Davide Perugini, Antonio Pipita, Stefano Panzeri

SafeStreets

Requirement Analysis and Specification Document
Software Engineering 2 Project

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Davide Perugini, Antonio Pipita, Stefano Panzeri
Version:	1.0
Date:	10-November-2019
Download page:	https://github.com/fafrullo2/PanzeriPeruginiPipita
Copyright:	Copyright © 2019, Davide Perugini, Antonio Pipita, Stefano Panzeri – All rights reserved

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

The purpose of this project is to study the requirements and provide a specification about SafeStreets, a crowd-sourced application that permits users to notify authorities about traffic violations.

This document represents the requirements analysis and specification document of SafeStreets, where purposes, goals, requirements and assumptions of the applications will be defined to provide a support for the stakeholders.

SafeStreets allows users to send pictures of traffic violations with every useful metadata about it (date, time, position, type of violation, etc. . .) to authorities.

In addition, the application provides users and authorities with tools to mine the data collected, for example highlighting the streets where parking violations happen frequently, or the most unsafe areas.

SafeStreets also has the possibility to cross its own data with information about accidents coming from the municipality (if the municipality offers this information as an open service) to indentify unsafe areas with more precision and suggest possible interventions.

Finally, if the local police offers a service that takes data and pictures from SafeStreets to generate traffic tickets, the application must ensure the veracity of the information and use data about issued tickets to build statistics (effectiveness of the service, most dangerous vehicles etc. . .).

1.1.1 Goals

- (G1) Allow users to signal traffic violations.
- (G2) Allow users to include pictures in violation reports.
- (G3) The system must be able to retrieve geographical position and time of a violation and add them to report.
- (G4) The system must recognise (from pictures) license plates.
- (G5) The system must store user made reports.
- (G6) Allow users and authorities to mine information collected by the application.
- (GA1.1) The system must be able to cross the data collected with information about the accidents coming from the municipality.
- (GA1.2) The system must be able to suggest possible interventions to decrease the risk of violations in unsafe areas.

- (GA2.1) Allow the local police to retrieve data about violations to generate traffic tickets.
- (GA2.2) The system must be able to ensure the veracity of the information sent by the users.
- (GA2.3) The system must track wether the police has taken care of a certain violation.
- (GA2.4) Traffic tickets must be issued to the person that owns the vehicle that committed the violation.
- (GA2.5) The system must be able to build statistics from the information about issued tickets.

1.2 Scope

In a metropolitan city like Milan, one of the biggest problems is the overflowing stream of vehicles in the streets that comes and goes from universities, stations, work etc…

This is the perfect context in which an application like SafeStreets should be developed.

The application is thought for a world in which most of the people always brings along a smart device like a smartphone, able to take photographs and with a stable connection to the internet.

SafeStreets will allow every person of a city to collaborate to make streets safer and help police and municipality to identify areas where violations happen more often.

Data collected by the service, can be later mined by both users and authorities; this can be useful for users, in order to avoid dangerous streets or really messy car parks, and for authorities in order to strengthen controls in the most unsafe areas or to plan possible interventions.

In a world where everyone owns a smart device, it is really easy to modify images so, to avoid fake data sent by the users, SafeStreets will also implement several countermeasures to check the veracity of every piece of information.

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

- User: the customer of the application that exploit the service to send pictures and informations about traffic violations
- Municipality: the government of a city; it can mine data from SafeStreets to obtain information about traffic violations and statistics.
- Authorities: comprehend the municipality and the local police.
- Traffic ticket: a fee issued by the local police to people that own a vehicle that committed traffic violation
- Traffic violation: occurs when drivers violate laws that regulate vehicle traffic on streets or parking.

1.3.2 Acronyms

- RASD: Requirement Analysis and Specification Document

1.3.3 Abbreviations

- (Gn): n-Goal
- (G1.n): n-Goal for advanced function 1
- (G2.n): n-Goal for advanced function 2
- (Dn): n-Domain assumption
- (Rn): n-Requirement

1.4 Revision history

1.5 Reference documents

- Specification document: “SafeStreets Mandatory Project Assignment”

1.6 Document structure

- Chapter 1: an introduction to SafeStreets; it describes the purpose and the goals that the application aims to reach. It defines also the scope of the application, that includes the analysis of the world and of the shared phenomena.
- Chapter 2: provides an overall description of the system functionalities. It contains the various charts and diagram describing the domain, the most important requirements, the needs of the users, the various stakeholders and various constraints and assumptions.
- Chapter 3: provide a more specific study about the requirements of the applications describing the various interfaces needed (user, software and hardware), functional requirements with associated use cases and use case/sequence diagrams, performance requirements, design constraints and various software attributes (reliability, availability etc. . .).
- Chapter 4: provides a formal analysis of the application using Alloy. Here will be shown the alloy model of the most critical aspects, various comments to show how the project has been modelled and the world obtained by running the model itself.
- Chapter 5: information about the effort spent by every member of the team on the project.

2 Overall Description

2.1 Product Perspective

The aim of SafeStreets is to increase the security around the streets of the cities also simplifying the interventions of the authorities.
In order to make this possible, the application must include some functionalities that helps determining for a report is real or fake.
To monitor the position of the user sending the violation report, SafeStreets will exploit device's available GPS sensor and will also include a detailed map where users can discover the most unsafe areas or streets.
To allow users to take pictures of the violation, the application will use the camera nowadays inserted in every smartphone.
SafeStreets will also need a stable connection to the internet in order to send violation reports or to mine data from the database.

In order to differentiate functionalities for the various types of users, the application will be accessible by two different clients (basic users and authorities).
Basic users needs to insert personal informations (eg. SSN) during the sign-in process to reduce the number of false reports.
Authorities, on the other hand, must insert authentication data (eg. Police service number) during the sign-in process to ensure that the data won't be accessed by not authorised people.
Depending on the kind authority, SafeStreets offers different levels of data visibility.

2.1.1 Class Diagram

The high-level class diagram provides a model of the application domain, containing the most important classes that will be necessary to build the system.

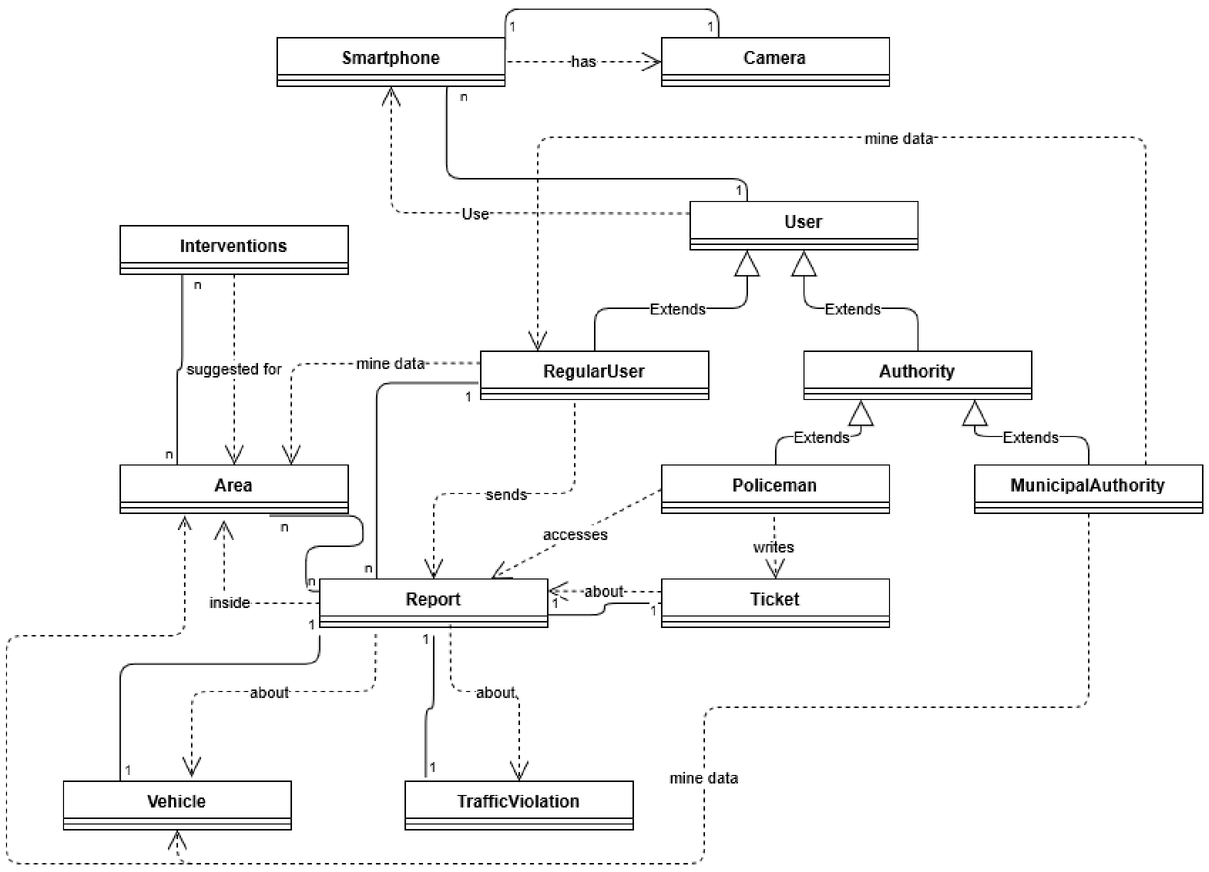


Figure 1: Class diagram

2.1.2 Statecharts

Now we will analyse the most critical aspects of the application workflow, modelling their behaviours using state diagrams.

Me: *inserts the image of the state diagrams*

2.2 Product Functions

In this section we analyse the main functions of SafeStreets:

- Data Management:

  The users must give the application the permission to access their data and device's functionalities, such as GPS position and Camera.
  They are also asked to insert personal data (eg. SSN) during the account registration phase.
  If these requirements are met, SafeStreets allows authenticated users to send violation reports including geographical position and pictures of the violation.
  All this data is then used to check the veracity of a report and to help the authorities to identify the violator.

On the other hand the authorities must insert authentication data in order to access the information stored by SafeStreets.
The system must be also able to use stored data about tickets issued by the local police to build statistics.

- Report management:

  One of the main features of SafeStreets is the possibility for users to send reports about the traffic violations that occurs around them.
  The system must be able to automatically decide wether a report must be discarded or kept in memory.
  The reasons for reports to be discarded are basically two: the report is fake (or not appropriate) or the same violation has already been reported (same photo for two different violations).

- user management:

  The system includes two different clients for the login of the different type of users.
  One is used by basic users that can send reports and discover the unsafe areas of the city, the other is used by registered authorities to mine data about reports or possible interventions to reduce violations.

- possible intervention suggestions

  SafeStreets must be able to automatically suggest possible interventions to avoid traffic violations in the most unsafe areas.
  The suggestions will be more accurate the more reports are sent from the same area.
  For example, if the applications receives a lot of reports about cars parked on the bike lane, there might be need of a barrier between the street and the lane.

- information crossing

  The system must be able to cross its data with informations about car accidents coming from the municipality.
  This functionality can be useful to identify the most dangerous areas or to accurately suggest interventions.

2.3 User Characteristics

Here is a list of the actors of the application:

- Basic users: Every authenticated user that is able to send violation reports and mine the application data to know the most unsafe areas.
- Local police users: Users with a police service number able to access data about reports sent by the basic users.
- municipalities: authorised users from the municipality of a city able to mine more detailed data such as the possible interventions suggested by SafeStreets.

2.4 Assumptions, dependencies and constraints

- (D1) Users' devices are connected to the internet
- (D2) Users' devices are equipped with a working camera

- (D3) GPS signal is always available on users' devices
- (D4) User device can provide a unique id (e.g. Google or Apple advertisement id)
- (D5) Only verified reports are accepted
- (D6) Users that want to access authorities' data must be acknowledged institutions
- (D7) Car's license plates are unique
- (D8) Every license plate is related to one and only one vehicle

3 Specific Requirements

3.1 External interface requirements

3.1.1 User Intefaces

3.1.2 Software Interfaces

3.2 Functional Requirements

(G0) Allow users to signal traffic violations (G1) Allow users to include pictures in violation reports (G4) The system must be able to retrieve the geographical position of a violation and add it to the report (G3) The system must recognise (from pictures) license plates (G5) The system must store user made reports (G2) Allow users and authorities to mine information collected by the application (GA1.1) The system must be able to cross the data collected with information about the accidents coming from the municipality. (GA1.2) The system must be able to suggest possible interventions to decrease the risk of violations in unsafe areas. (GA2.1) Allow the local police to retrieve data about violation to generate traffic tickets (GA2.2) The system must be able to ensure the veracity of the information sent by the users (GA2.3) The system must track whether the police has taken care of a certain violation (GA2.4) Traffic tickets must be issued to the person that owns the vehicle that committed the violation (GA2.5) The system must be able to build statistics from the information about issued tickets

3.2.1 Scenarios

1. **Bad Parking**: Bob is a commuter and every morning needs to reach the train station by car: the daily struggle to park his car in the surroundings is even worsened by parking violations. One morning, coming across another car occupying two parking slots, Bob decides to start contributing to SafeStreet service and its violation map: he downloads and installs SS app, accepts to grant Camera and GPS authorizations to it and fills a brief form for the registration. Once logged in, he uses the dedicated report feature. He takes a picture of the violation, inserts manually the license plate number and, after the application completes the verification of the inserted data, he sends the violation report, which also includes the GPS location and the current time.

2. **An anxious mother**: every day Laura lets his 12 years old son walk to school by himself. One day she hears about SS, a new app providing information about traffic violation in her city, and thinks that it can be useful to check if her son daily walking route is reasonably safe or not. Using SS violation map, which provides a street level highlight of violation distribution, she finds out that his son usually walks in potentially unsafe streets, with cars parked on sidewalks forcing pedestrians to walk in the middle of the street, and suggests him a new route based on the map provided by SS.

3. **Smart Municipality**: Milan municipality is looking for a large data set regarding traffic monitoring for his metropolitan area. The administration aims to mine useful information from it in order to identify critical areas and plan interventions. They found out SS and its violation DB, so they subscribe to the service as a public institution filling a required form and gain access to the needed data. With high privilege data access granted to public authorities, SS provide them a suitable amount of data for the administration goal.

4. **Traffic Monitoring Service**: Gotham City municipality has just introduced a new traffic plan in one of his main districts and wishes to track in an accurate way the evolution of traffic violations. SS should do the trick: Gotham municipality registers to SS service as public authority and now can have privileged access to the data collected by the application. However, Gotham City has no

infrastructure dedicated to large data set mining. No worries: the impact of the new traffic plan can be evaluated exploiting built-in SS violation map and traffic tickets emission trends.

5. **SafeStreet for Safer Cities**: the city of Monza provide a dedicated service offering data about incidents on its territory. SS has exploited its advanced features to cross its own information about traffic in Monza with the data provided, identifying possible unsafe areas and suggesting possible interventions. The administration of the city knows that managing the traffic plan of a vast city is complex and find that SS automated analysis could really lower effort and costs of the operations. Thus, the municipality decides to register to SS services and to exploit its traffic monitoring and intervention suggestion tool.

6. **Traffic Tickets**: NotSoSafe City is struggling trying to guarantee a decent level of violation control in his vast metropolitan area, considering his personnel available is limited. The municipality come to know that SS provide an efficient violation signalling service and a dedicated police officer client that even tracks ticket emissions. After registration as a public institution, the municipality issue to every policeman the installation of SS client on corporate (or personal) phone. Policemen can now take advantage of the real-time violation notification service of SS app for quick interventions if needed. Moreover, SS violation map offers the municipality a strong tool to simplify territory control: officer patrol activity can be directed to the most unsafe areas reported by the application.

7. **Policeman Intervention**: Anna is a police officer working for Rome municipality. The city administration has recently decide to exploit SS service in order to optimize traffic ticket emission. Anna is patrolling district one and suddenly receive a violation notification from SS client installed on her smartphone. The report is coming from a nearby street so Anna decide to take care of the report, marking that an officer has been dispatched through the dedicated app function. Once on place, Anna verifies that the violation reported has truly occurred and issues a traffic ticket. She finally marks the report as" taken care" within SS app and gets back to her patrolling activity.

3.2.2 Use Case Description

3.2.3 Use Case Diagram

3.2.4 Sequence Diagrams

3.2.5 Mapping on Requirements

3.3 Performance Requirements

3.3.1 Response time

Server side SS is a data intensive application. Big volumes of data will be written and read at the same time. Given the nature of the application itself, fast responses are essential in order to make the policemen do their job properly. On the data analytics side, instead, the responses can be slower.

List of the response times:

- report forwarded to application DB: 500 ms, medium priority
- response to policeman query on DB: 500ms, medium priority
- responses to policemen actions insertion in DB: 200 ms, high priority
- responses to regular users queries on data: 500 ms, low priority
- responses to municipal authorities data-mining actions: 1 min, low priority

3.3.2 Workload management

SS will need to be able to sustain a heavy workload of database transaction: there will be a lot of simultaneous read and write operations. The workload required will differ depending on the differnt types of users.

- **Regular users:** the number of data streams will vary greatly depending on the number of users and the size of the city immplementing SS. As a safety measure, supposing to implement SS in a city roughly se same size as Milano, the number of expected data streams could exceed 100'000 daily, considering both reads and writes
- **Authorities:** given the relative small number of municipal and police employees, a good extimate in a city the size of Milano could be around 2000 data streams daily on average, insignificant when compared to the other users' streams

In conclusion, it's evident that SS must be able to scale seamlessly and in an utomated fashion, without human intervention

3.4 Design Constraints

3.4.1 Standard Compliace

3.4.2 Hardware Limitations

3.5 Software System Attributes

3.5.1 Reliability

3.5.2 Avaibility

3.5.3 Security

3.5.4 Mantainability

3.5.5 Portability

4 Formal Analysis Using Alloy

This section contains the project analysis done using Alloy. The .als file can be found inside the RASD directory

4.1 Introduction

The alloy model is focused on the main entities and rules of SS:

- User-made reports
- Main actors
- Rules regarding user-made reports
- Rules regarding police intervention
- Rules regarding data mining rights

While, on the other hand, cross-database interactions have not been modeled and the relation between area and violation has not been modeled in its entirety, as the rules about how to assign a violation to an area are missing.
Please also note that, even though Authorities actually are users, this part has been omitted in the model for simplicity of description and analysis of the model itself.

4.2 Signatures

```
sig Boolean{}
sig True extends Boolean{}
sig False extends Boolean{}
sig Photo{}
sig Person{
}
sig Plate{}
sig Vehicle{
plate: lone Plate,
ownedby: one Person
}
sig User{
person: one Person,
areaOfInterest: one Area
}
sig GPScoords{
latitude: one Int,
longitude: one Int
}
sig Intervention{}
sig Area{
reportsInside: set Report,
dangerLevel: one Int
interventions: set Intervention
} {#interventions>0 implies #reportsInside>0}
```

```
sig PositionAndTime{
coords: one GPScoords,
time: one Int
}{time>=0 and time<7}
```

Note: the numbers related to time have been diminished in value for analysis performance reason

```
abstract sig ViolationType{}
sig ExpiredTicket extends ViolationType{}
sig UnauthorizedParking extends ViolationType{}
```

Note: UnauthorizedParking and ExpiredTicket are just two examples of the violations that may occurr

```
sig Violation{
vehicle: one Vehicle,
positionAndTime: one PositionAndTime,
violationType: one ViolationType,
photo: one Photo,
writtenPlate: one Plate
}
```

Note: vehicle represents the information retrived from the photo by the system, crossed with the database of car owners; on the other hand writtenPlate is the plate that the user that made the report wrote in it

```
sig Report{
maker: one User,
takenCareOf: one Boolean,
violation: one Violation,
dispatchedOfficer: lone Policeman
}
sig Authority{
person: one Person
}
sig MunicipalAuthority{
trackedUsers: set User
trackedArea: set Area
trackedVehicles: set Vehicle
}
sig Policeman extends Authority{}
sig Ticket{
segnalations: one Segnalation,
policeman: one Policeman,
issuedTo: one Person
}
```

4.3 Functions

```
fun getCoords [s:Segnalation]:GPScoord{
s.violation.positionAndTime.coords
}
fun getTime [s:Segnalation]:Int{
s.violation.positionAndTime.time
}
```

4.4 Facts

```
fact booleanValue{
#True=1 and #False=1 and #Boolean=2 and
(all b:Boolean | b=True or b=False) and
(no b: Boolean | b in True and b in False)
}
fact uniqueFoto {
all p1: Photo | no disj s1, s2 : Violation | s1.photo=p1 and s2.photo=p1
}
fact noLonePhoto {
all p1:Photo | p1 in Violation.photo
}
fact fact noSamePlate {
no disj vei1, vei2: Vehicle | vei1.plate=vei2.plate
}
fact noDoubleJob {
no p:Person | p in MunicipalAuthority.person and p in Policeman.person
no disj p1, p2: Policeman| p1.person=p2.person
no disj ma1, ma2: MunicipalAuthority | ma1.person= ma2.person
no disj u1, u2: User| u1.person=u2.person
}
fact cityLimits {
all gps: GPScoord | gps.latitude>0 and gps.longitude>0 and
gps.latitude<7 and gps.longitude<7
}
fact noDoubeCoordinates {
all c1: GPScoord | no c2: GPScoord | c1 != c2 and
c1.longitude=c2.longitude and c1.latitude= c2.latitude
}
fact areaProperties {
all a: Area| #a.reportsInside>=0 and
a.dangerLevel=#a.reportsInside
}
fact noMissmatchingPlates {
all v: Violation| v.vehicle.plate=v.writtenPlate
}
fact allPlates {
#Violation.writtenPlate=#Vehicle
}
fact violationTypeCardinality {
#ViolationType=2 and #ExpiredTicket=1 and #UnauthorizedParking=1
}
fact noViolationWithSamePhoto {
all disj v1, v2: Violation| v1.photo=v2.photo
}
fact noReportsDuplicate {
no disj r1, r2: Report| r1.violation=r2.violation
}
fact allSegnalationsInAnArea {
all s: Segnalation | s in Area.segnalationsInside
}
fact eitherAllTakenCareOrNone {
all s1, s2: Report | (getCoords[s1]=getCoords[s2] and
getTime[s1]=getTime[s2] and s1.violation.writtenPlate=s2.violation.writtenPlate
and s1.violation.violationType=s2.violation.violationType) implies
(s1.takenCareOf=s2.takenCareOf and s1.dispatchedOfficer=s2.dispatchedOfficer))
}
```

Note that in reality the constraints on location and time of violation would be different. As a matter of fact, SS would leave a margin for GPS coordinates and time of segnalations, but for analysis complexity reasons those bounds have been simplified in an equality relation.

```
fact sameVehicle {
all disj s1, s2 : Report | all t: Ticket |
(s1 in t.report and s2 in t.report) implies s1.violation.vehicle=s2.violation.vehicle
}
fact takenCareOfRule {
all s: Report | s in Ticket.segnalations iff s.takenCareOf=True
}
fact rightPersonBilled {
all t: Ticket| t.issuedTo=t.report.violation.vehicle.ownedby
}
fact noDoubleBilling {
all t1: Ticket| all s: Report| s in t1.report implies no t2:Ticket| t2 != t1 and s in t2.report
}
fact dispatchedOfficerWritesTheTicket {
all t: Ticket| t.policeman=t.report.dispatchedOfficer
}
fact ifTakenCareThenOfficerDispatched {
all r: Report| r.takenCareOf implies #r.dispatchedOfficer=1
}
fact noMunicipalDBNoTickets{
#Ticket>0 implies all v:Vehicle| v.ownedby=1
}
```

4.5 Assertions

4.5.1 G1

```
assert eachSegnalationHasOneAndOnlyPhoto {
all disj s1, s2: Violation|
#s1.photo=1 and #s2.photo=1 and s1.photo != s2.photo and #Photo=#Report
}
```

4.5.2 G2

```
assert dataMining {
all u: User| all ma: MunicipalAuthority|
#u.areaOfInterest>=0 and #ma.trackedAreas>=0 and
#ma.trackedUsers>=0 and #ma.trackedVehicles>=0
}
```

4.5.3 G3

```
assert everySegnalationHasOneAndOnlyPlate {
all disj s1, s2: Violation|
s1.writtenPlate=s2.writtenPlate iff s1.vehicle=s2.vehicle
}
```

4.5.4 G4

```
assert everySegnalationHasTimeAndPlace {
all s1: Violation|
#s1.positionAndTime=1 and #s1.positionAndTime.coords=1 and #s1.positionAndTime.time=1 and
#s1.positionAndTime.coords.latitude=1 and #s1.positionAndTime.coords.longitude=1
}
```

4.5.5 GA1.2

```
assert interventionsSuggested {
all a: Area| #a.interventions>=0
}
```

4.5.6 GA2.4

```
assert ticketToVehicleOwner {
all v:Vehicle | all s: Report | all t: Ticket|
(s in t.segnalations and v = s.violation.vehicle) implies t.issuedTo=v.ownedby
}
```

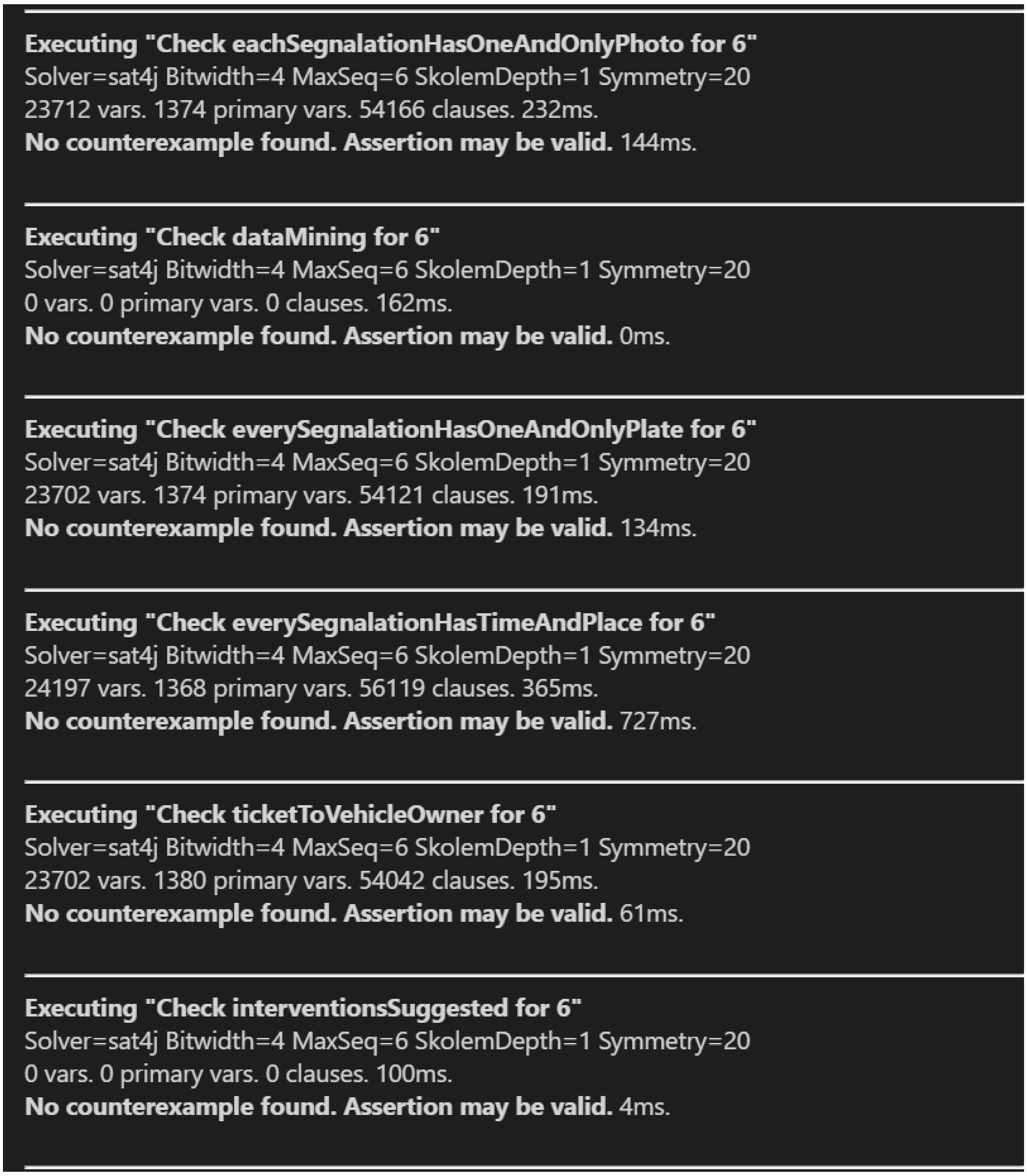
4.6 World

```
pred world1 {
#Vehicle=2
#Report=2
#Ticket=1
#Violation=3
#Policeman=1
#User=1
#MunicipalAuthority=1
#Person=3
#Area=1

    no p: Person| p in Policeman.person and p in User.person
}

run world1 for 6
```

```
check eachSegnalationHasOneAndOnlyPhoto for 6
check dataMining for 6
check everySegnalationHasOneAndOnlyPlate for 6
check everySegnalationHasTimeAndPlace for 6
check intervetionsSuggested for 6
check ticketToVehicleOwner for 6
```

4.7 Results

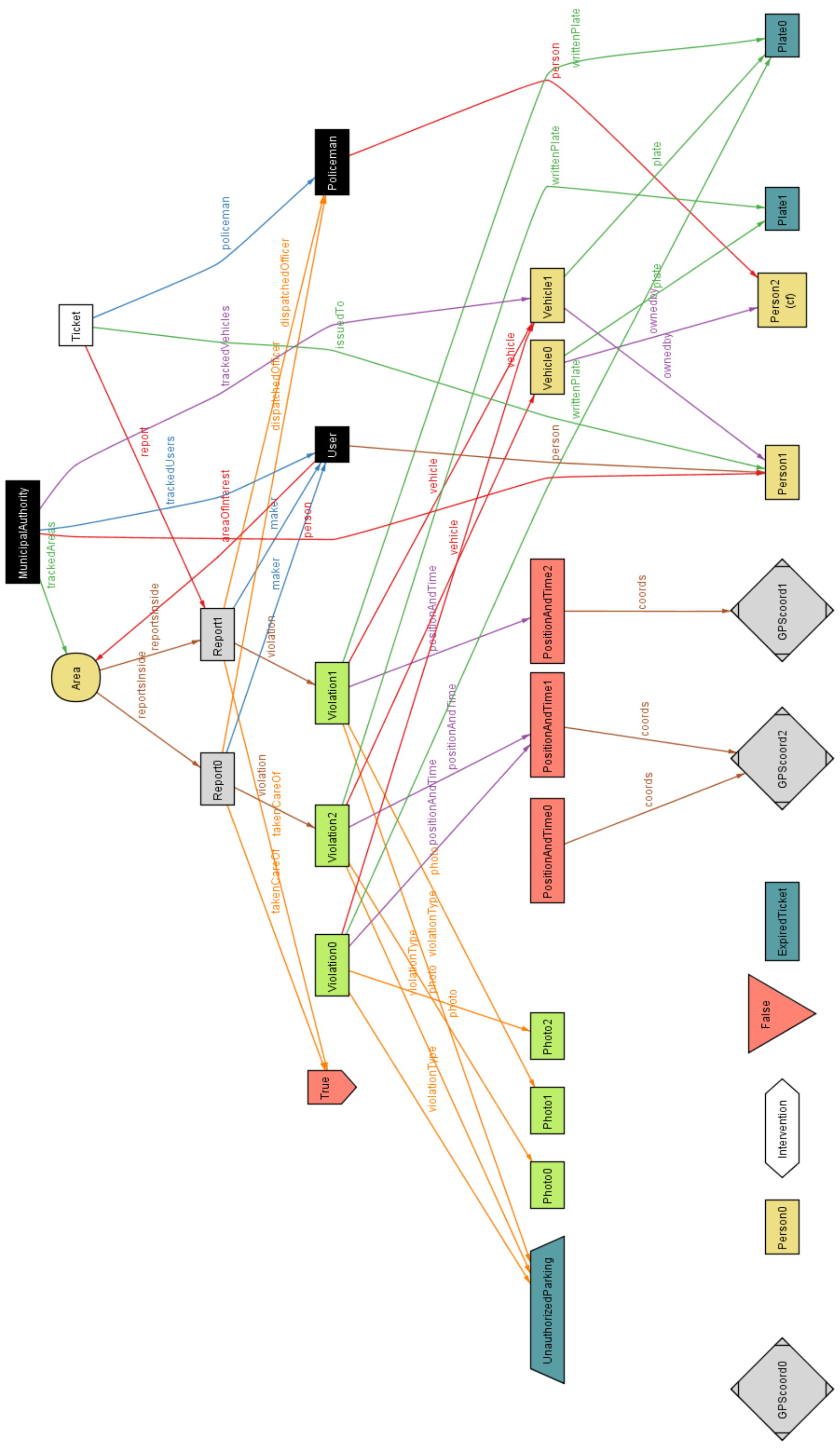
4.7.1 Checks on assertions



```
Executing "Check eachSegnalationHasOneAndOnlyPhoto for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
23712 vars. 1374 primary vars. 54166 clauses. 232ms.
No counterexample found. Assertion may be valid. 144ms.

Executing "Check dataMining for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
0 vars. 0 primary vars. 0 clauses. 162ms.
No counterexample found. Assertion may be valid. 0ms.

Executing "Check everySegnalationHasOneAndOnlyPlate for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
23702 vars. 1374 primary vars. 54121 clauses. 191ms.
No counterexample found. Assertion may be valid. 134ms.

Executing "Check everySegnalationHasTimeAndPlace for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
24197 vars. 1368 primary vars. 56119 clauses. 365ms.
No counterexample found. Assertion may be valid. 727ms.

Executing "Check ticketToVehicleOwner for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
23702 vars. 1380 primary vars. 54042 clauses. 195ms.
No counterexample found. Assertion may be valid. 61ms.

Executing "Check interventionsSuggested for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20
0 vars. 0 primary vars. 0 clauses. 100ms.
No counterexample found. Assertion may be valid. 4ms.
```

4.7.2 World generated

See next page.


MunicipalAuthority
Ticket
Policeman
User
Area
Report1
Report0
Violation1
Violation2
Violation0
True
Vehicle1
Vehicle0
PositionAndTime2
PositionAndTime1
PositionAndTime0
Photo2
Photo1
Photo0
UnauthorizedParking
Plate0
Plate1
Person2 (cf)
Person1
GPScoord1
GPScoord2
ExpiredTicket
False
Intervention
Person0
GPScoord0
policeman
issuedTo
report
trackedVehicles
trackedUsers
trackedAreas
areaOfInterest
reportsInside
dispatchedOfficer
maker
person
violation
takenCareOf
vehicle
positionAndTime
violationType
photo
writtenPlate
plate
ownedBy
coords

5 Effort Spent

Provide here information about how much effort each group member spent in working at this document. We would appreciate details here.

References

[1] S. Bernardi, J. Merseguer, and D. C. Petriu. A dependability profile within MARTE. *Software and Systems Modeling*, 10(3):313–336, 2011.