Safe Streets

Nicola Rosetti Simone Sartoni Vittorio Torri

Contents

List of Figures

1 Introduction

1.1 Scope

SafeStreets is an application meant to provide a mechanism to more efficiently detect parking violations and to try to make streets safer. This is meant both for normal people and for municipality agents. The application domain concerns different types of world phenomena, shared phenomena and machine phenomena.
We can identify the following world phenomena:

- [W1] A car is illegally parked
- [W2] An accident occurs
- [W3] An intervention is performed to make the roads safer

the following shared phenomena controlled by the world and observed by the machine:

- [S1] A violation report is sent by a user
- [S2] An accident report is provided by the municipality system
- [S3] New information about issued tickets are provided by the municipality system

and the following shared phenomena controlled by the machine and observed by the world:

- [S4] A traffic ticket is emitted by an agent, exploiting the municipality system, after a report analysis
- [S5] An agent is sent to verify the correctness of a violation report
- [S6] A suggestion for a safety improvement on streets is made available

1.2 Purpose

The system is characterized by the following goals, distinguished by the function provided:

- Violations and tickets function:
  - [G1] People must be allowed to report parking violations (missing parking disk, not paid parking meter, illegally parked vehicles).
  - [G2] Municipality agents must be notified about reports of potential violations in their area of interest, which can be possibly used to generate traffic tickets.

- Statistic functions:
  - [G3] People must be allowed to retrieve information about streets or areas with the highest frequency of violations.
  - [G4] People must be able to retrieve statistics and trends about the accidents correlated to the parking violations, the effectiveness of SafeStreets initiatives and the issuing of traffic tickets.
  - [G5] Municipality supervisors must be able to retrieve data about the vehicles with the highest number of violations.
  - [G6] Municipalities must be suggested for possible interventions about the mostly unsafe areas.

In the following table is shown which world and shared phenoma are involved for each goal.

	W1	W2	W3	S1	S2	S3	S4	S5	S6
G1	X			X					
G2	X			X			X	X	
G3	X			X		X			
G4	X	X		X	X	X	X		
G5	X					X	X		
G6			X						X

1.3 Definitions, acronyms and abbreviations

- *GPS*: Global Positioning System
- *HTTPS*: HyperText Transmission Protocol over SSL
- *IDS*: Intrusion Detection System

1.4 Revision history

- 10-11-2019 Version 1.0

1.5 Reference documents

- Assignment Document "SafeStreets Mandatory Project Assignment.pdf"

1.6 Document structure

The document consists of 4 chapters:

Chapter 1: In this chapter is presented basically the domain of the application, taking into consideration all the world and shared phenomena that the application observes and/or controls. The scope of the application is specified, listing all the goals. It also contains additional information to make the reading of the document more understandable.

Chapter 2: In this chapter the application itself is taken into consideration, concerning its functions and the potential stakeholders that actually will use it. Considering users and the world in which they act, all the assumptions made on the domain are listed. A series of UML diagrams (Class and State diagrams) are then presented to help understanding how the application should work in terms of entities involved and functionalities.

Chapter 3: In this chapter the application behavior is described: at first some scenarios are identified and then some UML diagrams (sequence and use cases diagrams) are provided, with a detailed analysis of the use cases; at last by specifying requirements in terms of :

- functional requirements;
- non-functional requirements;
- interface requirements.

Chapter 4: In the last chapter a brief but concise and effective formal analysis of the problem is presented using alloys. Some requirements and domain assumptions are formally specified and their consistency is verified, with the help of the Alloy tool.

2 Overall Description

2.1 Product perspective

In figure 2.1 is reported an *UML Class Diagram* which represents the domain of the application with main concepts and data involved, including their relationships.
Not all classes will necessary become effective classes in the implementation (e.g. the *Visitor* class). Some data appear to be duplicated in the *TrafficTicket* class but it needs to be considered the fact that the application will receive also data about traffic tickets which have not been issued as a consequence of a SafeStreets report, but instead have been independently issued by the local police in their traditional control activities. They are necessary to correctly build statistics and they need to include data which are already present in the corresponding report, when it exists.
For what concerns the *Accidents* here it has been supposed a representation for all types of accidents, coming from the municipality data, with a specified type for accidents related to bad parked cars.
For what concerns statistics here there is a double hierarchy: a division between area and street statistics and a division between public and the reserved (i.e. only for municipality agents/supervisors) ones. As it will be explained later, the system allows users to see the areas or streets with the most frequent violations or with the better improvements and similar other classifications, but here, in the Class Diagram, only the basic data which allow to build this more complex statistics are represented, both for simplicity and because probably other classifications will be calculated on demand, avoiding to maintain a copy constantly updated.
In figure 2.2 is reported an *UML State Diagram* to clearly represent the state evolution of a *ViolationReport*, which is the main object which change its state in the time, while the other ones do not present significant evolutions from this point of view.

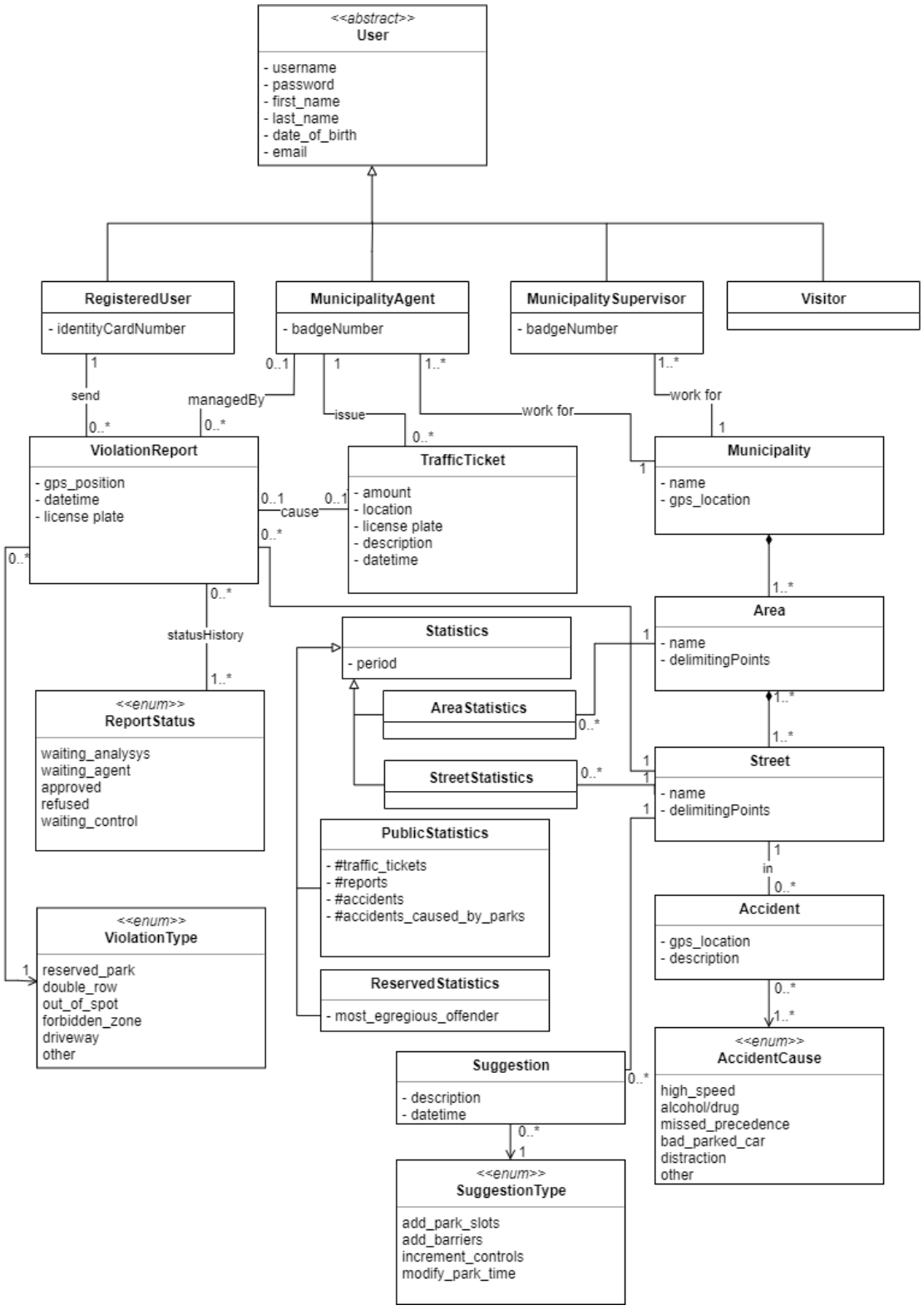

Figure 2.1: UML Class Diagram

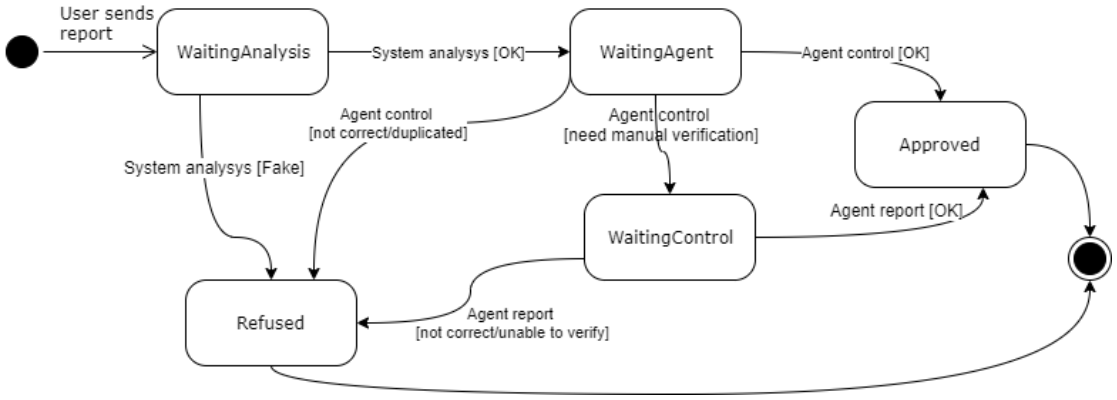

Figure 2.2: UML State Diagram for ViolationReport

2.2 Product functions

In this part are underlined the main functions offered by SafeStreets, divided by the users that can actually have access to them.

2.2.1 Visitor

Registering to the application

The system offers the possibility to Visitors to register and actually become users of the application. When registering the visitor needs to provide his name, surname, birthdate, fiscal code and identity card number. They are mandatory so if some of them is not provided then the registration process fails.

2.2.2 AuthenticatedUser

Reporting a violation

This is the most important function offered by SafeStreets. This function allows Users, when logged in, to report a potential violation that has occured in the streets. They can achieve that by inserting some mandatory data. When reporting a violation, a user must insert the type of violation that has been detected (missing park disk, car illegally parked in the bike lane, car illegally parked in some reserved parking spot, not paid parking disk). Subsequently, the user must provide at least one picture of the violation. Through the pictures, the user should provide clear evidence of the violation and the vehicle involved (in particular the license plate) before sending the report for a verification to the server. The user can help recognition providing the license plate of the vehicle as plain text. Before sending the report, the application attaches to it the current position of the User (using GPS position) and date and time of the report.

2.2.3 Municipality agent

Authority agent registering

Differently from the function previously explained and meant for normal Users, this function is only meant for Municipality agents. Each agent can register to SafeStreets to exploit his services as agent. He needs to provide his name, surname, birthdate and rank. After providing a password, the registration is sent for a verification to the municipality service and completed if the data inserted is correct. An ID code is then given to the agent to be uniquely identified.

Violation checking

After logging in as a municipality agent, the system notifies the agent about all the current not resolved reported violations. The agent can visualize individually each report, starting from the oldest one in terms of date and time, checking if the report corresponds to an actual violation. He can either directly emit a traffic ticket or send an agent on the street to check and eventually emit the ticket. If he decides to send an agent to verify the report he can put it on hold and take it back later from the "On hold" queue. It can also refuse the report if it is duplicated or if he is sure that is not a real violation. In this case he can also access the report author's data. He can then proceed and check the next report on the queue, if present.
Additionally, when he is logged, the agent is notified of new incoming reports.
Reports which are automatically rejected by the system controls are shown to the agents as rejected and they can check them, obtain the author's data and possibly change the automatically taken decision.

2.2.4 Municipality supervisor

Suggesting possible solutions

This function consists of a task periodically executed by the system (once a month) which tries to analyze reported violations, issued tickets, accidents information and data about the streets network coming from the municipality to suggest possible interventions to increase the safety (e.g. add barriers, create new park spots, increase the allowed parking time, send more agents in a certain area in certain hours...). When the system finds new suggestions for certain streets it sends them to the municipality supervisors.

Getting sensible information

This function allows logged in municipality supervisors to retrieve information about the vehicles with the highest number of violations in a selected area. The system must return information belonging to the supervisor's area of competence, communicated during the registration process.

2.2.5 Everyone

Consulting statistics

This function allows visitors, users, municipality agents or supervisors to retrieve information about the streets and areas with the highest frequency of violations; it is possible to retrieve statistics and trends about the accidents correlated to the parking violations, the effectiveness of SafeStreets initiatives and the issuing of traffic tickets. The information provided must be mined by the system from the reported violations, crossed with information from municipality. The

system must not allow user to see confidential data about other people; it must allow the user to choose a topic: areas with most accidents, areas with the highest number of traffic tickets issued or areas where there have been the best improvements and in the end the system must show to the user the information about the topic selected.

2.3 User characteristics

The users of the service are the following:

- *Visitor*: a non-logged user which can only consult statistics about areas with the highest frequency of violations, highest rate of accidents related to the violations, traffic tickets issued and safety improvements.
- *Authenticated User*: an identified user which, in addition to the *visitor*, can report a parking violation, sending the details to the municipality authorities.
- *Municipality agent*: an agent of the local police which is notified about the violation reports for his municipality and can manage them as described in *Violation checking* function.
- *Municipality supervisor*: he has a full access to the application data, including all statistics and the suggestions to improve safety in the most dangerous areas.

2.4 Assumptions, dependencies and constraints

2.4.1 Dependencies and constraints

The presence of some services provided by the municipalities is necessary to make all SafeStreets functions operative. In particular the following services are requested:

- *Identity Card Check*: allows to retrieve data of a person given its identity card number. It's request for a strong user authentication.
- *Statistics Information*: return information about the accidents occurred in the municipality streets and about the parking violations traffic tickets issued outside the SafeStreets system, with position and causes.
- *Road Network Information*: allows to retrieve information about the streets characteristics, such as park slots, park time, sidewalks, barriers.
- *Traffic Ticket Issue*: allows to send traffic tickets to a certain person by a certain agent
- *Agents Identity Check*: allows to check the identity of a registering agent given his personal data, including his badge number.

2.4.2 Domain Assumptions

Here are shown the domain assumptions:

- [D1] GPS is always precise, giving the position with an error of maximum 5 meters.
- [D2] Smartphone cameras have enough resolution to enable recognizing algorithms

- [D3] Every municipality which exploits the SafeStreets services has defined some areas in which the city is divided, eventually overlapped, and every street of the city is at least in one area (in the worst case the entire city is a single area)
- [D4] The identity card is unique for each person.
- [D5] Each person has only one identity card.
- [D6] The *Traffic Tickets Issue* service of the municipalities always issue traffic tickets if it receives the correct information and if it is available (i.e. it is not in a downtime moment).
- [D7] Every registered municipality agent has a unique ID (badge number) inside its municipality.
- [D8] Every municipality that uses SafeStreets has at least one Agent and one Supervisor that are in charge for managing SafeStreets services.
- [D9] Every agent and supervisor works for exactly one municipality.

3 Specific Requirements

3.1 Scenarios identification

Here are presented possible scenarios for the different users of the system.

Scenario 1. Report of a violation

Mark is a man in his thirties. He's an employee at Esselunga supermarkets. One day, walking down the main street of his city he finds a car parked illegally in the middle of the bike lane. Mark is registered in the application SafeStreets. He takes out his smartphone and opens the application. After writing his credentials and logging in, he clicks the button "Report a violation" in the home page. He inserts the type of violation, so in this case he just writes "car illegally parked in bike lane", and takes three different pictures of the car: one of the front of the car, clearly showing the license plate; the second showing the entire car and, in the background, the signal of bike lane; the third one highlighting important elements of the streets where the potential violation occurred, to help the matching of the photo with the GPS position of the user. He then confirms clicking on the "Confirm" button. The potential violation is now sent to the server for a verification. Finally, he closes the application and continues his walk.

Scenario 2. The agent receives a notification and issues a traffic ticket

Lukas is a municipality agent that is working as usual at his desk and has the SafeStreets Web app opened in the background on his PC. While checking his papers he receives a notification from the SafeStreets. He opens the window of the app and finds that a new report has been made about a violation. Lukas clicks on the row linked to the new violation. He observes that the violation has been reported by a certain Gianluca Verdi. The pictures of the report clearly show the vehicle that has made the violation (not paid parking meter), his license plate and the place where the violation occurred. There is now enough evidence that allows Lukas to issue a traffic ticket to the owner of the vehicle.

Scenario 3. A policeman receives a notification and send an agent to control because the picture is not clear

Laura is a municipality agent that has just started her new day at work. She works in Milan's police station. She starts her PC at her desk, opens Internet Browser and search for SafeStreets Web App. After logging in, the reported violation queue is filled with 10 notifications of new potential violations reported. She opens the first one and she discovers that the system wasn't able to correctly identify the vehicle involved and the user hasn't inserted an existing license plate. She checks the pictures but she can't find out what is written on the license plate. She tells

his supervisor the problem, in order to send an agent to directly check in the place of the supposed violation (based on the GPS position sent by the user) and, possibly, to issue a traffic ticket.

Scenario 4. A user checks for unsafe streets in his zone

Bob is a curious user that has the SafeStreets application installed in his iPhone 8 but is not registered in the SafeStreets database. One day, he witnesses an accident in the streets while driving his car, where a car hits a biker during a parking manoeuvre on a runs over a biker correctly biking in his bike lane. As he goes home, he's curious about the most dangerous and unsafe areas and streets in his city and he wants to see if the street where he saw the accident is one of them. He opens the SafeStreets application and clicks the button "Check statistics in a city". After this, a new page opens in which Bob clicks on the "Check for most dangerous areas/streets". The last click he does is on the "Search for a city using your GPS position" button. Now the screen shows a map with highlighted the most unsafe areas (with most accidents) in his city. Bob selects then the area in which he is interested in and he sees which are the most dangerous streets in the area. Bob discovers that the street he was searching for is the most dangerous one. Finally, Bob closes the application.

Scenario 5. Needs to travel and searches for safe places for a car

Marie is a woman who often travels for work. She typically travels by car. She lives in Milan and this week she needs to go to Turin. He wants to know where she can safely park the car and take an hotel reservation. Before searching for an hotel, she takes out her Huawei and opens the SafeStreets application. She clicks the button "Check statistics in a city". After this, a new page opens in which Bob clicks on the "Check for most dangerous areas/streets". The last click he does is on the "Select a city you want" button. She chooses "Turin" among all the possible ones. Now she can use found information to look for safe zones near her workplace.

Scenario 6. New intervention suggestion for an unsafe street

Marco is a local police supervisor in the municipality of Varese. He logins in the SafeStreets application and he sees a notification about a suggestion for an intervention to improve safety. He opens it and he reads that in Via Roma there have been a lot of reports and consequent tickets in the last months due to cars parked on the sidewalk. The system is suggesting to add barriers to prevent cars from going on the sidewalk. This seems a good suggestion for him, he will discuss it with the competent municipality sector.

Scenario 7. The user registers to the application

Luke and Walter are two close friends. Luke doesn't know about SafeStreets application while Walter is a regular registered user of it. One day, Walter and Luke are walking down a road. Walter notices a potential violation and stops. He takes out his phone and opens the SafeStreets application. Luke asks Walter what he's doing and Walter explains what SafeStreets is. Walter is pretty convincing and makes Luke install the application. Luke learns that to report violations he needs to register. So, he starts the process of registration providing his name, surname, birthdate, fiscal code and number of the identity card. After this, he creates his own unique username and the password. On screen, Luke sees that the registration has been successful.

Scenario 8. A supervisor wants to check see what areas are registering improve-

ments since the municipality has started to use SafeStreets

Giulio is a local police supervisor in the municipality of Milan. It's six months since they have started to use the SafeStreets application to receive report from citizens about parking violations and he want to check what benefits it has brought. He opens the web application, he logins inserting his credentials and he clicks on Improvements under the Statistics section. Here he can see what streets have seen the best reduces in the number of issued tickets. He sees that a lot of central zones have seen reductions from 20 to 50 percent, while in the peripheral areas the numbers are much less substantial. He is happy because of this result, but he wants to improve it and he thinks to start a marketing campaign to encourage more people to use the application, especially in the peripherals.

Scenario 9. Marco wants to make a joke to a friend and send a fake report

Marco and Luigi are two very close friends. One day Marco decides it's time to make a very not funny joke to his friend and he decides to use the SafeStreets application to make him receive a traffic ticket. He finds an illegally parked car, he photographs it and then he tries to modify the license plate in the photo with Luigi's car license plate. He sends the report with the modified photo thinking about the face of Luigi when he will receive the ticket. Some weeks later he receives a letter from the judicial authority stating that they have started a criminal proceeding against him for the crime of slander (see *[SLANDER]*). Probably he won't do it another time.

Scenario 10. A municipality agent registers to the application

Ben is a new local police agent in the municipality of Bologna hired only two days ago and his supervisor says him that he has to register to the SafeStreets application. So Ben open the already installed application from his Computer and clicks on the "Register" button on the homepage. He inserts his name, surname, birthdate, rank and badge number, then he chooses his password, equal to "LKJVC89", and confirm the inputs. After the registration is processed, Ben receives his unique ID and is returned to the home page. He can now properly work.

3.2 UML diagrams analysis

3.2.1 Use case diagram

In figure 3.1 is report the UML Use Case Diagram.

3.2.2 Use case identification

In this section the various use cases present in the use case diagram and derived from the scenarios previously described are illustrated in detail.

1. User logs in

Actors: User

Entry condition:

- The User needs to have the application opened

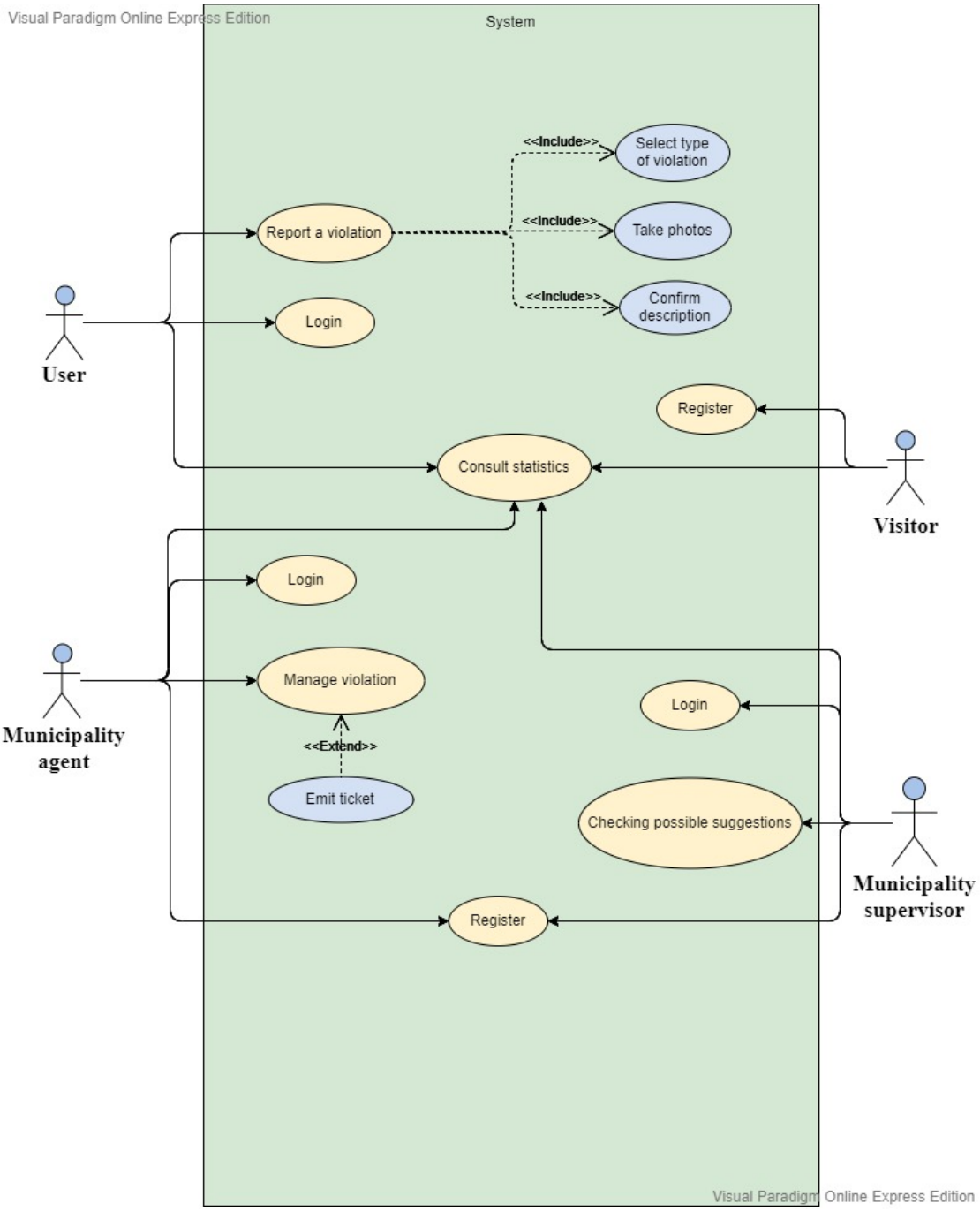

Figure 3.1: Use Case Diagram

Flow of events:

- The User clicks on the login button
- The user input his username (ID) and his password
- The user clicks on the “confirm” button
- The system redirects the taxi driver to his personal homepage

Exit conditions:

- The user is successfully redirected to his personal homepage

Exceptions:

- The code and password provided by the user are not correct. In this case, the system does not redirect the user to his personal homepage but notifies him that an error has been made and allows to input his code and password again

2. User reports a violation

Actors: Authenticated User

Entry condition:

- The user must be logged in and on his personal homepage.

Flow of events:

- The user input clicks on the “Report violation” button.
- The user sets the “type of violation” button to a desired value, chosen from a choice list.
- The user clicks on the “take photos” button and complete the operation.
- The user eventually fills the form “license plate”.
- The user clicks on the “confirm” button and is redirected to his personal home page.

Exit conditions:

- The system checks the report and manages it, checking if images are corrupted and eventually forwarding the report to the municipality agent.
- Before confirming, the User clicks on the "Back” button and cancels the current report. Pictures are cancelled too.

Exceptions:
There are no exceptions for this use case.

3. Agent or supervisor registers

Actors: Municipality agent or Municipality supervisor

Entry condition:

- the agent or supervisor needs to have the Web App opened.

Flow of events:

- The agent/supervisor opens the application and arrives at the homepage.
- The agent/supervisor clicks on the "Register" button.
- The agent/supervisor input his name, surname, birthdate and rank.
- The agent/supervisor input his choosen password.
- The agent/supervisor clicks on the "Complete Registration" button.
- The system processes the registration and send information to the Municipality Service.
- The system notify the agent/supervisor about his new Unique Identifier (ID) and suggest to remember it.
- The agent/supervisor clicks on the "Understood" button.
- The system redirects the agent/supervisor to login page.

Exit conditions:

- The agent/supervisor is successfully redirected to login page.

Exceptions:

- The name, surname, birthdate, rank and badge number furnished by the agent are not found by the municipality service. In this case, the system does not redirect the agent/-supervisor to the login page, but notifies him that an error has been made and allows to insert the data again.

4. Agent or supervisor logs in

Actors: Municipality agent or Municipality supervisor

Entry condition:

- the agent/supervisor needs to have the Web App opened.

Flow of events:

- The agent/supervisor opens the application and arrives at the homepage.
- The agent/supervisor clicks on the “Login” button.
- The agent/supervisor input his username and password.
- The agent/supervisor clicks on the log in button.
- The system redirects the agent/supervisor to his personal page.

Exit conditions:

- The agent/supervisor is successfully redirected to his personal page.

Exceptions:

- The username and password furnished by the agent/supervisor are not correct. In this case, the system does not redirect the agent/supervisor to his personal page, but notifies him that an error has been made and allows to input his username and password again.

5. Checking statistics

Actors: everyone (logged or not)

Entry condition:

- The user needs to have the application opened

Flow of events:

- The user clicks on the Checking statistics button in the main page
- The user chooses to get statistics either from his city or from another one
- The user selects the type of statistics: data about a specific street or area, or a classification of streets or area on the base of the number of reports,tickets,accidents or increase/reduction in these values.
- After checking, user presses the Back button to return to the home page
- The system asks the user if he is sure to exit with a "Yes" or "No" choice
- The user presses the button he wants, confirming his choice

Exit conditions:

- User is successfully brought back to the home page

Exceptions:
There are no exception in this case

6. Supervisor checks potential intervention

Actors: Supervisor

Entry condition:

- The supervisor must be logged in. Possibly he have received a notification about a new available suggestion.

Flow of events:

- The supervisor clicks on the "Recent Suggestions" button or clicks on the new suggestion notification.
- The system redirects the supervisor to a page where he can see the list of suggestions from the most recent to the oldest
- The supervisor chooses a suggestion.

- The system shows the supervisor the whole text of the suggestion.
- The supervisor clicks the “Done” button and is redirected again to his homepage.

Exit conditions:

- The supervisor is successfully brought back to his personal home page.

Exceptions:

- There are no recent suggestions, so the system notifies the supervisor about this exception. The system redirect the supervisor to his personal page.

7. Agent checks a notified violation

Actor: Municipality Agent

Entry condition:

- The agent needs to be logged in the municipality Web App

Flow of events:

- The agent receives the notification through the Web App of a new incoming report violation
- Clicking on it, or entering in the Reports section, the system shows him the first report in the queue.
- If the pictures are not clear enough, the agent contacts the supervisor to send an agent in the street to check and in the meanwhile he can put the report on hold by clicking on the button "On Hold".
- If he is sure that what is reported is not a violation or if he recognize it as a duplicate he can discard the report with the button "Discard". Before doing this he can possibly retrieve data about the author of the report, clicking on the button "Author".
- Otherwise the agent clicks the "Send ticket" button to file the report and send a ticket to the recognized owner of the vehicle. The system allows him to specify the amount of the ticket and a description.

Exit conditions:

- The violation is filed and the agent can continue with the next report in the queue, if it is not empty.

Exceptions:

- The municipality service used to issue traffic tickets is unreachable: in this case the system informs the agent and does not fill the report. The agent can retry until the system answers correctly.

8. User registers to the Application

Actor: Visitor

Entry condition:

- User needs to have the application opened in the main page

Flow of events:

- User clicks on the “sign in” button.
- User provides the required information: fiscal code, identity card number, name username and birthdate;
- User provides a Username and a password;
- User clicks on “Confirm” button

Exit conditions:

- The registration is completed and a message of confirmation is sent to the user, redirecting him to the main page of the application

Exceptions:

- The username provided is already in use the system shows an error message to the user; he has to insert a new Username (and password)
- The sensitive information inserted are not correct (fiscal code or identity card number). The system shows an error message and the user has to insert again all his information.
- User doesn’t complete the registration process by pressing the "back" button in the page. He’s redirected to the main page.

In the following table is shown which goals are involved in every use case.

	G1	G2	G3	G4	G5	G6
UC1	X					
UC2	X					
UC3		X				X
UC4		X				X
UC5			X	X	X	
UC6						X
UC7		X				
UC8	X					

3.2.3 Sequence diagrams

in figures 3.2, 3.3, 3.4, 3.5 are reported some UML Sequence Diagrams for the most interesting use cases.

3.3 External interface requirements

3.3.1 User Interfaces

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget,

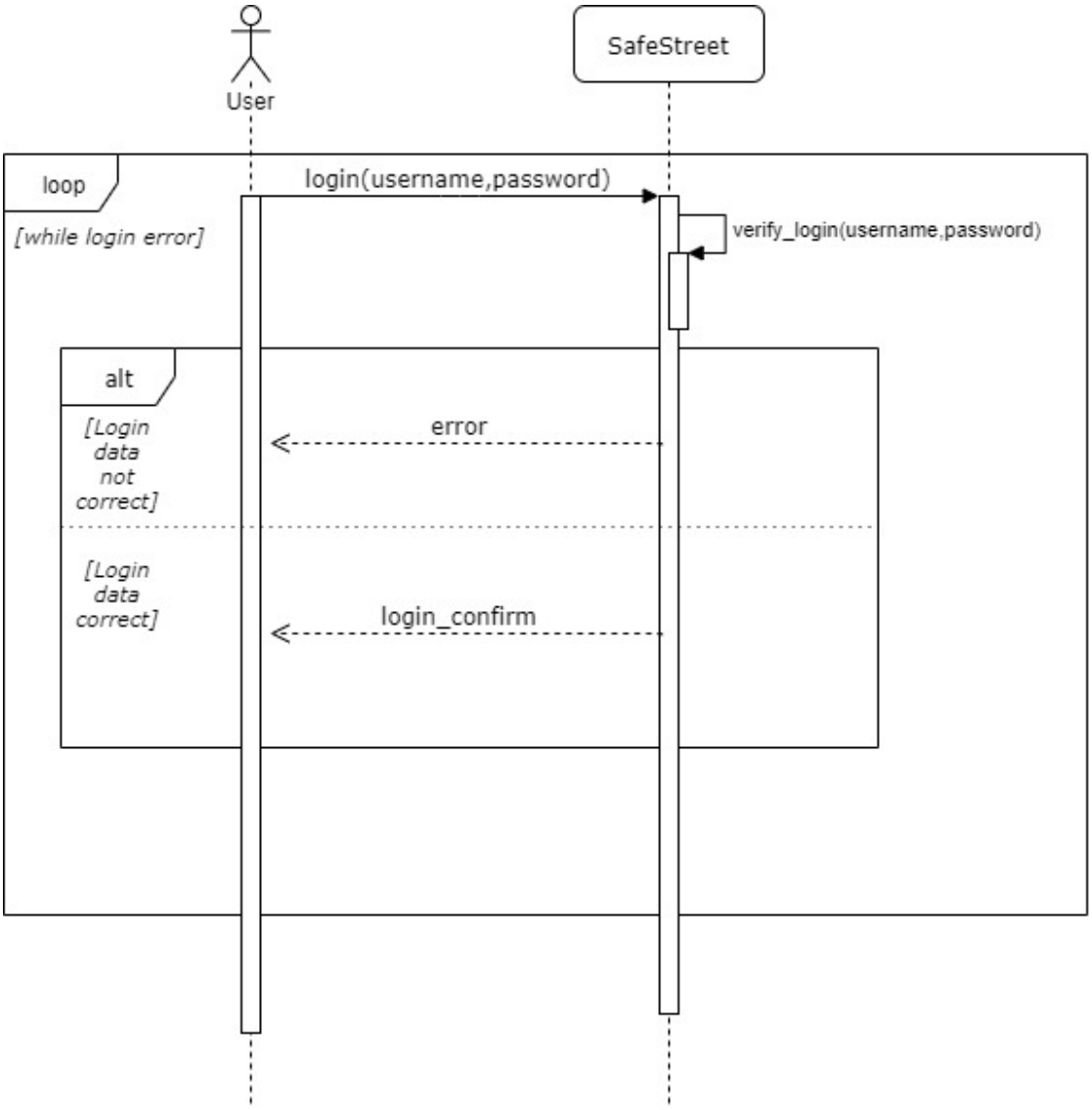

Figure 3.2: Sequence diagram for use case n.1 ((User logs in)

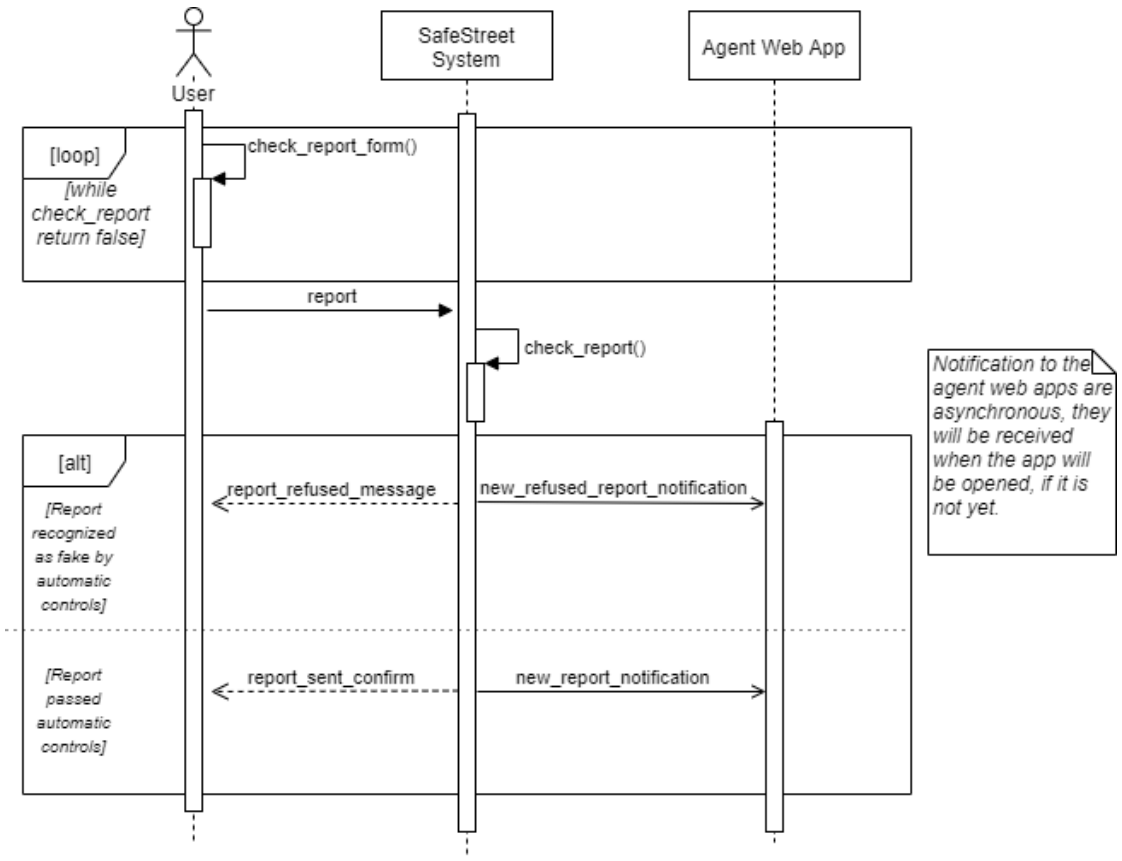

Figure 3.3: Sequence diagram for use case n.2 (User reports a violation)

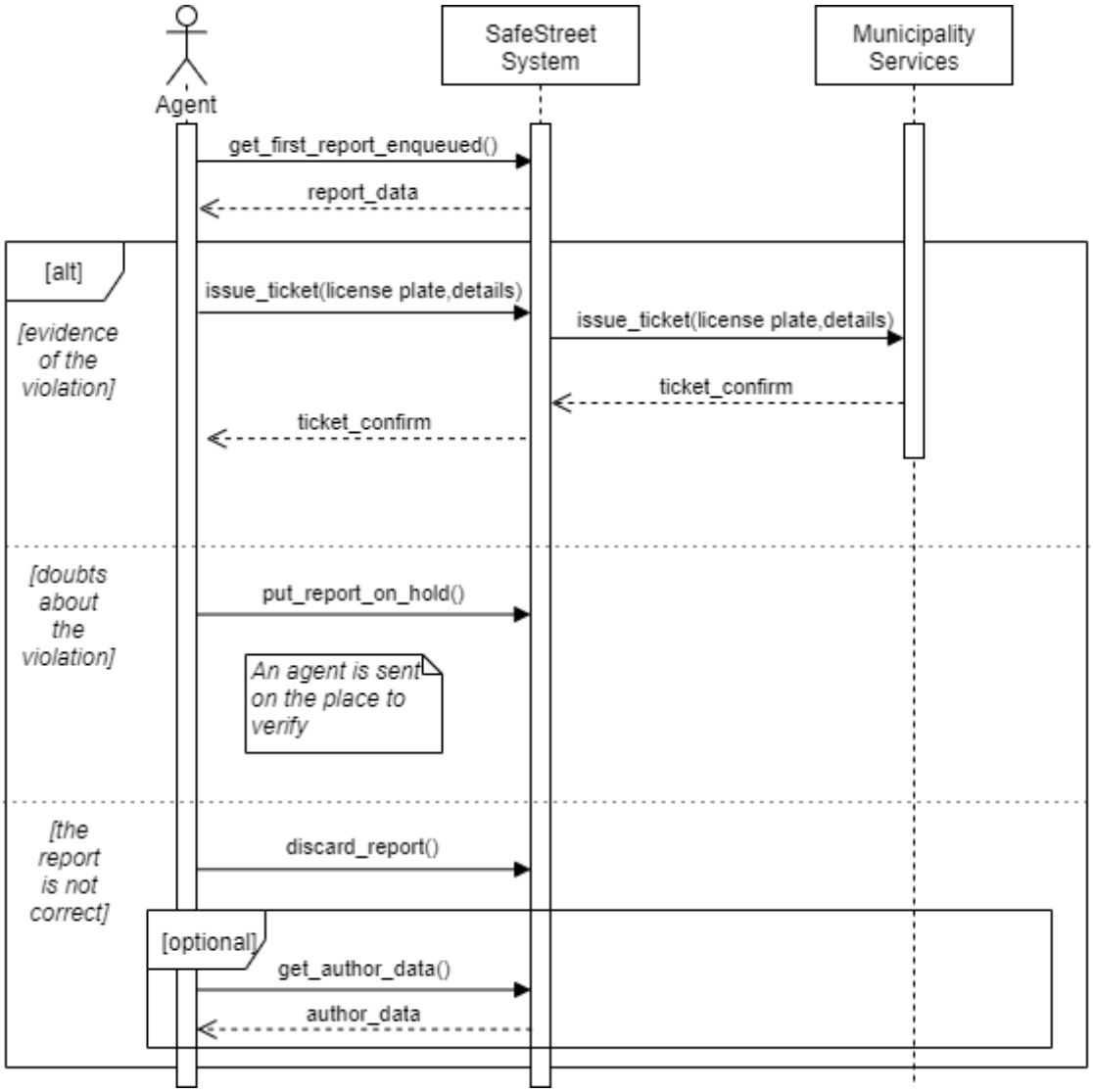

Figure 3.4: Sequence diagram for use case n.7 (Agents checks a notified violation)

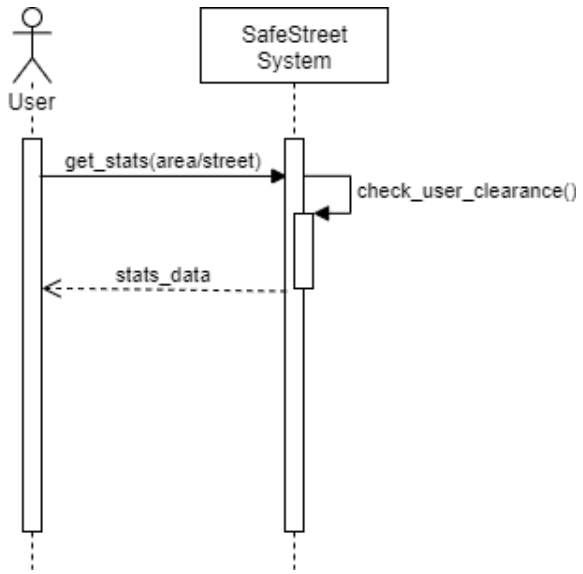

Figure 3.5: Sequence diagram for use case n.5 (Checking statistics)

consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.3.2 Hardware Interfaces

No hardware interfaces are provided, being SafeStreets just a software system.

3.3.3 Software Interfaces

The system does not provide any software interface, because there are no other application which actually need to retrieve data from it.
The system has to call the municipalities services to retrieve some information (see 2.4.1).

3.3.4 Communication Interfaces

The communication between users and SafeStreets servers exploit internal APIs through the *HTTPS* protocol. The same is assumed for the communication with the municipalities systems. For the *users* the communication is unidirectional, in the sense that they cannot receive requests/notification by the server, all communications start from them.
The *municipality agents* can be notified from the server when there are new report to be analyzed.

The *municipality supervisors* can be notified about suggestions for interventions on the most unsafe areas.

3.4 Functional requirements

For each goal we describe here the necessary functional requirements, which guarantee its satisfaction together with the domain assumptions:

- [G1] People must be allow to report parking violations (missing parking disk, not paid parking meter, illegally parked vehicles).
  - [R1] The system must allow people to register to it providing personal data (name, surname, birthdate, identity card number, fiscal code) and selecting a username and a password.
  - [R2] The system must verify the correctness of the provided personal data of a registered user checking them from the identity card number, blocking the registration if they are not correct.
  - [R3] The system must allow registered users to login through their username and password.
  - [R4] The system must allow logged user to fill a report violation form.
  - [R5] The system must let the user select the type of violation detected.
  - [R6] The system must allow the user to insert the license plate in a violation report.
  - [R7] While reporting the violation, the system must allow users to take one or more pictures of the potential violation.
  - [R8] The system must not allow users to choose pictures not taken in the moment of the report.
  - [R9] The system must collect the current position of the user, using GPS.
  - [R10] The system must allow user to confirm or delete the current report.
  - [R11] After confirmation, the system must add the current date and time to the report.
  - [R12] The system must store confirmed report.
- [G2] Municipality agents must be notified about reports of potential violations in their area of interest, which can be possibly used to generate traffic tickets.
  - [R13] The system must check reports to try to find if the pictures of the violations have been modified.
  - [R14] The system must try to find, according to the GPS position of the user and the pictures sent, if the position is fake or not.
  - [R15] The system must discard the report if it has been recognized as fake according to the previous requirements (R13-R14).
  - [R16] The system must try to automatically recognize the license plate in the photo, possibly with the help of the value inserted by the user.
  - [R17] The system must store into stable memory the reported violation if correct (i.e. not recognized as fake).

  - [R18] The involved municipality must be calculated considering in which city the reported violation has been found, based on the GPS position of the user that has sent the report.
  - [R19] The system must send the reported violations to the involved municipality.
  - [R20] The system must allow an agent to see the reports for its municipality, checking them in order of arrival
  - [R21] The system must allow an agent to issue a traffic ticket to a certain person (i.e. license plate) through the correspondent municipality service.
  - [R22] The system must allow an agent to put on hold a violation report if it needs to be checked in person.
  - [R23] The system must allow an agent to discard a violation if it has been verified as fake or it cannot be verified (the vehicle is not there anymore) or it is a duplicated report.
  - [R24] The system must allow an agent to retrieve the data of the author of a violation report.
  - [R25] The system must allow an agent to create an account, asking the municipality services to verify its identity.
  - [R26] The system must allow an agent to login, inserting its username and password.

- [G3] People must be allowed to retrieve information about streets or areas with the highest frequency of violations.
  - [R27] The system must mine this information from the reported violations.
  - [R28] The system must allow the users (even if not authenticated) to select the see information for a city. The user can choose either the city where he is, using the GPS position, or an arbitrary selected location.
  - [R29] The system must allow the user to select information about streets or areas in the city selected and to specify if he wants information for a specific street or area or a classification of streets or areas.
  - [R30] The system must show the the data corresponding to the selection of [R29]

- [G4] People must be able to retrieve statistics and trends about the accidents correlated to the parking violations, the effectiveness of SafeStreets initiatives and the issuing of traffic tickets.
  - [R31] The system must take information about accidents and tickets from the municipality.
  - [R32] The system must use this information to build statistics, crossing them with reported violations.
  - [R33] The system must not allow users to see confidential data about other people.
  - [R34] The system must allow the user to choose a topic: areas or streets with most accidents, areas or streets with the highest number of traffic tickets issued, areas or streets where there have been the best improvements, information for a specific area or street.
  - [R35] The system must show to the user the information about the topic selected according to [R34].

- [G5] Municipality supervisors must be able to retrieve data about the vehicles with the highest number of violations.
  - [R36] The system must allow authenticated supervisors to retrieve information about the vehicles with the highest number of violations in a selected area or street.
  - [R37] The system must allow supervisors to access only information about their own municipality.
  - [R38] The system must allow a supervisor to create an account, asking the municipality services to verify its identity.
  - [R39] The system must allow a supervisor to login, inserting its username and password.
- [G6] Municipalities must be suggested for possible interventions about the mostly unsafe areas.
  - [R40] The system must take information about accidents, tickets and street networks (bike lanes, sidewalks, parking areas,...) from the municipality, exploiting the municipality services.
  - [R41] The system must elaborate this information, combined with reports information, and try to find possible solutions for problems.
  - [R42] The system must notify the municipality supervisors about the solutions it has found.

3.5 Performance requirements

Performance requirements are not particularly critical for the system, but it is anyway desirable that all requests sent to the server are answered within 1 second, to assure a good user experience. The server infrastructure will be designed to be scalable so that it will be possible to adapt it to the increment of users when the app diffusion will increase.

3.6 Design constraints

3.6.1 Regulatory policies

The application will only record the data strictly correlated to the reported violations and the data provided by the users during the registration. This data will be used only for the purposes of the system and will be treated confidentially, according to the *GPDR* rules.
In particular the statistical analyses performed will never show publicly any information which can be related to a specific person.

3.6.2 Hardware and software limitations

The following requirements are necessary to install the mobile application:

- *Operating system*: Android 5+ or iOS 9+
- *Hardware*: to allow the access as logged user the smartphone needs to have a camera and a GPS sensor

The camera is needed to take photos of the violations and the GPS is necessary to automatically record the position. The users have to give the relative permissions to the application. A base necessary requirement to use any functionality is the presence of an internet connection.
This requirements allow the majority of people to use the application (see *[OS-STAT]*).
The authorities have the possibility to use a web interface, accessible through every modern browser.
Everyone can consult the publicly available statistics also through the *SafeStreets website*, with any modern browser.

3.7 Software system attributes

3.7.1 Availability

The availability is not a critical requirement, but the system has to guarantee a 99% of uptime (max 3.65 days/year of downtime) to ensure that the users can normally use it.

3.7.2 Security

Security is a critical requirement for this system, considering the confidential information that are transmitted through it. It is assured by the use of the *HTTPS* protocol for all communications and by the follow of the best security practices for the servers management, protecting them with IDS, maintaining the data ciphered and assuring the access only to the authorized users.
Every activity performed by municipality agents and supervisors will be logged to ensure its traceability.

3.7.3 Maintainability

The system will be realized following the best software engineering practices to ensure its maintainability and expandability in the future.

3.7.4 Portability

The system is actually designed to be compatible with most of Android and iOS devices (smartphones and tablets) and from the authorities side it can be accessed from any web browser, so it is very portable. It will be important to maintain it compatible with the future releases of this two operating systems and with any other new operating system or device that will acquire an important market share.

4 Formal Analysis using Alloy

4.1 Introduction

In this section is shown a formal analysis of some aspects of the system using Alloy. The main purpose of this analysis is to formally define the constraints which allow the system to work properly, both external constraints (i.e. domain assumptions) and internal constraints (i.e. requirements).
The signatures defined do not have an exact correspondence with the classes of the Class Diagram (see 2.1), even if there is a good matching. The main differences are the absence of some fields or classes which are not useful for this modeling activity and the presence of some signatures (*StatisticsRequest* and the related *StatisticsLevel* and *RequestStatus*) which do not have correspondent classes in the previously shown uml diagram. This is due to the fact that they are technical classes, not necessary to be shown in the general abstract class diagram, whose purpose is just to show the domain of the application, but it can be useful to have them here as signatures to allow to express a particular constraint (i.e. the privacy levels of statistics data). There are some simplifications, in particular the use of the Alloy built-in type *Time* which easy allows to define time instants and check their order, but clearly does not have the same expressive power of a *DateTime* type of a high-level programming language.
This Alloy model manly consists of signatures and facts. A summary of what is expressed through the facts and a matching with the previously expressed requirements and domain assumptions (see 3.4 and 2.4.2) is the following:

- Every area has at least one city and every street can be only in areas of the same municipality [D3]
- The state evolution of a *Report* is consistent with the StateDiagram (2.2), every report is checked by a municipality agent of the same municipality where the violation has occurred [R15-22]
- Every *StatisticsData* is referred to an area or to a street, but not both [R29]
- The are no users with the same identity card number, fiscal code or username [R2]
- Every municipality has at least one street [the concept of Municipality impose this]
- Agents must check the reports in the order of emission [R20]
- An agents cannot issue traffic tickets in another city [R19, R21]
- For every approved report there is a correspondent ticket [R21]

- For every street where there have been at least two accidents caused by bad parked cars and two approved reports there must be a suggestion. This is an easy example to answer the requirement [R41]
- *ReservedStatistics* cannot be accessed by normal users [R33]
- Correctness of statistics data [R29,R30,R34,R35,R36]

This model does not easy allow to express assertions to be checked, because it already covers almost all about the requirements/assumptions which involves and there are no very interesting and non trivial properties which should be implied by it and so should be checked through assertions. Anyway there are a pair of assertions which state that the number of approved reports must be smaller or equal to the number of the tickets, for all agents and for all streets. This clearly follows by the fact that every approved reports has a consequent ticket and it's not a simple equality because the municipality can issue other tickets, not associated to reports, but still in the application data thanks to the information collected from the municipality for statistics elaboration.
At the end of the Alloy code there are some predicates which are then shown and explained.
The predicates are consistent and no counterexample has been found for the assertions, by the tool:

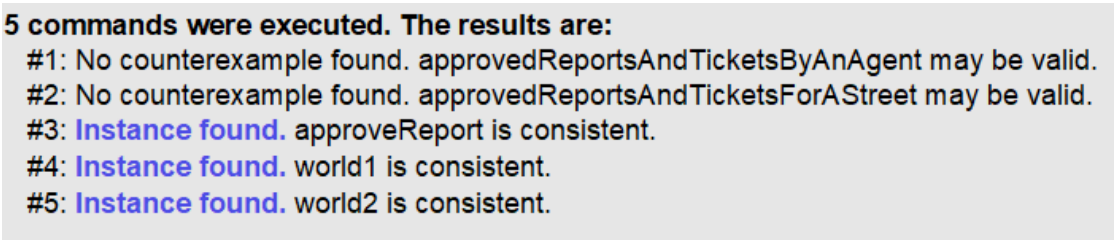

Figure 4.1: Alloy tool validation results

4.2 Alloy code

```
open util/time
open util/ordering[Time] as T0

abstract sig User{
        username : one String,
}

sig AuthenticatedUser extends User {
        identityCard : one String,
        fiscalCode : one String
}{
        identityCard ≠ fiscalCode
}
```

```
sig MunicipalityAgent extends User {
        municipality : one Municipality
}

sig MunicipalitySupervisor extends User{
        municipality : one Municipality
}

sig Municipality{ }

sig Street{
        area: some Area
}{
        --a street can belong to different areas, but they must be
           ↪  all areas of the same municipality
        no disj a1,a2:Area | a1 in area and a2 in area and a1.
           ↪ municipality ≠ a2.municipality
}

sig Area{
        municipality: one Municipality
}{
        some s:Street | this in s.area
}

sig LicensePlate {}

sig Report{
        author: one AuthenticatedUser,
        -- for simplicity the gps position is substituted by a
           ↪ direct association with the Street
        street: one Street,
        currStatus: one ReportStatus,
        statusHistory: Time -> lone ReportStatus,
        time: one Time,
        licensePlate: one LicensePlate,
        agent: lone MunicipalityAgent
}{
        --the currentStatus is the last status in the
           ↪ statusHistory
        currStatus = {s:ReportStatus | some t1:Time | (t1->s in
           ↪ statusHistory  and no t2:Time | (gt[t2,t1] and (some
           ↪  s2:ReportStatus | t2->s2 in statusHistory)))}
        --a ReportStatus cannot be present more than one time in
           ↪ the history
        no s:ReportStatus | some disj t1,t2:Time | t1->s in
           ↪ statusHistory and t2->s in statusHistory
        --the state evolution must be consistent with the state
           ↪ diagram (see fig. 2.2)
```

```
        no t1,t2:Time, s1,s2:ReportStatus | (gt[t2,t1] and (no t3:
            ↪ Time| gt[t3,t1] and lt[t3,t2] and t3 in
            ↪ statusHistory.ReportStatus) and t1->s1 in
            ↪ statusHistory and t2->s2 in statusHistory and (s2
            ↪ not in s1.nextStatus))
        WaitingAnalysis in Time.statusHistory
        --the statusHistory cannot start before the report time
        no t:Time | (t in statusHistory.ReportStatus and lt[t,time
            ↪ ] )
        --every checked report must have a corresponding agent
        (currStatus = Approved or currStatus = Refused or
            ↪ currStatus = WaitingControl) implies (agent ≠ none)
        --the agent of the report must work for the municipality
            ↪ which the report refers to
        (agent ≠ none) implies agent.municipality = street.area.
            ↪ municipality


}

abstract sig ReportStatus {
        nextStatus : set ReportStatus
}
one sig WaitingAnalysis extends ReportStatus{}
{
        #(nextStatus) = 2 and (WaitingAgent in nextStatus) and (
            ↪ Refused in nextStatus)
}
one sig WaitingAgent extends ReportStatus{}
{
        #(nextStatus) = 3 and (WaitingControl in nextStatus) and (
            ↪ Refused in nextStatus) and (Approved in nextStatus)
}
one sig Approved extends ReportStatus{}
{
        nextStatus = none
}
one sig Refused extends ReportStatus{}
{
        nextStatus = none
}
one sig WaitingControl extends ReportStatus{}
{
        #(nextStatus) = 2 and ( Approved in nextStatus) and (
            ↪ Refused in nextStatus)
}

sig Ticket{
        street: one Street,
```

```
        report: lone Report, --not all ticket are issued as a
            ↪ consequence of a SafeStreet report
        time: one Time,
        agent: one MunicipalityAgent,
        licensePlate : one LicensePlate
}

abstract sig StatisticsData{
        startTime : one Time,
        endTime: one Time,
        street: lone Street,
        area: lone Area
}{
        --a statistics must refer to a street or to an area and
            ↪ not both
        (street = none and area ≠ none) or (area = none and street
            ↪  ≠ none)
        gte[endTime,startTime]
}

sig PublicStatisticsData extends StatisticsData {
        num_report : one Int,
        num_tickets: one Int,
        num_accidents: one Int,
        num_park_accidents: one Int
}

sig ReservedStatisticsData extends StatisticsData {
        most_egregious_offender : one LicensePlate
}

abstract sig StatisticsLevel{}

one sig PublicStatistics extends StatisticsLevel {}
one sig ReservedStatistics extends StatisticsLevel{}

sig StatisticsRequest{
        user : one User,
        level : one StatisticsLevel,
        status : one RequestStatus
}

abstract sig RequestStatus{}
one sig RequestAccepted extends RequestStatus{}
one sig RequestRefused extends RequestStatus{}

sig Suggestion{
        street: one Street,
        time: one Time
```

```
}

sig Accident{
        street: one Street,
        cause: one AccidentCause,
        time : one Time,
        licensePlate: some LicensePlate
}

abstract sig AccidentCause{}

one sig HighSpeed extends AccidentCause{}
one sig DrugOrAlchol extends AccidentCause{}
one sig BadParkedCar extends AccidentCause{}
one sig Other extends AccidentCause{}

--uniqueness of some identification data

fact noDifferentUsersWithSameIdentityCard{
        no disj u1,u2: AuthenticatedUser | u1.identityCard = u2.
            ↪ identityCard
}

fact noDifferentUsersWithSameUsername{
        no disj u1,u2: User | u1.username = u2.username
}

fact noDifferentUsersWithSameFiscalCode{
        no disj u1,u2: AuthenticatedUser | u1.fiscalCode = u2.
            ↪ fiscalCode
}

--every municipality needs to have at least one agent and one
   ↪ supervisor
fact allMunicipalityHaveOneAgentAndSupervisor{
        all m:Municipality | ( ( some a:MunicipalityAgent | a.
            ↪ municipality = m) and ( some s:
            ↪ MunicipalitySupervisor | s.municipality = m))
 }

--every municipality needs to have at least one street (and so one
   ↪  area, given that every street belongs to at least one
fact allMunicipalityHaveOneStreet{
        all m:Municipality | ( some s:Street| s.area.municipality
            ↪ = m)
 }

--agents must check reports in order of emission
fact reportAnalysisOrder{
```

```
        no r1,r2:Report | gt[r2.time,r1.time] and (r1.currStatus =
            ↪  WaitingAnalysis or r1.currStatus = WaitingAgent )
            ↪ and (r2.currStatus not in WaitingAgent) and (r2.
            ↪ currStatus not in WaitingAnalysis)
}

--every ticket can be issued only by agents of the city where the
    ↪ violation occured
fact noAgentCanIssueTicketsInAnotherCity{
        no t:Ticket | t.agent.municipality & t.street.area.
            ↪ municipality = none
}

--for every approved report there must be one ticket with
    ↪ corresponding data, referred to that report
--and for every ticket referred to a report, this must be Approved
    ↪  and the data must correspond
fact ticketsForApprovedReports{
        all r:Report |r.currStatus = Approved implies one t:Ticket
            ↪  | ( t.licensePlate = r.licensePlate and t.report =
            ↪   r and t.agent = r.agent and t.street = r.street)
        all t:Ticket | t.report ≠ none implies one r:Report | (t.
            ↪ report = r and r.licensePlate = r.licensePlate and r
            ↪ .currStatus = Approved and t.agent = r.agent and t.
            ↪ street = r.street)
        all t:Ticket | t.report ≠ none implies gte[t.time,
            ↪ getReportApproveTime[t.report] ]
}

--return the approvation time for a report
fun getReportApproveTime[r:Report] : set Time{
        {t:Time | some s:ReportStatus | ( t->s in r.statusHistory
            ↪ and s=Approved) }
}

--all streets where there have been at least two accidents caused
    ↪ by parking or two approved violation reports must present a
    ↪ suggestion
--here, considering the Time type available, has been used the
    ↪ interval [t.prev,t] to find the events corresponding to a
    ↪ Suggestion at the time t
--clearly this should be a defined interval of time (eg: a month)
fact suggestionForUnsafeStreets{
        all s:Street, t:Time | ( #(getParkAccidentsInStreetAndTime
            ↪ [s,t.prev,t]) ≥ 2 and #(
            ↪ getApprovedReportsInStreetAndTime[s,t.prev,t]) ≥ 2 )
            ↪  iff  (some sg:Suggestion|sg.street=s and sg.time =
            ↪ t)
}
```

```
--return the set of Approved Reports of the Street s with time
    ↪ between t1 and t2
fun getApprovedReportsInStreetAndTime[s:Street,t1,t2:Time] : set
    ↪ Report{
        {r:Report | r.street = s and r.currStatus = Approved and
            ↪ gte[r.time,t1] and lte[r.time,t2]}
}

--reserved statistics can only be accessed by Municipality
    ↪ Supervisors
fact statisticsPrivacy{
        all sr:StatisticsRequest | sr.status = RequestAccepted iff
            ↪  ( sr.level = PublicStatistics or (one u:
            ↪ MunicipalitySupervisor | sr.user = u))
}

--ensure public statistics values coherence with reports,tickets
    ↪ and accidents data
fact publicStatDataCoherent{
        all psd:PublicStatisticsData |
                ((psd.street ≠ none) implies (
                        psd.num_report = #(
                            ↪ getReportsInStreetAndTime[psd.street
                            ↪ ,psd.startTime,psd.endTime]) and
                        psd.num_tickets = #(
                            ↪ getTicketsInStreetAndTime[psd.street
                            ↪ ,psd.startTime,psd.endTime]) and
                        psd.num_accidents = #(
                            ↪ getAccidentsInStreetAndTime[psd.
                            ↪ street,psd.startTime,psd.endTime])
                            ↪ and
                        psd.num_park_accidents = #(
                            ↪ getParkAccidentsInStreetAndTime[psd.
                            ↪ street,psd.startTime,psd.endTime])
                ))
                and
                ((psd.area ≠ none) implies(
                        psd.num_report = #(getReportsInAreaAndTime
                            ↪ [psd.area,psd.startTime,psd.endTime
                            ↪ ]) and
                        psd.num_tickets = #(
                            ↪ getTicketsInAreaAndTime[psd.area,psd
                            ↪ .startTime,psd.endTime]) and
                        psd.num_accidents = #(
                            ↪ getAccidentsInAreaAndTime[psd.area,
                            ↪ psd.startTime,psd.endTime]) and
                        psd.num_park_accidents = #(
                            ↪ getParkAccidentsInAreaAndTime[psd.
```

```
                                ↪ area,psd.startTime,psd.endTime])
                ))
}

--return the set of Reports for the Street s between t1 and t2
fun getReportsInStreetAndTime[s:Street, t1, t2:Time]: set Report{
        {r:Report | r.street = s and gte[r.time,t1] and lte[r.time
            ↪ ,t2]}
}

--return the set of Tickets for the Street s between t1 and t2
fun getTicketsInStreetAndTime[s:Street, t1, t2:Time]: set Ticket{
        {t:Ticket | t.street = s and gte[t.time,t1] and lte[t.time
            ↪ ,t2]}
}

--return the set of Accidents for the Street s between t1 and t2
fun getAccidentsInStreetAndTime[s:Street, t1, t2:Time]: set
    ↪ Accident{
        {a:Accident | a.street = s and gte[a.time,t1] and lte[a.
            ↪ time,t2]}
}

--return the set of Accidents caused by BadParkedCar for the
    ↪ Street s between t1 and t2
fun getParkAccidentsInStreetAndTime[s:Street, t1, t2:Time]: set
    ↪ Accident{
        {a:getAccidentsInStreetAndTime[s,t1,t2]| a.cause =
            ↪ BadParkedCar}
}

--return the set of Reports for the Area a between t1 and t2
fun getReportsInAreaAndTime[a:Area, t1, t2:Time]: set Report{
        {r:Report | a in r.street.area and gte[r.time,t1] and lte[
            ↪ r.time,t2]}
}

--return the set of Tickets for the Area a between t1 and t2
fun getTicketsInAreaAndTime[a:Area, t1, t2:Time]: set Ticket{
        {t:Ticket | a in t.street.area and gte[t.time,t1] and lte[
            ↪ t.time,t2]}
}

--return the set of Accidents for the Area a between t1 and t2
fun getAccidentsInAreaAndTime[a:Area, t1, t2:Time]: set Accident{
        {ac:Accident | a in ac.street.area and gte[a.time,t1] and
            ↪ lte[a.time,t2]}
}
```

```
--return the set of Accidents caused by BadParkedCar for the Area
    ↪ a between t1 and t2
fun getParkAccidentsInAreaAndTime[a:Area, t1, t2:Time]: set
    ↪ Accident{
        {ac:getAccidentsInAreaAndTime[a,t1,t2]| ac.cause =
            ↪ BadParkedCar}
}

--ensure reserved statistics values coherence with the tickets
    ↪ data
fact reservedStatDataCoherent{
        all rsd:ReservedStatisticsData |
                ((rsd.street ≠ none) implies (
                        let max = #(getTicketsForLPStreetTime[rsd.
                            ↪ most_egregious_offender,rsd.street,
                            ↪ rsd.startTime,rsd.endTime]) |
                        (all lp:LicensePlate | #(
                            ↪ getTicketsForLPStreetTime[lp,rsd.
                            ↪ street,rsd.startTime,rsd.endTime]) ≤
                            ↪  max)
                ))
                and
                ((rsd.area ≠ none) implies (
                        let max = #(getTicketsForLPAreaTime[rsd.
                            ↪ most_egregious_offender,rsd.area,rsd
                            ↪ .startTime,rsd.endTime]) |
                        (all lp:LicensePlate | #(
                            ↪ getTicketsForLPAreaTime[lp,rsd.area,
                            ↪ rsd.startTime,rsd.endTime]) ≤ max)
                ))


}

--return the set of Tickets for the LicensePlate lp, on the Street
    ↪  s, between t1 and t2
fun getTicketsForLPStreetTime[lp:LicensePlate,s:Street,t1,t2:Time]
    ↪  : set Ticket{
        {t:Ticket | t.licensePlate = lp and t.street = s and gte[t
            ↪ .time,t1] and lte[t.time,t2]}
}

--return the set of Tickets for the LicensePlate lp, in the Area a
    ↪ , between t1 and t2
fun getTicketsForLPAreaTime[lp:LicensePlate,a:Area,t1,t2:Time] :
    ↪ set Ticket{
        {t:Ticket | t.licensePlate = lp and a in t.street.area and
            ↪  gte[t.time,t1] and lte[t.time,t2]}
}
```

```
--reports approved by an agent are less than or equal to the
   ↪ number of tickets issued by that agent
assert approvedReportsAndTicketsByAnAgent{
        all a:MunicipalityAgent | #{r:Report | r.agent = a and r.
           ↪ currStatus = Approved} ≤ #{t:Ticket | t.agent = a}
}

check approvedReportsAndTicketsByAnAgent for 10

--the number of issued tickets for a street is less than or equal
   ↪ to the number of approved reports for that street
assert approvedReportsAndTicketsForAStreet{
        all s:Street | #{r:Report | r.street = s and r.currStatus
           ↪ = Approved} ≤ #{t:Ticket | t.street = s}
}


check approvedReportsAndTicketsForAStreet for 10

--approvement of a report
--the previously expessed facts imply the generation of the
   ↪ corresponding ticket and the respect of the status order
pred approveReport[r,r':Report] {
        (r.currStatus = WaitingAgent or r.currStatus =
           ↪ WaitingControl) ∧
        r'.author = r.author ∧
        r'.street = r.street    ∧
        r'.time = r.time ∧
        r'.licensePlate = r.licensePlate ∧
        r.statusHistory in r'.statusHistory ∧
        Approved in Time.(r'.statusHistory)
}

run approveReport for 5 but exactly 10 String, exactly 3 Time, 0
   ↪ StatisticsRequest, 0 StatisticsData

pred world1{
        #Municipality = 1
        #Street = 1
        #Report = 2
        #Accident = 2
        #Ticket = 2
        #LicensePlate = 2
        #StatisticsRequest = 0
        #PublicStatisticsData = 0
        #ReservedStatisticsData = 0
        some a1,a2:Accident, r1,r2:Report  | a1≠a2 and a1.cause =
           ↪ BadParkedCar and a2.cause = BadParkedCar and r1≠r2
```

```
            ↪ and r1.currStatus = Approved and r2.currStatus =
            ↪ Approved  and a1.time = a2.time and a1.time = r1.
            ↪ time and a1.time = r2.time
}

run world1 for 5 but exactly 5 String, exactly 3 Time

pred world2{
        #Municipality = 1
        #Street = 1
        #Area = 1
        #Report = 2
        #Accident = 1
        #Ticket = 1
        #LicensePlate = 2
        #StatisticsRequest = 1
        #PublicStatisticsData = 1
        #ReservedStatisticsData = 1
        some psd:PublicStatisticsData, t:Ticket,r:Report | t.time
            ↪ = psd.startTime and psd.startTime = r.time
}

run world2 for 5 but exactly 5 String, exactly 2 Time
```

4.3 Generated worlds

The predicate world1 generates a first world (figure 4.2) which shows the presence of a *Suggestion* as a consequence of the two *Approved Reports* and the two *Accidents* caused by *BadParkedCars* in the unique *Street* of the world, all over two consecutive time instants (not shown in figure, the diagram is projected over time for better readability). It can also be observed the presence of the two *Tickets* associated to the two *Approved Reports* and the correspondence of *MunicipalityAgents* for *Tickets* and relative *Reports*.
The second one (figure 4.3) shows Statistics information which are coherent with the tickets, reports and accidents data (it's still projected over Time, but it can be seen that all events happen at the same time).
Finally the predicate *ApproveReport* (figure 4.4) shows the change of state of a Report which becomes *Approved*, showing the consistency of the state evolution and the corresponding *Ticket* generated.

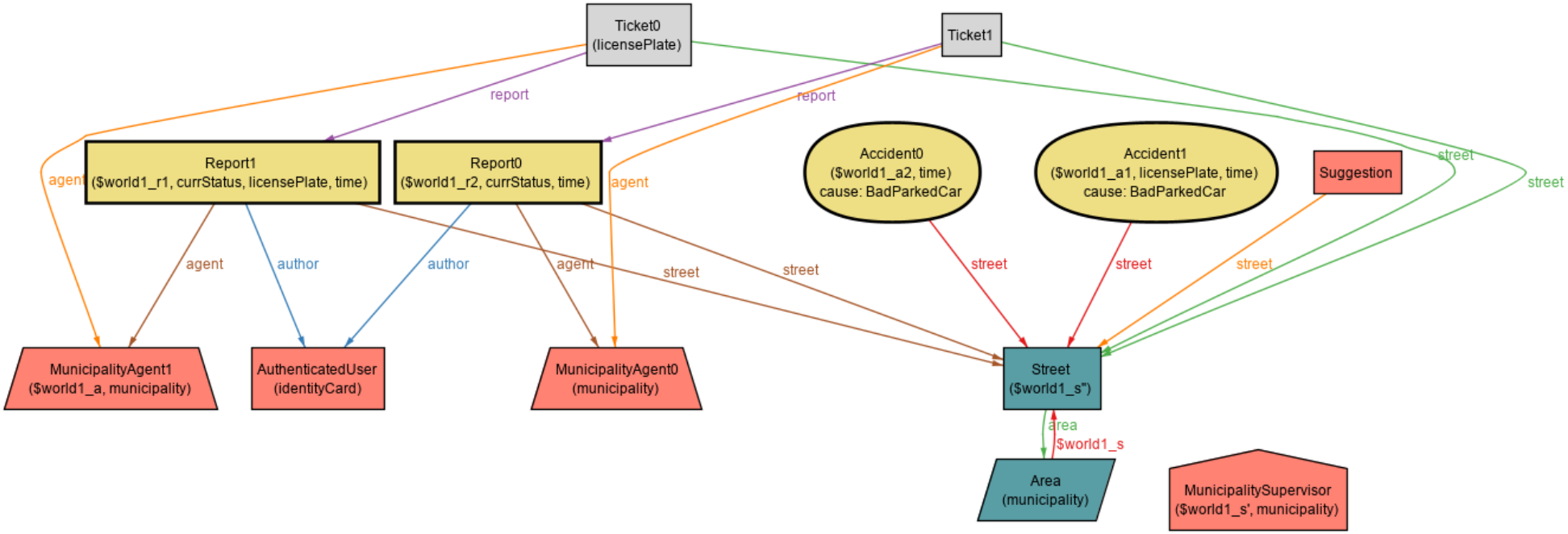

Figure 4.2: Alloy World n.1

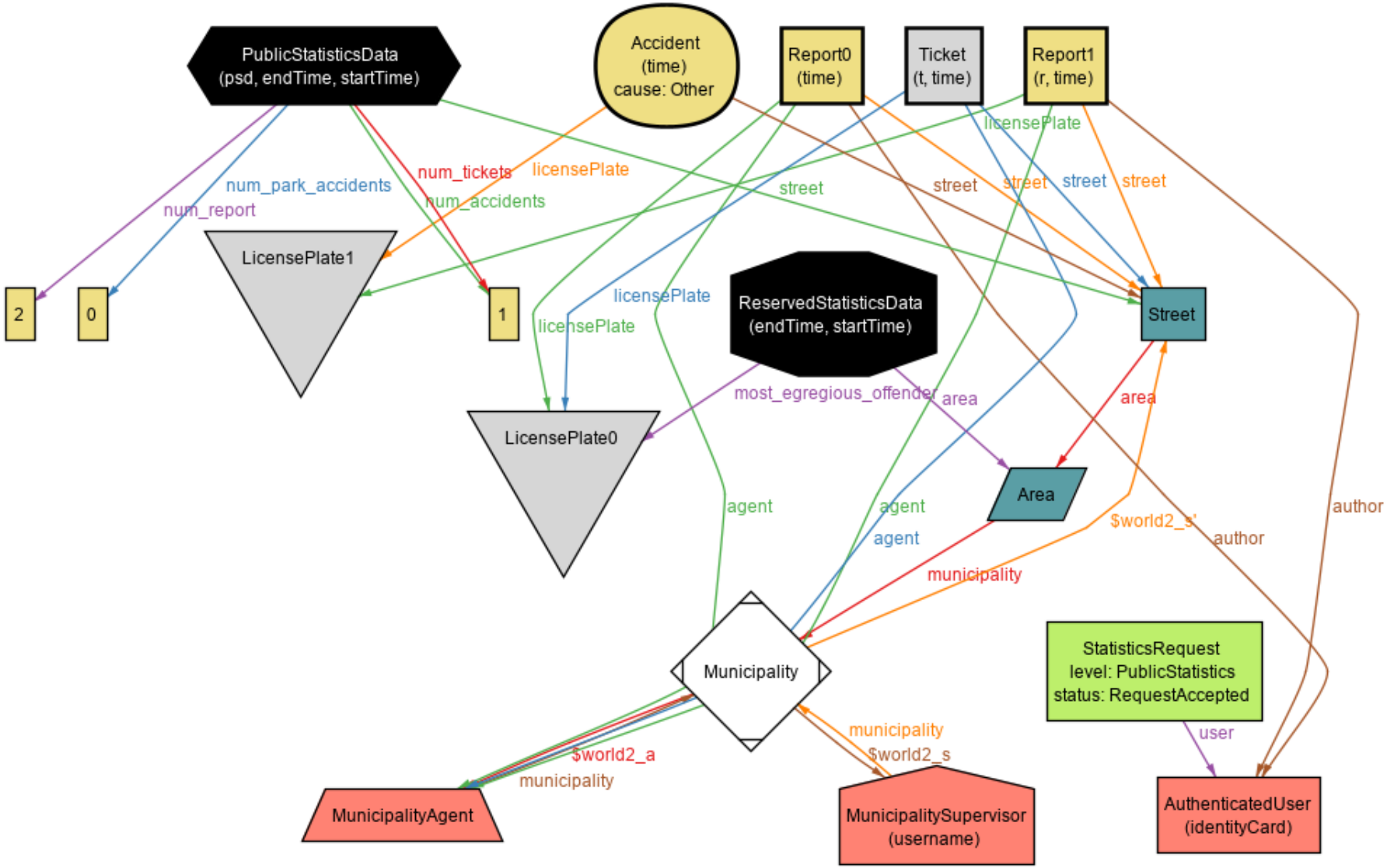

Figure 4.3: Alloy World n.2

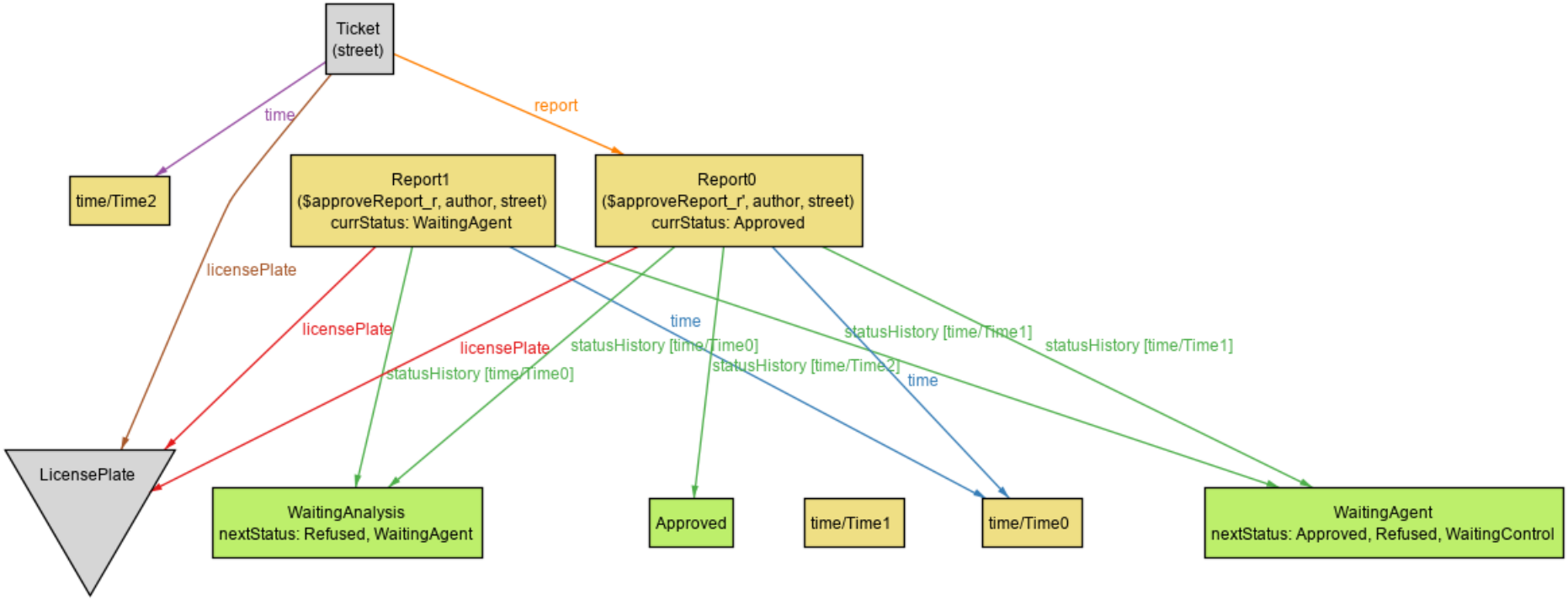

Figure 4.4: Alloy predicate ApproveReport

5 Effort spent

Nicola Rosetti

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Goals

Simone Sartoni

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Goals
20-10-2019	4.5 h	Goals revision and requirements outline
22-10-2019	3 h	Goals and requirements revision
23-10-2019	1.5 h	Use Case UML model design and draw
24-10-2019	2 h	Goal and requirements revision, functions
25-10-2019	2 h	Use Case UML model update
27-10-2019	1.5 h	Use Cases writing
29-10-2019	2 h	Use Case UML model and use cases revision
31-10-2019	1 h	Scenario and use cases revision
4-11-2019	1.5 h	Goals transcription on the RASD
5-10-2019	2.5 h	Requirements transcription on the RASD

Vittorio Torri

Date	*Hour*	*Section*
17-10-2019	1.5 h*	Goals
20-10-2019	1 h	Users, Hardware and software limitations, goal refinement
21-10-2019	1 h	Software, Hardware and Communication Interfaces, Performance requirements, Design constraints, Software system attributes
22-10-2019	1 h	Goal and requirements revision
23-10-2019	1 h	Goal and requirements revision
26-10-2019	2 h	Scenarios, requirements, functions, scope
28-10-2019	1.5 h	Product perspective and UML Class Diagram
29-10-2019	1 h	State diagram and sequence diagram
31-10-2019	0.5 h	Sequence diagram
31-10-2019	1 h*	Alloy
02-11-2019	1.5 h	Sequence diagrams & minor fixes
03-11-2019	4 h	Alloy
04-11-2019	1 h	Alloy
05-11-2019	1.5 h	Alloy
06-11-2019	1.5 h	Alloy
07-11-2019	1h	General revision
08-11-2019	2h	General revision
08-11-2019	2h	Alloy
08-11-2019	2h	Web interface mockup

** Group work*

6 References

- *[OS-STAT] https://gs.statcounter.com* - statistics on operating systems market share and versions diffusion
- *[SLANDER]* see art. 368, 370 and 69 c.p.p. (Italian law)