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 send pictures and informations about traffic violations.
- (G2) Allow users and authorities to mine information collected by the application.
- (G3) The system must recognise (from pictures) and store license plates.
- (G4) The system must be able to retrieve the geographical position where the violation occurred.
- (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.

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.

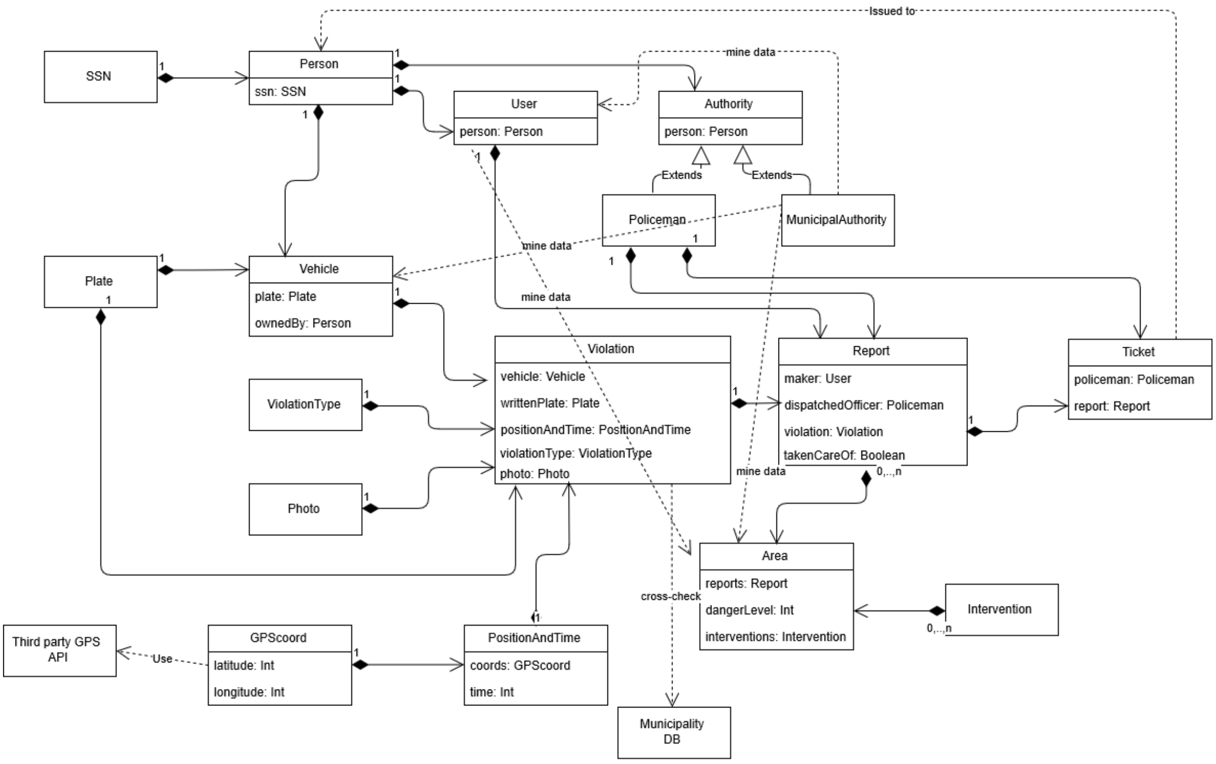


Figure 1: Class diagram

- 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) Every user can provide a unique 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 person

3 Specific Requirements

Organize this section according to the rules defined in the project description.

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 Signatures

```
sig Boolean{}
sig True extends Boolean{}
sig False extends Boolean{}
sig Photo{}
sig CF{}
sig Person{
cf: one CF
}
sig Plate{}
sig Vehicle{
plate: one 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 segnalation reports

```
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
}
```

4.2 Functions

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

4.3 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 noSameCF {
no disj p1, p2: Person | p1.cf=p2.cf }
fact getCoords {
}
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 }
```

4.4 Assertions

4.4.1 G1

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

4.4.2 G2

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

4.4.3 G3

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

4.4.4 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
}
```

4.4.5 GA1.2

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

4.4.6 GA2.4

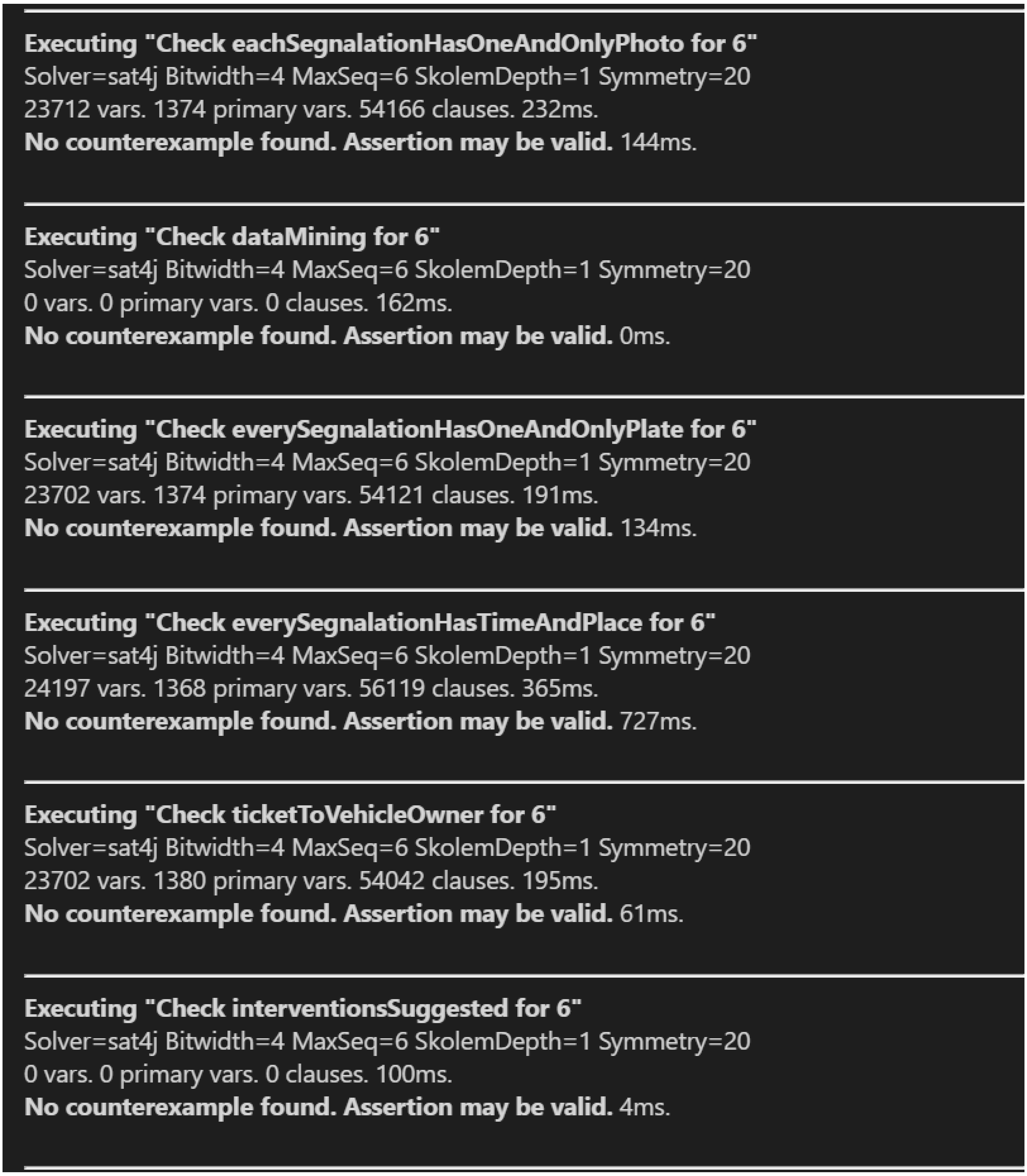
```
assert ticketToVehicleOwner {
all v:Vehicle | all s: Report | all t: Ticket|
(s in t.segnalations and v = s.violation.vehicle) implies t.issuedTo=v.ownedby
}
```

4.5 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
```

4.6 Results

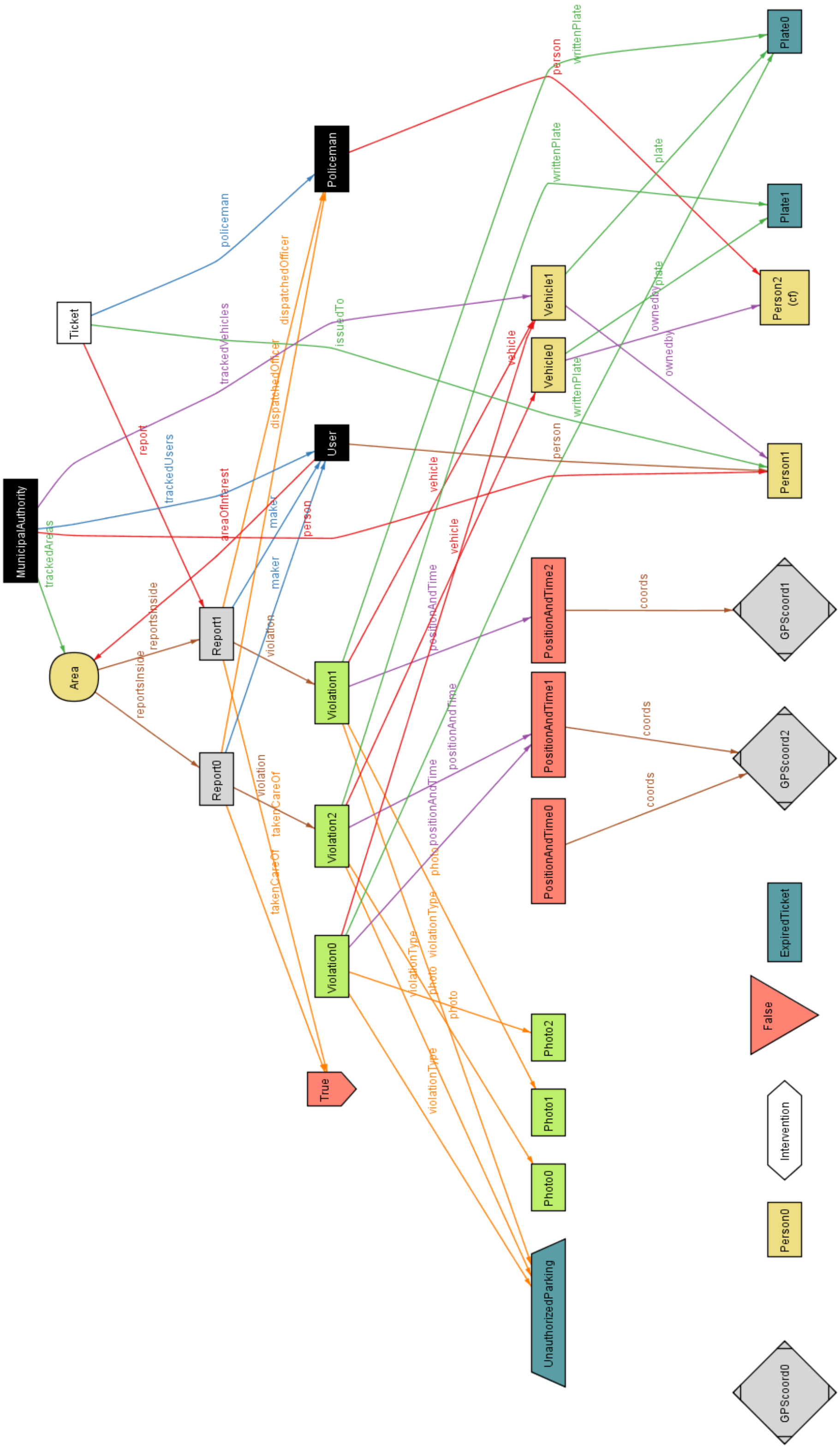
4.6.1 Checks on assertions



```
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.
```

4.6.2 World generated

See next page.


MunicipalAuthority
Ticket
Area
Report1
Report0
Policeman
User
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
trackedAreas
trackedUsers
trackedVehicles
areaOfInterest
policeman
issuedTo
report
dispatchedOfficer
reportsInside
maker
person
violation
takenCareOf
vehicle
positionAndTime
photo
violationType
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.