reach the destination in time a
  warning is generated
 - if the user inserts an event which leaves no time for flexible events a
  warning is generated
*/

// ---------------------- SIGNATURES ----------------------
sig Calendar {
  events: set Event,
  eventsWithOverlaps: set Event,
  unreachableEvents: set Event
}

sig Event {
  travelTime: Int,
  starting: Int,
  ending: Int
} { starting < ending ∧travelTime ≥ 0 ∧starting ≥ 0 ∧ending > 0 }
```

```
// --------------------- FACTS ---------------------
fact {
  emptyCalendar[calOrdering/first]
}

// --------------------- DYNAMIC BEHAVIOR ---------------------
// these predicates represent operations on the dynamic system

// Adds an event to a calendar
pred addEventToCalendarPrecondition[e: Event, c: Calendar] {
  e not in c.events
}

pred addEventToCalendar[e: Event, c, c': Calendar] {
  addEventToCalendarPrecondition[e, c]
    c'.events = c.events + e
  else c' = c

  addEventToCalendarPrecondition[e, c] ∧
  overlapsWithOtherEvents[e, c]
    c'.eventsWithOverlaps = c.eventsWithOverlaps + e +
              { e2 : c.events | twoEventsOverlap[e, e2] }
  else c.eventsWithOverlaps = c'.eventsWithOverlaps

  addEventToCalendarPrecondition[e, c] ∧
  unreachableInTime[e, c]
    c'.unreachableEvents = c.unreachableEvents + e
  else c'.unreachableEvents = c.unreachableEvents
}

// Removes an event from a calendar
pred removeEventFromCalendarPrecondition[e: Event, c: Calendar] {
  e in c.events
}
pred removeEventFromCalendar[e: Event, c, c': Calendar] {
  removeEventFromCalendarPrecondition[e, c]
    c'.events = c.events - e ∧
    c'.unreachableEvents = c.unreachableEvents - e ∧
    c'.eventsWithOverlaps = c.eventsWithOverlaps -
              { e2 : c.events | twoEventsOverlap[e, e2] }
```

```
  else c' = c
}

// --------------------- HELPERS ---------------------

pred emptyCalendar [c: Calendar] {
  no c.events ∧no c.eventsWithOverlaps ∧no c.unreachableEvents
}

// Evaluates whether there's another overlapping event in the calendar
pred overlapsWithOtherEvents[e: Event, c: Calendar] {
  ! (no e2: c.events | e ≠ e2 ∧twoEventsOverlap[e, e2])
}

// Evaluates whether two events overlap
pred twoEventsOverlap[e1, e2: Event] {
  e1.starting < e2.ending ∧e1.ending > e2.starting ∧e1 ≠ e2
}

fun precedingEvents[e: Event, c: Calendar] : set Event {
  { e': c.events | e'.ending ≤ e.starting }
}

// Evaluates whether an event is unreachable in time
pred unreachableInTime[e: Event, c: Calendar] {
  ! (no e' : precedingEvents[e, c] |
    e'.ending + e.travelTime > e.starting
  )
}

// --------------------- ASSERTIONS ---------------------

// Asserts that if two overlapping events are added
// a warning associated with them is generated
assert addingOverlappingEventsGeneratesWarning {
  all c, c', c'': Calendar, disj e1, e2: Event |
    // When adding two overlapping events
    e1 ≠ e2 ∧
    twoEventsOverlap[e1, e2] ∧
    addEventToCalendarPrecondition[e1, c] ∧
```

```
    addEventToCalendar[e1, c, c'] ∧
    addEventToCalendarPrecondition[e2, c'] ∧
    addEventToCalendar[e2, c', c'']

    // a warning is generated for the overlapping events
    (e1 + e2) in c''.eventsWithOverlaps
}

assert addEventToCalendarAddsEvent {
  all c, c': Calendar, e: Event |
    addEventToCalendarPrecondition[e, c] ∧
    addEventToCalendar[e, c, c']  e in c'.events
}

assert addingUnreachableEventGeneratesWarning {
  all c, c': Calendar, e: Event |
    unreachableInTime[e, c] ∧
    addEventToCalendarPrecondition[e, c] ∧
    addEventToCalendar[e, c, c']
    e in c'.unreachableEvents
}

assert removeUndoesAdd {
  all c, c', c'': Calendar, e: Event |
    addEventToCalendarPrecondition[e, c] ∧
    addEventToCalendar[e, c, c'] ∧
    removeEventFromCalendarPrecondition[e, c'] ∧
    removeEventFromCalendar[e, c', c'']  c = c''
}


// --------------------- CHECKS ---------------------

/* Check ok
check twoEventsOverlapSanity {
  all e1, e2: Event |
  twoEventsOverlap[e1, e2] ≤ > twoEventsOverlap[e2, e1]
}
*/
```

```
// works, no counterexamples found
// check addingOverlappingEventsGeneratesWarning for 10 expect 0
// works, no counterexamples found
// check addEventToCalendarAddsEvent expect 0
// check removeUndoesAdd for 3
// works, no counterexamples found
// check addingUnreachableEventGeneratesWarning expect 0
```

7 Appendix

7.1 Tools used

We used the following tools to produce this document:

- **LaTex** as typesetting system to write this document
- **LyX** as editor
- **PlantUML** to draw all the diagrams
- **Alloy Analyzer 4.2** to write and verify alloy models

7.2 Hours spent

Data	Alfonso	Carsenzuola	Cremonese
04/10/2017	4	4	4
05/10/2017	2	2	2
07/10/2017	3	3	3
09/10/2017	1	1	1
10/10/2017	1	1	1
16/10/2017	3	3	3
18/10/2017	4	3	4
19/10/2017	2	2	2
23/10/2017	1.5	3	3
24/10/2017	1	1	1
25/10/2017	1	1	1
26/10/2017	3	2.5	3
27/10/2017	4.5	4.5	4.5
28/10/2017	7	7	8
29/10/2017	8	8	8
Totale	46	46	48.5

7.3 Acknowledgments

The template for this document was taken (with explicit prior permission) from the SE2 project of Colaci, De Pasquale, Rinaldi (https://github.com/giancarlocolaci/SE2_Project_Colaci_DePasquale_Rinaldi)

7.4 References

Google Calendar	https://calendar.google.com/calendar/
Google Maps	https://www.google.it/maps/
City Mapper	https://citymapper.com/
Uber	https://www.uber.com/
Enjoy	https://enjoy.eni.com/
Mobike	https://mobike.com/
European Cookie Law	http://eur-lex.europa.eu/LexUriServ/...
European General Data Protection Law	http://eur-lex.europa.eu/legal-content/...
iCalendar RFC	https://tools.ietf.org/html/rfc5545
CalDAV RFCs	https://tools.ietf.org/html/rfc4791
CalDAV RFCs	https://tools.ietf.org/html/rfc6638