POLITECNICO DI MILANO
Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

TrackMe

Requirement Analysis and Specification Document
(RASD)

Reference professor:
Elisabetta Di Nitto

Mandatory Project:
Alessia Buccoliero, matricola 920484
Emilio Corvino, matricola 920429
Gianluca Drappo, matricola 920155

Anno Accademico 2018-2019

Version 1.0

Contents

Chapter 1

Introduction

1.1 Purpose

This document will provide an analysis of requirements and specifications of three new services the *TrackMe* company is willing to launch.
TrackMe is a company that wants to develop a software-based service allowing Third-parties to monitor the location and health status of individuals. Said services are described below.

1.1.1 Data4Help

Data4Help allows *TrackMe* to gather data from Users and to provide them to Third-parties upon request. In particular, it has the following goals:

[G1.1] Third-parties have to be able to monitor the location and health status of Individuals.

- [G1.1.1] Third-parties have to be able to request data of an Individual through his/her fiscal code.
- [G1.1.2] Third-parties have to be able to request data of groups of Individuals.
- [G1.1.3] Third-parties have to be able to subscribe to user data, specifying the frequency of updates and the desired granularity of data.

[G1.2] Individuals have to be able to decide whether to share their not anonymised data (gathered through smart devices) or not and to see what they are sharing to whom.

1.1.2 AutomatedSOS

AutomatedSOS lets Third-parties provide assistance to Elderly people in case of sudden illness by sending them an ambulance. Hence, its goals are:

[G2.1] Elderly people's health status has to be constantly monitored.

[G2.2] An ambulance has to arrive to Elderly people's location when their vital signs go below a certain threshold.

1.1.3 Track4Run

Track4Run allows the organisation of runs, the subscription to them and the tracking of runners through *Data4Help* gathered location. These are the objective the service aims to achieve:

[G3.1] Organisers have to be able to define the details of runs.

[G3.2] Runners have to be able to enrol to runs.

[G3.3] Visitors have to be able to track Participants' position on a map.

1.2 Scope

Nowadays being able to retrieve users data is very important for many companies operating in several fields (think about insurance, health, fitness...), since it allows them to provide assistance, tailor their services to the user, and so on. With the rising amount of smart devices capable of gathering data from the wearer, this necessity could easily be satisfied: smart-watches and fitness bands are being refined and they are also becoming more affordable.

On the other hand, there is the need to ask permission to the users to be allowed to exploit their data and they have to be informed about what is being gathered and who is requesting it, especially with the recent the introduction of GDPR (General Data Protection Regulation) in Europe.

TrackMe is willing to satisfy these necessities through ***Data4Help***, a service capable of balancing users' privacy with companies' need of data. This will be done allowing both Individuals and Third-parties to register to the service, so that the latter can perform requests over the former, which can accept or refuse them. Moreover, *TrackMe* wants to launch two more services exploiting ***Data4Help***'s framework:

- ***AutomatedSOS***, a software dedicated to Elderly people that will call an ambulance whenever the health signs of the person (gathered through ***Data4Help***) are below a critical threshold;
- ***Track4Run***, a software for runs organisation, where spectators can follow the participants thanks to the localisation provided by ***Data4Help***.

1.3 Definitions, Acronyms and Abbreviations

In this section the definitions, the acronyms and the abbreviations used throughout the document are explained in detail.

1.3.1 Definitions

- **Application, Software, System**: these terms refer to *Data4Help*, *AutomatedSOS* or *Track4Run*, depending on which of them is being described, in their entirety (design and implementation alike).
- **Critical threshold**: this expression refers to the values that, when trespassed, make *AutomatedSOS* call for an ambulance. (See [D2.2] in section 2.4.1).
- **Health status:** this expression refers to the status of the Individual inferred from the health signals gathered through smart devices.
- **Maps, Map service:** these terms refer to Google's map service, Google Maps.
- **Query, group search:** these terms refer to the data requested by Third-parties through *Data4Help* involving a group of Individuals.
- **Smart devices:** the ones taken into consideration are smart-watches and fitness-bands.
- **Subscription:** this term refers to the request of continuously updated data performed by a Third-party. Such data can belong to an Individual or may refer to a group search.
- **User:** this term refers to all possible customers of *TrackMe*, such as Individuals, Third-parties, Elderly people, Organisers, Runners, Visitors (see section 2.3 for further details).

1.3.2 Acronyms

- **R.A.S.D.:** Requirement Analysis and Specification Document.
- **A.P.I.:** Application Programming Interface.

1.3.3 Abbreviations

- **D4H:** *Data4Help*.
- **ASOS:** *AutomatedSOS*.
- **T4R:** *Track4Run*.
- **[D.1.k]:** *Data4Help*'s k-th domain assumption.
- **[D.2.k]:** *AutomatedSOS*' k-th domain assumption.
- **[D.3.k]:** *Track4Run*'s k-th domain assumption.
- **[D.2-3.k]:** *AutomatedSOS*' and *Track4Run*'s k-th domain assumption.

- **[G.1.k]:** *Data4Help*'s k-th goal.
- **[G.2.k]:** *AutomatedSOS*' k-th goal.
- **[G.3.k]:** *Track4Run*'s k-th goal.
- **[R.1.k]:** *Data4Help*'s k-th requirement.
- **[R.2.k]:** *AutomatedSOS*' k-th requirement.
- **[R.3.k]:** *Track4Run*'s k-th requirement.

1.4 Revision History

Version	Comments
1.0	First delivery of RASD

Table 1.1: Revision history table

1.5 Reference Documents

1. Specification document: *Mandatory Project Assignment A.Y. 2018-2019. pdf*
2. Software Engineering 2 course slides
3. Previous mandatory project examples:
   3.1. Specification document: *Mandatory Project Assignment A.Y. 2017-2018.pdf*
   3.2. *RASD to be analyzed.pdf*
4. 29148-2011 - ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes –Requirements engineering

1.6 Document Structure

The structure of this RASD documents follows IEEE standard for the most part, therefore it is divided into six chapters:

1. Introduction
2. Overall description
3. Specific Requirements
4. Formal Analysis using Alloy

5. Effort Spent

6. References

The overall aim of this document is to state what are the requirements of the software *TrackMe* wants to develop.

In the first chapter a general overview of the purpose of the software-to-be has been provided, together with the description of the context in which it is going to operate. Being an introductory chapter, all the terminology that will be used in the rest of the document has been clearly defined.

In the second chapter a broader description of the software is provided. Here, a class diagram that specifies the relations of the software-to-be with the world can be found, together with the detailed definition of the the assumptions made, the users of the software and its functions.

The third chapter delves into the details of the specification, talking about the various interfaces offered by the software-to-be, defining both its functional requirements through use cases, sequence and activity diagrams, and its non-functional requirements, along with the constraints it has to respect. In this part, differently from the IEEE standard, the description about Performance Requirements is included with the other System Attributes, in order to make the document more homogeneous.

The fourth chapter presents the formal analysis of the most critical parts of the software-to-be using an Alloy model, also showing a world obtained by running such model.

The fifth chapter contains the effort spent by each member of the group in order to realise the RASD.

In the sixth and last chapter all the sources of information exploited to write the present document are listed.

Chapter 2

Overall description

2.1 Product perspective

The application will offer some services based on data gathered from individuals. In particular, the three services are the following:

- The *Data4Help* service has as the main goal of providing user data to Third-parties, allowing them to know their position and health status. To do that, the Individuals have to collect their data through smartwatches and allow the system to store them. Upon successful requests, the system provides Third-parties with the already collected data and will send the new data according to the frequency of updates and granularity specified during the request compilation, if a subscription is performed. At any time, Third-parties can unsubscribe, in this way they won't receive updates but but they can still access already gathered data, relative to past requests.

- The *AutomatedSOS* service has as the main goal of offering an SOS service to Elderly people. To do that, the system accepts Elderly people subscriptions in order to collect and analyses data gathered by their *Data4Help* account. In case of critical health values, the system sends the exact position to the ambulance service, in order to send an ambulance to the rescue.

- The *Track4Run* service has as the main goal of organising runs. The system allows organisers to define all details of a run: path, starting date and time, maximum number of participants. After the creation of a run, people registered to *Data4Help* service can enrol to the race simply through their D4H account. In order to not exceed the maximum participants number, the software keeps track of the number of runners already enrolled. During the competition, everyone (an account is not needed) can monitor on a map the exact position of all runners; at the end of the race,

the system stores the rankings to make it available in the future to both participants and organisers.

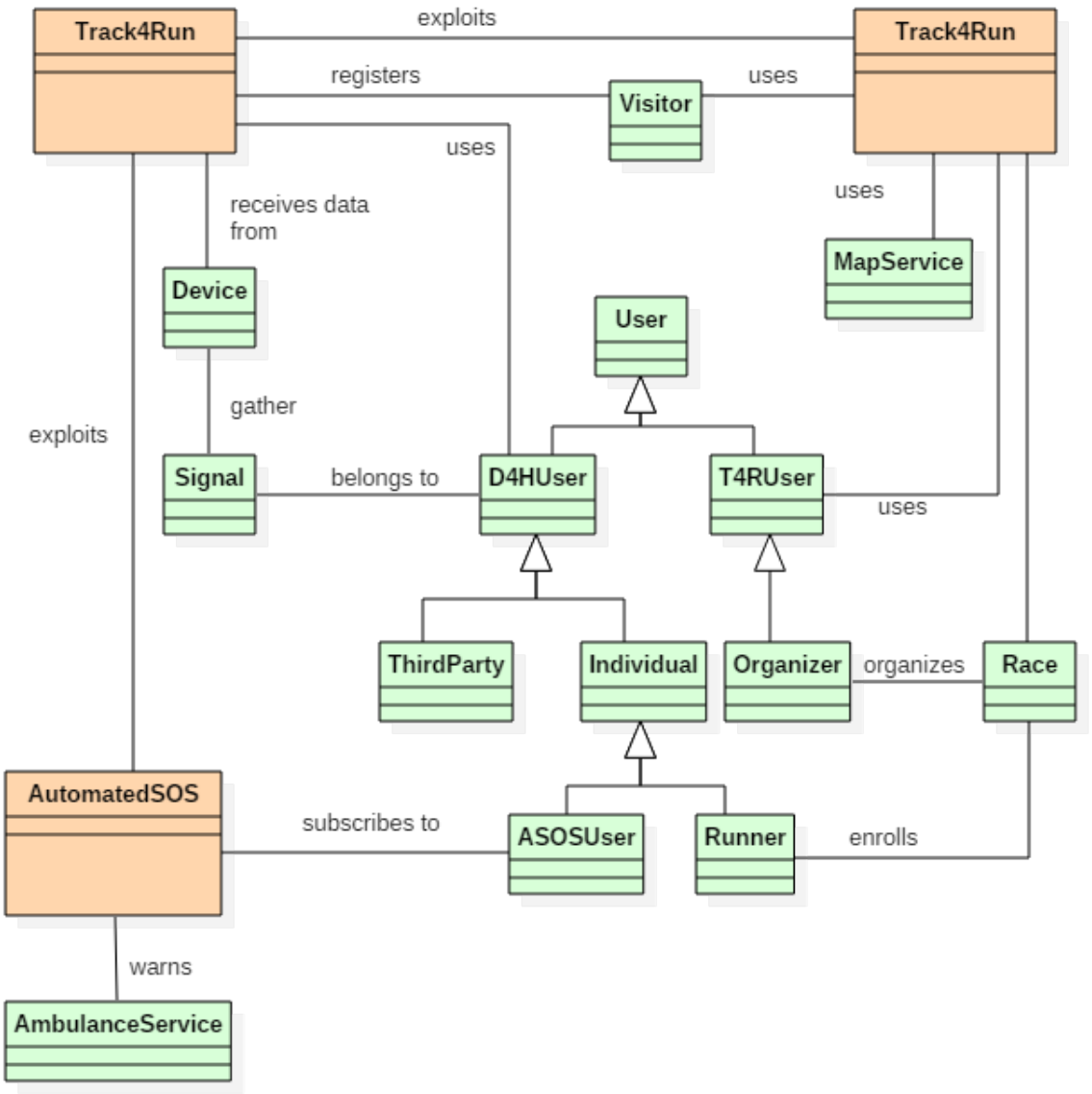

Figure 2.1: Class diagram of the entire system

2.2 Product functions

In this section there is an analysis of functional requirements (grouped by three main services) that the application needs to reach the main goals.

2.2.1 Data4Help

[R1.1] The system allows Individuals to register, requiring name, surname, fiscal code, password and email in order to create a new Individual account.

[R1.2] The system allows Third-parties to register, requiring the company name, VAT number, email and password in order to create a new Third-party account.

[R1.3] The application retrieves from smart-watches the following data, associating them to the user:

- beats per minute (bpm);
- number of steps;
- location;
- time-stamp of the measurement.

[R1.4] The frequency is 30sec

[R1.5] The system allows Third-parties to request data of Individuals and the latter to accept or refuse them.

[R1.5.1] If the Individual accepts, the system sends to the Third-party all the data gathered from the specified user.

[R1.5.2] If the Individual refuses or he/she is not registered, the system sends to the Third-party an error message.

[R1.6] Upon Third-parties request, the system can perform parametric searches based on geographical areas, age, genre, time of the day. The result of the query is then stored and evaluated for approval.

[R1.6.1] If the request is approved, the saved data is provided to the Third-party.

[R1.6.2] If the request is not approved, an error message is sent to the Third-party.

[R1.6.3] In order to anonimize data, the application only sends information related to the health status of individuals (e.g. bpm, step).

[R1.7] The application allows Third-parties to subscribe to certain data.

[R1.7.1] The application, if the Third-party subscribed to some data (belonging to Individuals or group searches), sends updates to the Third-party aggregating the data with the specified granularity and frequency.

[R1.8] The application allows Individuals withdrawals of consent for accessing their data and Third-party unsubscriptions.

[R1.8.1] If the Individual withdraws consent for the access of his data, a message is sent to the subscribed Third-party and data is no longer provided.

[R1.8.2] If the Third-party no longer wishes to collect data from an Individual, a message is sent to him/her and his/her data is no longer sent.

[R1.9] The system allows Third-parties to access to their request history and results.

[R1.10] The system allows Individuals to check gathered data.

2.2.2 AutomatedSOS

[R2.1] The system has to analyse user data and sends the location of the user to the ambulance service in case of emergency, when health values goes under the threshold.

[R2.2] The system gives the possibility to subscribe to the service upon request, only after checking the user age from his/her fiscal code.

2.2.3 Track4Run

[R3.1] The system allows Participants to log into the application through their Data4Help account and, after they are logged in, to enrol the the runs available.

[R3.2] The system allows the enrolling of Participants only if they are not enrolled in a race at the same time.

[R3.3] The system allows Visitors to access the application as host-users, without registration, by giving them only the possibility to track the Participants.

[R3.4] At the end of each run, the systems shows the rank to all Visitors watching it.

[R3.5] The system stores the ranks of finished races so that both Participants and Organisers can access to the race history.

[R3.6] The system allows the registration of Organisers without requiring also Data4Help registration.

[R3.7] In order to organise a run, the application requires track, data, time and the maximum number of participants.

[R3.8] The system needs to update subscriptions to the runs, decreasing the number of allowed participants. It must not allow further subscriptions if there are no leftover places.

[R3.9] The system does not allow that two or more runs are organised either at the same place or at the same time.

2.3 User characteristics

The main actors who interact with the application will be presented under this section.

- *Visitors*: They are not registered users, the system allows them to register through a sign up service either to Data4Help, becoming individuals or third-parties, or to Track4Run, becoming organisers. The application lets them also to be spectators of a run organised with Track4Run service.
- *Individuals*: They are single users registered to Data4Help service. It means that the system lets them sign in, accept or refuse Third-parties requests and monitor data acquired from them.
- *Third-parties*: They are companies that are registered to Data4Help. Also in this case, the application provides a sign in service, so as to let companies interact and perform single and groups requests.
- *Ambulance service*: The software has to interact with the ambulance service to provide a non–intrusive SOS service to Elderly people, in case of illness. The application sends the exact position where the ambulance has to go.
- *Elderly people*: They are Individuals, the software allows them to subscribe to an SOS service provided by AutomatedSOS, through their Data4Help account (they must have one). They are assumed to be over 65 years old.
- *Map service*: The application needs to interact with the map service, *Google Maps*, in order to let visitors and users see on a map the position of all runners during the run.
- *Organisers*: They are companies that are registered to Track4Run service. The system provides them a sign in interface and, after log in, lets them organise runs. They don't need to be registered to Data4Help.
- *Runners*: They are Individuals, the application allows to enrol in a race through their Data4Help account (they must have one).

2.4 Assumptions, dependencies and constraints

In this section all the domain assumptions made in order for the software-to-be to work are listed, together with the dependecies and the constraints it has to respect.

2.4.1 Domain assumptions

It is assumed that the following properties hold in the world.

Data4Help

[D1.1] Acquired data is precise enough.

[D1.2] Each Individual can be identified unambiguously through his/her fiscal code.

[D1.3] One anonymized, data cannot be associated to specific Individuals.

[D1.4] Every interaction gets correctly encoded.

[D1.5] Third-partied group queries are accepted only if the number of results is greater than 1000.

[D1.6] Third-parties are assumed to be companies that want to gather data, therefore they have a VAT code.

[D1.7] Third-parties know the fiscal code of the Individual they are looking for.

AutomatedSOS

[D2.1] An ambulance service capable of handling requests exists.

[D2.2] The thresholds beyond which a person is considered to be in need of help are under 40 bpm and above 130 bpm.

[D2.3] Elderly people are over 65 years old.

[D2.4] The service is offered in Italy (118 is the emergency number for the ambulance service).

Track4Run

[D3.1] Maps and tracking services are accurate enough.

[D3.2] The defined path exists.

2.4.2 Dependencies

It has to be noted that both *AutomatedSOS* and *Track4Run* need *Data4Help* in order to work. In particular, from the point of view of the two applications, these assumptions have to hold:

[D2-3.1] Customers (only Participants for *Track4Run*) have to be registered to *Data4Help*.

[D2-3.2] User data is gathered through *Data4Help*.

[D2-3.3] Data gathered through *Data4Help* is valid.

Moreover, the services refer to Google Maps for the provision of maps.

2.4.3 Constraints

A smart-watch or a fitness-band are needed in order to use the software and the health data provided is limited to what can be gathered from these devices and to what can be directly calculated from their measuring.

Chapter 3

Specific Requirements

3.1 External Interface Requirements

3.2 Functional Requirements

3.2.1 Data4Help

Individual registration/log in

Every user who wants to use *D4H* should be registered to the application System. After registration the user has to sign in, in order to collect data.

Actors	• Visitor • Individual
Goals	
Entry Condition	The user who wants to register must have a smart-watch or fitness-band
Events Flow	1. The Visitor accesses the application log in page. 2. The Visitor clicks on the "*Register*" button and then fills in all the mandatory information (name, surname, fiscal code, password and email) in order to create an Individual account. 3. The D4H System registers the user and sends back a confirmation email. 4. The Visitor has to associate his smart-watch or fitness-band to the account. 5. The Visitor becomes an Individual, inserting its email and password in the log in page.
Exit Condition	Now the user has his personal account.
Exception	The Registered user is not able to sign in the System because the ID (fiscal code for individual) or password are wrong or if he did not confirmed his email. In these situations, the system will show the user an error message.

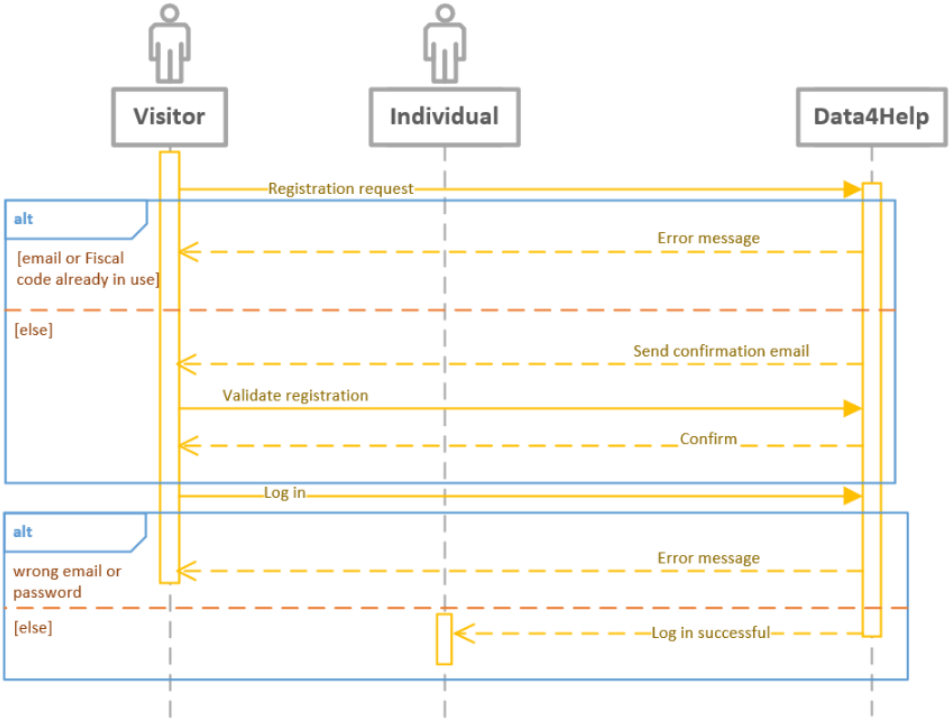

Figure 3.1: Sequence diagram representing the registration/log in phase of an Individual.

Third-party registration and log in

In order to have access to the functionalities of the application, Visitors have to register to the service as Third-parties. After that, they will be able to log into the application using their email and password and to perform their researches.

Actors	• Visitor • Third-party
Goals	G1.1, G1.2, G1.3, G1.4
Entry Condition	There is no entry condition.
Events Flow	1. The Visitor accesses the registration page. 2. The Visitor clicks on the "*Register*" button and then fills in all the necessary information useful to be recognised as Third-party (company name, VAT code, email and password). 3. *Data4Help* creates a new account and sends a confirmation email to the just registered Third-party. 4. The newly registered Third-party can now log into the application using the email and password previously provided, so to have access to the available features.
Exit Condition	The Third-party is registered and able to log into *Data4Help*.
Exception	The Visitor is not able to register because the email or the VAT code are already in use. A registered Third-Party is not able to log in because the email or password inserted are wrong or the email has not been confirmed. These exceptions will be handled by the system sending an error message.

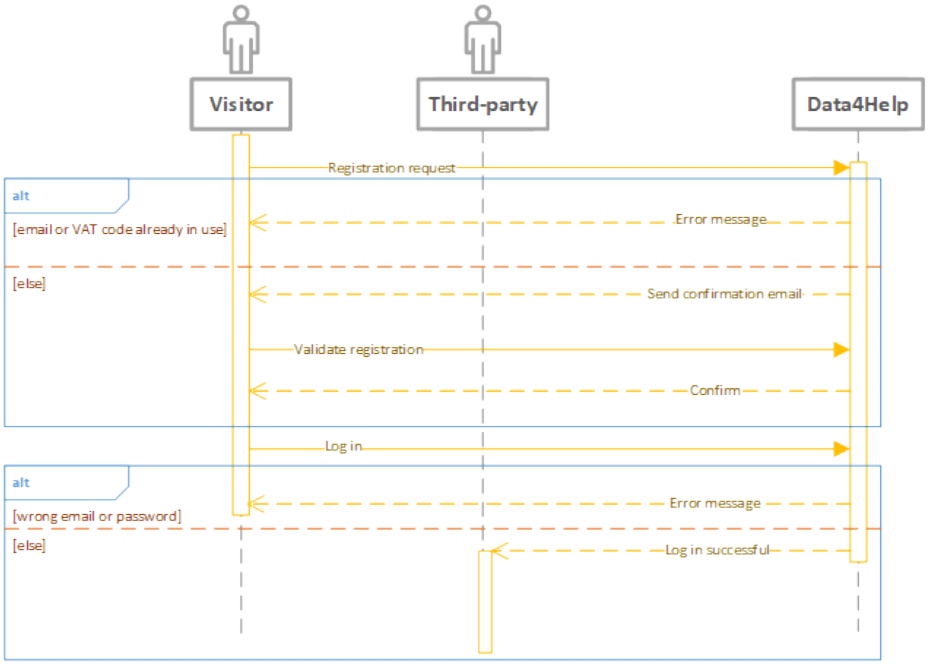

Figure 3.2: Sequence diagram representing the registration/log in phase of a Third-party.

Single user data request and subscription

Third-parties can perform a request to Data4Help for a specific users by specifying his/her fiscal code. Data4Help forwards it to the User that can accept or refuse it. Then, in case of positive answer, Data4Help sends the user's health data to the applicant. If it is not, it sends a notification to the third-party informing it about the refuse.

Actors	• Third-party • Individual
Goals	[G1.1.1]
Entry Condition	Third-party has to be registered to Data4Help.
Events Flow	1. The Third-Party selects "search by Fiscal Code" and it inserts the Fiscal Code of the desired person. 2. The System receives the request. 3. The System looks for the user connected to the Fiscal Code requested and forwards him/her the request. 4. The User receives the request with the name of who is requesting his/her data. 5. The User can accept or refuse the request. By accepting it, he/she allows the System to send all his/her health data to the applicant.
Exit Condition	The Third-party has the user's health data and receive updates of them periodically.
Exception	The Fiscal Code requested doesn't match with any registered user. In this case, the system notifies it with a pop-up in the third-party application. In addition, the user may refuse the request. Then the System send a message to the Third-party notifying it about that.

Single User data unsubscription

After a successful subscription request, both Third-party or Individual can make a request in order to break the subscription to the specified data.

Actors	• Third-party • Individual
Goals	[G1.2]
Entry Condition	Third-party is logged in and has previously subscribed to the Individual's data.
Events Flow	1. The Third-Party or Individual access to the subscription page and select the subscription to be cancelled. 2. The Third-party or the Individual presses "*Unsubscribe*" button. 3. The system deletes the Third-party subscription and sends an email to both Third-party and Individual.
Exit Condition	The Third-party is not subscribed anymore, but it has access to the previously received data.
Exception	There are no exceptions.

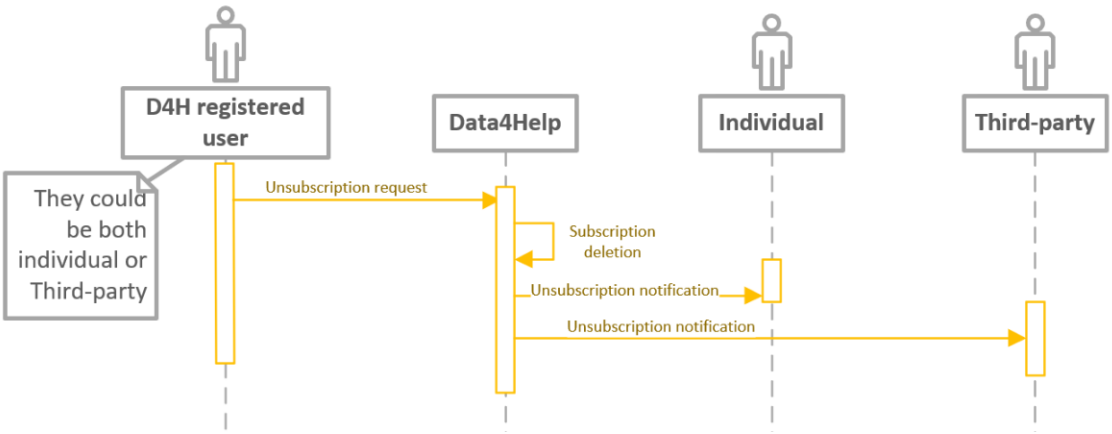

Figure 3.3: Sequence diagram representing the Unsubscription process for an individual data subscription.

Aggregated user data request and subscription

Third-parties can perform group queries, asking for data coming from several Individuals that are within certain parameters specified by them (geographical areas, age, genre, time of the day). These requests are evaluated directly by D4H, that sends back the requested data only if the number of Individuals involved in the research is greater than 1000. If this is not the case, D4H sends an error message to the Third-party.

Actors	• Third-party
Goals	[G1.1], [G1.1.2]
Entry Condition	The Third-party is logged in.
Events Flow	1. The Third-party clicks on the "*Aggregated data research*" button. 2. The Third-party inserts the parameters of interest. 3. The Third-party decides whether it wants to subscribe to this particular research by checking the specific box or not, then clicks on the "*Search*" button to have D4H perform the query. 4. *Data4Help* does the research and evaluates the results. 5. D4H sends back the result to the Third-party.
Exit Condition	The Third-party receives the requested information.
Exception	The number of Individuals involved in the query is lower than 1000, therefore D4H sends an error message back to the Third-party.

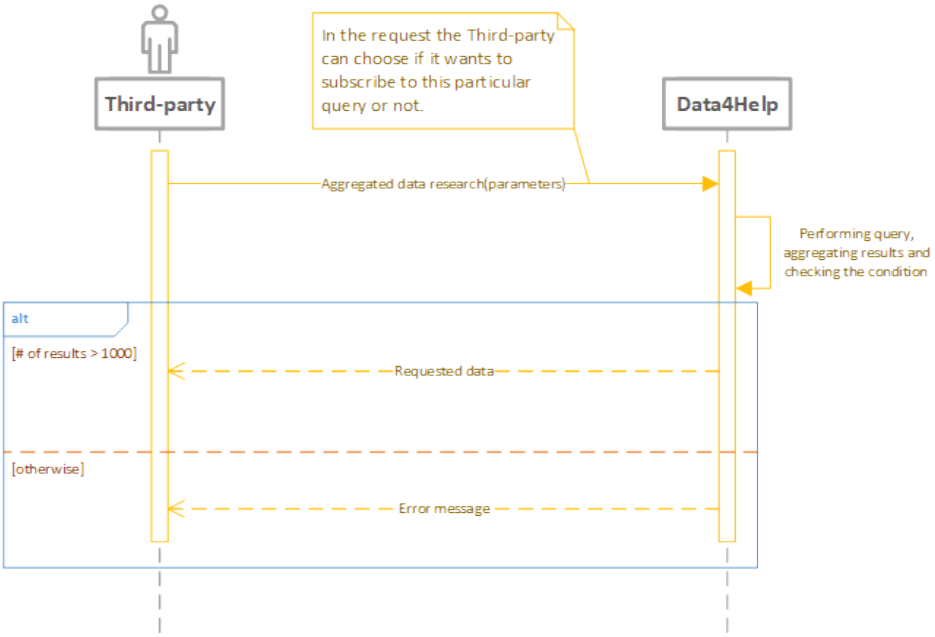

Figure 3.4: Sequence diagram representing the group requests performed by the Third-party.

Aggregated user data unsubscription

After a successful subscription to a group search, a Third-party can cancel it at any time if it does not wish to receive any further update. He will, however, still have access to the data collected until that point.

Actors	• Third-party
Goals	
Entry Condition	The Third-party is logged in and has previously subscribed to a group search.
Events Flow	1. The Third-Party accesses to the subscription page and selects the subscription to be cancelled. 2. The Third-party presses the "*Unsubscribe*" button. 3. The system deletes the Third-party subscription and sends an email to it for confirmation.
Exit Condition	The Third-party is not subscribed anymore, but it has access to the previously received data.
Exception	There are no exceptions.

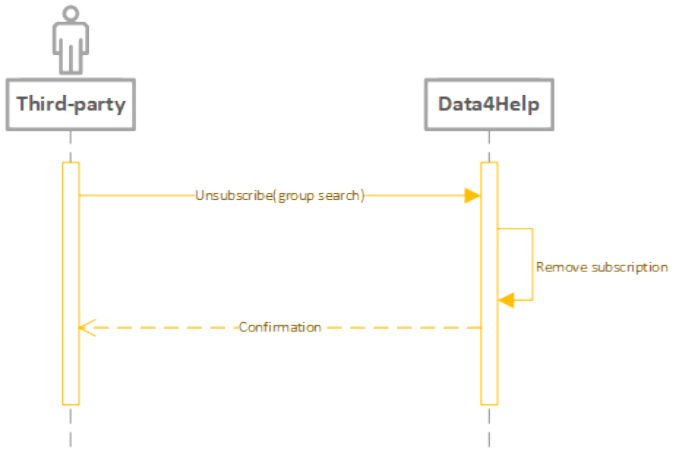

Figure 3.5: Sequence diagram representing the unsubscription from a group search by a Third-party.

Data Collection

Upon registration, Individuals allows the D4H system to collect their data and store them. To gather data, it's mandatory to have a device (smart-watch or fitness-band).

Actors	• Individual
Goals	
Entry Condition	The user is correctly registered and logged into the D4H system.
Events Flow	1. Smart-watches or fitness-band samples specific data. 2. This device sends acquired data to D4H system. 3. The system stores data and makes them visible to the owner Individual.
Exit Condition	The individual has personal data storage in his D4H account.
Exception	If the associate device (smart-watch or fitness-band) does not work or does not send data anymore the exception will be handled by the system by sending an error message.

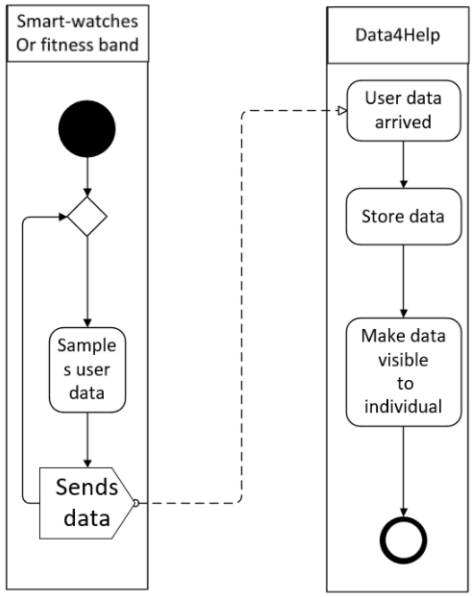

Figure 3.6: Activity diagram representing data storage process.

3.2.2 Automated SOS

Subscription Request

Every Individual, user registered to D4H, can ask for subscription to Automated SOS service, but only Elderly people will be allowed to use it.

Actors	• Individual
Goals	
Entry Condition	The user has a D4H account.
Events Flow	1. The Individual presses on "*AutomatedSOS service*", in order to make a new subscription request. 2. The ASOS System checks if the petitioner is at least 65 years old. 3. If verified, the user gets subscribed.
Exit Condition	The Individual is now subscribed to Automated SOS service.
Exception	If the petitioner is less then 65 years old, the system handles the exception, sending him an error message.

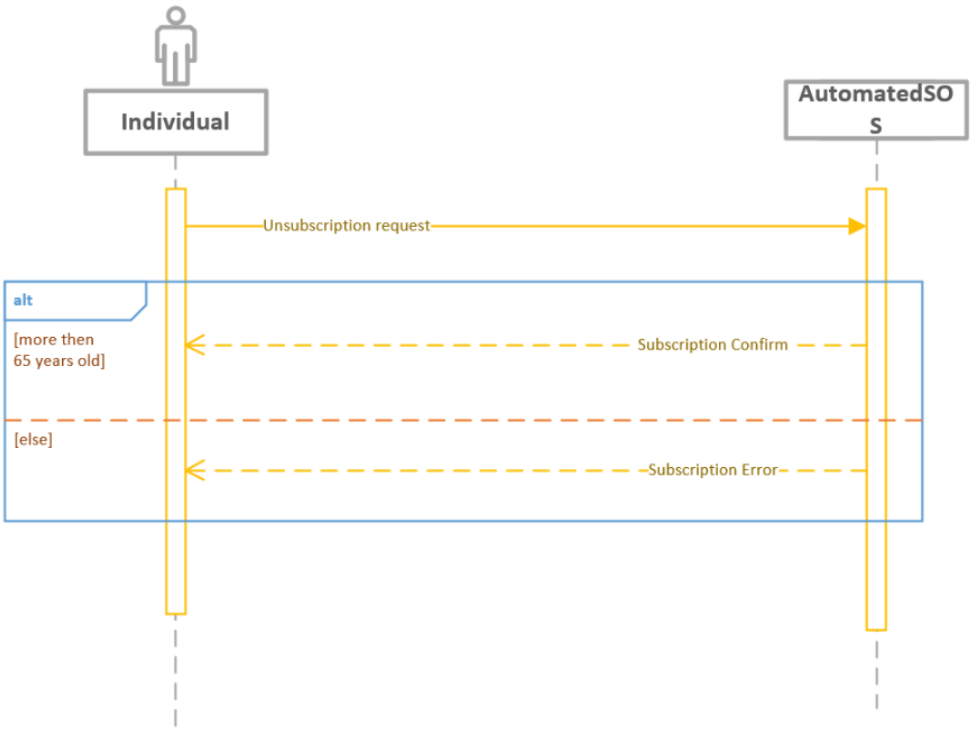

Figure 3.7: Sequence diagram representing the subscription request for ASOS service.

Ambulance dispatching

Whenever the system receives a value that goes beyond the critical threshold, ASOS contacts the ambulance service, asking for assistance.

Actors	• Elderly person • Ambulance service
Goals	
Entry Condition	The user has an ASOS account and is logged in.
Events Flow	1. D4H collects through the connected device the Elderly person's data. 2. ASOS checks if the values are under the critical threshold. 3. If so, ASOS contacts the ambulance service, giving the location of the Elderly person gathered with the last measurement. Otherwise, it does nothing. 4. The ambulance service sends an ambulance to the Elderly person location.
Exit Condition	The ambulance service has been warned and an ambulance in arriving at the Elderly person place.
Exception	There are no exceptions.

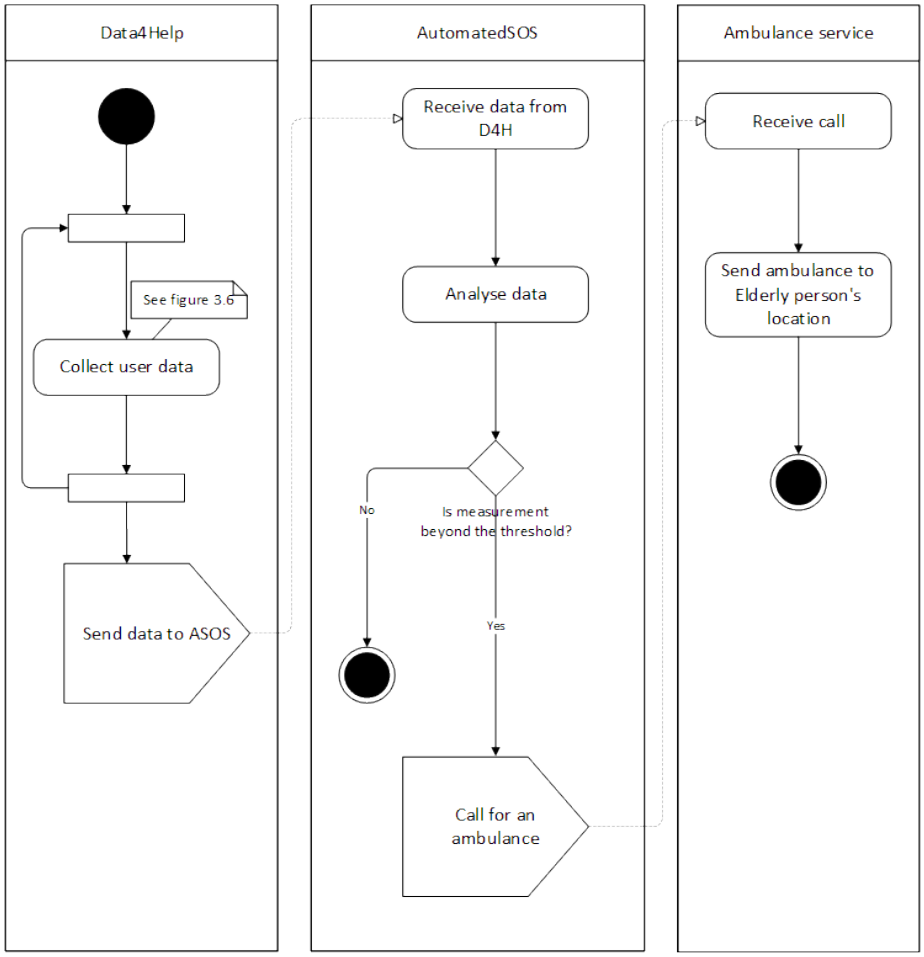

Figure 3.8: Activity diagram representing the call for an ambulance that ASOS performs when a critical measurement is collected by D4H.

3.2.3 Track4Run

Organiser registration and log in

Any Organiser needs to register to T4R before being able to organise and publish races. This account is in no way dependant to D4H and Organisers do not need to be registered to D4H too. Therefore, if they are already registered to D4H as Third-parties, they still need a new account, but can use the same information provided for D4H.

Actors	• Visitor • Organiser
Goals	
Entry Condition	There is no entry condition.
Events Flow	1. The Visitor accesses the application log in page. 2. The Visitor clicks on the "*Register*" button and then fills in all the mandatory information (company name, VAT code, email and password). 3. T4R creates a new account and sends a confirmation email to the just registered Organiser. 4. The newly registered Organiser can now log into the application using the email and password previously provided, so to have access to the available features.
Exit Condition	The Organiser is registered and able to log into *Track4Run*.
Exception	The Visitor is not able to register because the email or the VAT code are already in use. A registered Organiser is not able to log in because the email or password inserted are wrong or the email has not been confirmed. These exception will be handled by the system sending an error message.

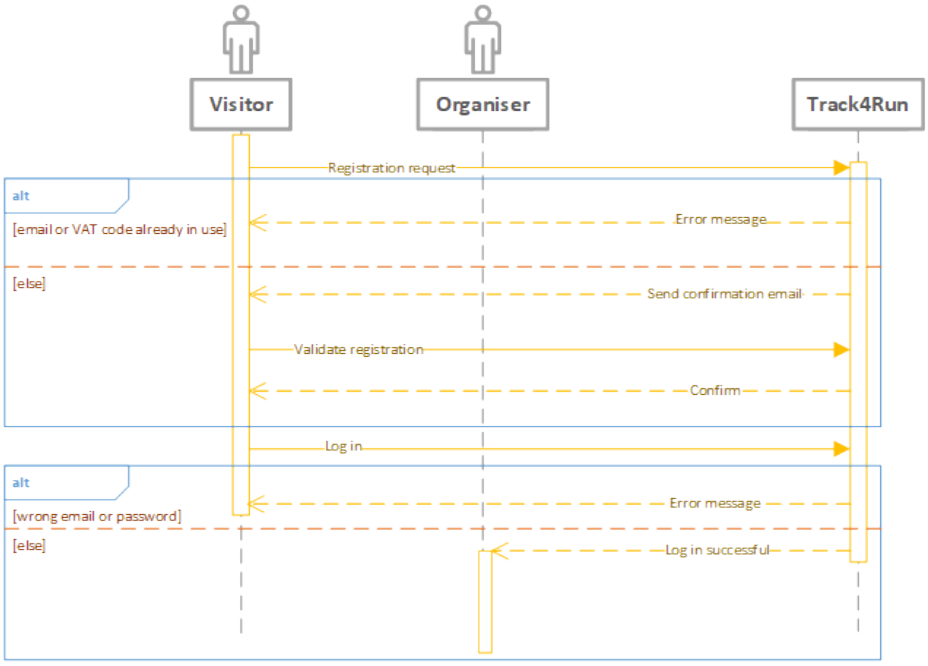

Figure 3.9: Sequence diagram representing the registration of an Organiser to T4R.

Run organisation

The T4R system allows registered organisers to organise a run. They have to specify the time and the place, but it is not possible to add if there is another, already stored, race in the same place and at the same time. Different runs at the same time but different place are allowed.

Actors	• Organiser
Goals	
Entry Condition	Organiser must have an organiser account in T4R system.
Events Flow	1. The Organiser presses on "*New Run*" button. 2. The Organiser specifies date, time, place and the maximum number of participants. 3. The system creates this new run, stores it and sends a notification email to the Organiser.
Exit Condition	A new race has been created.
Exception	If there is another stored race with the same data, time and place, the system handles the exception, showing an error message to the organiser.

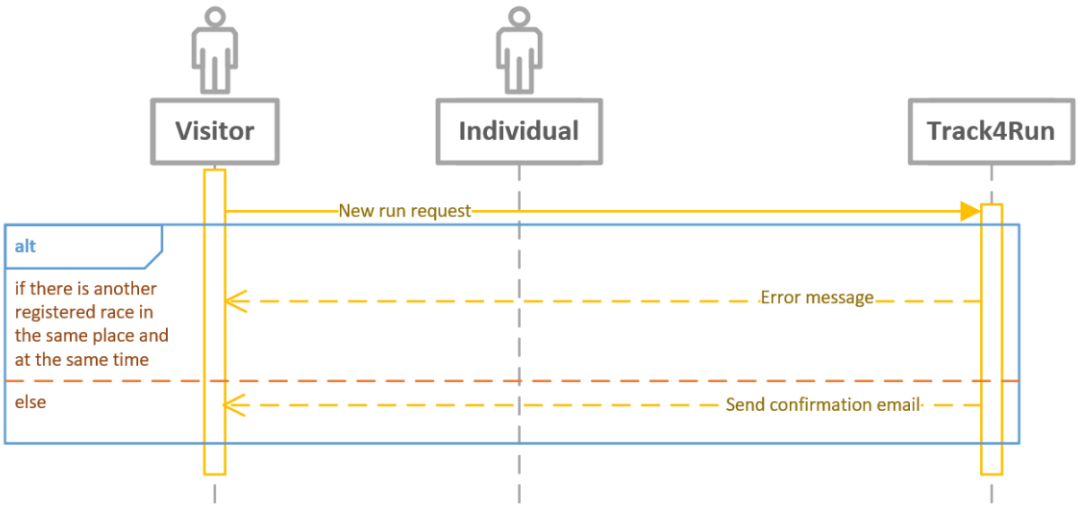

Figure 3.10: Sequence diagram representing the request in order to create a new run.

Enrolment to a run

Any Participant can log into T4H using his D4H (Individual) account. After doing that, he/she will be able to see all available runs and to enrol to them.

The races can be filtered by time and zone, as to simplify the research.

Actors	• Participant
Goals	
Entry Condition	The Participant is logged into T4R, using his D4H Individual account.
Events Flow	1. The participant clicks on the "*Show available races*" button. 2. The Participant looks through the available races, eventually filtering them to ease the search, and selects one of them. 3. The Participant presses on "*Enrol*" button. 4. T4R sends a confirmation mail to the Participant.
Exit Condition	The Participant is enrolled to the selected race.
Exception	If the Participant is already enrolled to another race at the same time, an error message is sent to him/her. If the race has reached the maximum number of subscriptions, an error message is sent to the Participant.

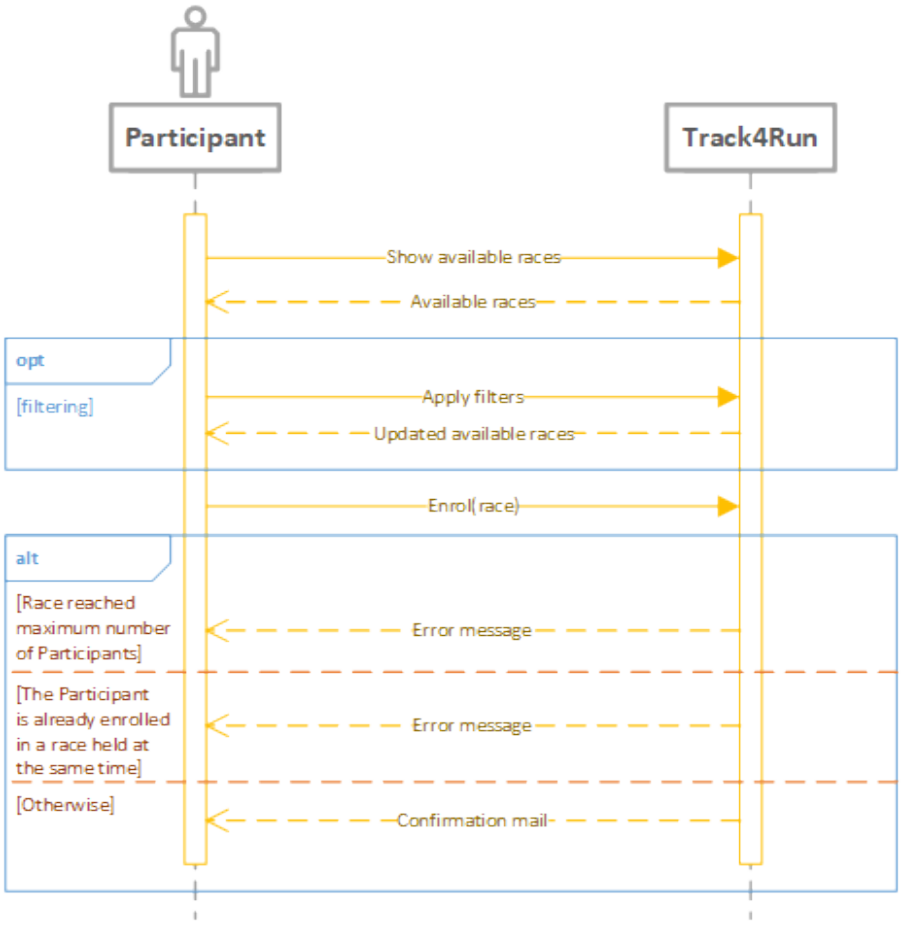

Figure 3.11: Sequence diagram representing the enrolment to a run of a Participant.

Participant tracking

Every Visitor, registered or not, is able to track participants on a map during the race and, if there is more then one run at the same time, they can choose which run to watch. To provide this service, the system interacts with Google Maps.

Actors	• Visitor • Map service
Goals	
Entry Condition	There is no entry condition
Events Flow	1. The Visitor launches the application. 2. They press on the "*Track runners*" button. 3. The system shows the list of current runs. 4. The Visitor selects the run of interest. 5. The system asks for map and runners positions to the map service. 6. The map service sends to the system map and runners location. 7. The system shows the map with all runners current position. 8. If the Visitor follows the run until the end, T4R shows to him the rank.
Exit Condition	Visitors are now able to see runners on a map (and the rank if the stay until the race is over).
Exception	If there are no runs in that moment, the system handles the exception, showing a notification message.

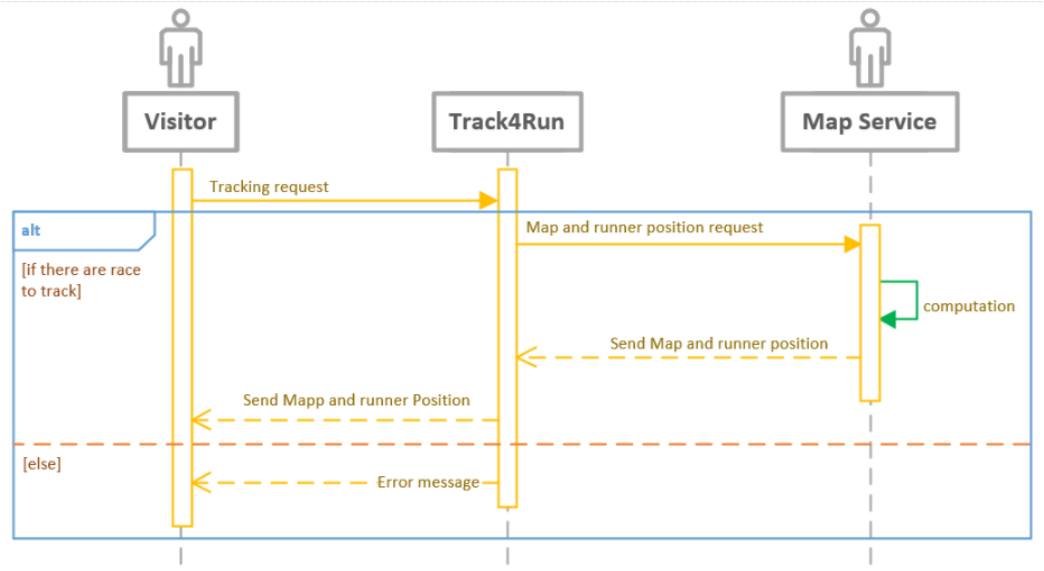

Figure 3.12: Sequence diagram representing the request in order to see participant position on a map during the run.

Rank history

Participants and Organisers are able to see the rank for all past races they have took part in or organised, respectively. Since the procedure to access the rank history is the same for the two actors, only one use case is provided.

Actors	• Participant / Organiser
Goals	
Entry Condition	The Participant has took part in / The Organiser has organised at least one run and is logged in.
Events Flow	1. The Participant / Organiser clicks on the "*Runs history*" tab. 2. The Participant / Organiser selects the run of which he/she wishes to see the rank. 3. T4R shows the rank to the Participant / Organiser.
Exit Condition	The Participant / Organiser is able to see the rank of the selected race he/she has took part in / organised.
Exception	If the Participant has not took part in any run, an error message is sent to him/her. If the Organiser has not organised any run, an error message is sent to him/her.

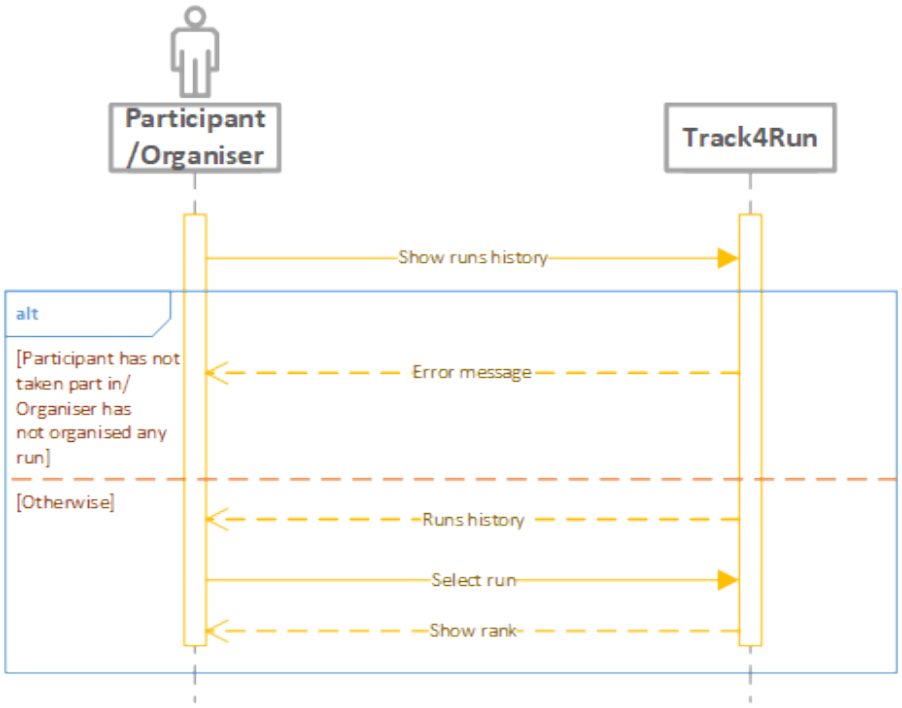

Figure 3.13: Sequence diagram representing request for the rank of a run.

Chapter 4

Formal analysis using Alloy

In this chapter we present how the critical components of each *TrackMe* service have been modelled using Alloy, i.e. a formal language for specifying software models.

4.1 Data4Help Alloy analysis

In Data4Help the most critical parts identified are the requests of the third parties. In both cases, requests for a single user and requests for multiple data, Data4Help must satisfy some requirement in order to protect the privacy of its users. In particular the system must return the health data of the requested user only under his agreement and, in case of request for data of a group of individuals, it has to accept it and forward them only if the amount of people, that match with the filters chosen by the third parties, is greater than 1000, in order to be able to anonymise them.

4.1.1 Signatures

Here we present all the signature composing the model.

```
open util/boolean

    sig User {
        fiscalCode: one FiscalCode,
        uData: one UserData,
        hData: some HealthData
    }{
        no disj hd1, hd2: HealthData |
                hd1.acquisition = hd2.acquisition
    }

    sig FiscalCode{}

    // Gender and age.
    sig UserData{}

    //The health parameters acquired
    sig HealthData{
```

```
    bpm: Int,
    dailySteps: Int,
    acquisition: AcquisitionData
} { bpm ≥ 0 and dailySteps ≥ 0 }

//The information of the acquisition
sig AcquisitionData{
    position: Location,
    time: Time,
    date: Date
}

//The location of gathering
sig Location{}

//The date of gathering
sig Date{}

//The time of sampling
sig Time{}

//The ID is the Fiscal code or the VAT code of the third party.
sig ThirdParty {}

one sig Data4Help{
    users: set User
}

sig SingleRequest{
    thirdParty : ThirdParty,
    fiscalCode: one FiscalCode
}

sig MultipleRequest{
    thirdParty : ThirdParty,
    filterUserData: set UserData,
    filterAcquisitionData: set AcquisitionData
}{ #filterUserData > 0 or #filterAcquisitionData > 0}

//Each Reply is associated to one Request and can be positive or negative
    ↪ , and it is positive will return the data requested, otherwise
    ↪ the field data will be empty
sig SingleReply{
    associatedRequest: one SingleRequest,
    answer: Bool,
    data: lone HealthData
}

sig MultipleReply{
    associatedRequest: one MultipleRequest,
    data: set HealthData
}
```

4.1.2 Facts

In this section are defined all the constraints needed in order to create a consistent model of the requests.

```
//Each fiscal code is related to one user
fact noFiscalCodeWithoutUser{
    no fc: FiscalCode | all u: User | !(fc in u.fiscalCode)
}

//Each UserData is related to one User
fact noUserDataWithoutUser{
```

```
    no ud: UserData | all u: User | !(ud in  u.uData)
}

// Each health data corresponds to one user
fact noHealthDataWithoutAUser{
    no hd : HealthData | all u: User |
            !(hd in u.hData)
}

//Each acquisition data corresponds to a health data
fact noAcquisitionDataWithoutHealthData{
    no aq : AcquisitionData | all hd: HealthData |
            !(aq in hd.acquisition)
}

fact noTimeWithoutAcquisition{
    no t: Time | all aq: AcquisitionData | !(t in aq.time)
}

fact noDateWithoutAcquisition{
    no d: Date | all aq: AcquisitionData | !(d in aq.date)
}

fact noLocationWithoudAcquisition{
    no l: Location | all aq: AcquisitionData | !(l in aq.position)
}

//We assume that all the users considered are registered
fact allUsersRegistered {
    all u: User |
            u in Data4Help.users
}

//The fiscal code is unique
fact uniqueUser{
    all disj u1, u2: User | ( u1.fiscalCode ≠ u2.fiscalCode)
}

// The third party's ID is unique
fact uniqueThirdParty{
    all disj tp1, tp2 : ThirdParty | tp1 ≠ tp2
}

//No Health Data Acquire from the same User at the same time
fact differentAcquisition{
    no hd1, hd2: HealthData | one u: User |
            (hd1 in u.hData) and (hd2 in u.hData) and hd1.acquisition =
                ↪ hd2.acquisition
}


// SINGLE REQUESTS

fact allSingleReqHasAReply{
    #SingleRequest = #SingleReply
}

fact uniqueSingleRequest{
    all disj r1, r2: SingleRequest | r1 ≠ r2
}

//Each single request has a single reply
fact singleReplyUnique{
    all disj srp1,srp2 : SingleReply |
            srp1.associatedRequest ≠ srp2.associatedRequest
}
```

```
// A third party can not request the same fiscal code in different
    ↪ request
fact noMultipleSingleRequestFromTheSameThirdPartyToASingleUser{
    no disj srq1, srq2: SingleRequest |
            srq1.thirdParty = srq2.thirdParty and srq1.fiscalCode = srq2.
                ↪ fiscalCode
}


//This fact certificate that a reply has data of the requested user if
    ↪ the answer of the user is positive.
fact singleRequestAccepted {
    all srp: SingleReply |  one u: User |
            u.fiscalCode = srp.associatedRequest.fiscalCode and (#srp.
                ↪ data > 0 iff srp.answer.isTrue) and(srp.answer.isTrue
                ↪  iff (u.hData in srp.data ))
}


// MULTIPLE REQUESTS

fact allMulReqHasAReply{
    #MultipleRequest = #MultipleReply
}

//All the requests are different
fact uniqueMultipleRequest {
    all disj r1, r2: MultipleRequest | r1 ≠ r2
}

//Each multiple request has a multiple reply
fact multipleReplyUnique{
    all disj mrp1,mrp2 : MultipleReply |
            mrp1.associatedRequest ≠ mrp2.associatedRequest
}


//The fact certificate that a reply for a multiple request contains data
    ↪ just if the number of users with that filter is grather then 1000
    ↪  (to verify the assertion we use 2 instead 1000).
fact multipleRequestAccepted {
    all mrp : MultipleReply | all u: User |
            ( (u.hData in mrp.data and #mrp.data > 0 ) iff
                    #usersWithCorrectFilters[mrp.associatedRequest.
                        ↪ filterUserData, mrp.associatedRequest.
                        ↪ filterAcquisitionData] > 2)
}

//This function gives the subset of user of data4Hepl with the data
    ↪ requested.
fun usersWithCorrectFilters [ userFilter: UserData, acqFilter:
    ↪ AcquisitionData ] : set User {
    {u : Data4Help.users | (u.uData = userFilter) or (u.hData.acquisition
        ↪  = acqFilter)}
}
```

4.1.3 Asserts

At the end these two asserts verify if the model built respects all the constraints to guarantee the requirements before explained.

```
// This assertion check if a single request returns data to the third
    ↪ party if and only if the user's answer is positive
assert singleRequest {
    no r: SingleReply | one u: User |
```

```
            (!r.answer.isTrue) and  r.associatedRequest.fiscalCode = u.
                ↪ fiscalCode  and (u.hData in r.data)
}

// This assertion check if a multiple request returns data if and only if
    ↪  the number of user with a
// specific requirement is grather than 1000 (in the model we used 2 to
    ↪ semplify the check)
assert multipleRequest{
    no r: MultipleReply | some u: User |
            (u.hData in r.data and #r.data>0) and
                    #usersWithCorrectFilters[r.associatedRequest.
                        ↪ filterUserData, r.associatedRequest.
                        ↪ filterAcquisitionData] =< 2
}

// This pred show the model
pred show{}
```

4.2 AutomatedSOS Alloy analysis

AutomatedSOS is a service on top of Data4Help that allows the monitoring of the elderly users, and provides them an ambulance in case of sudden illness. Thus, it is very important that the service sends a request to the ambulance provider when the user's hearth frequency exceed a threshold. The following model has been built in order to guarantee this very critical constrain.

4.2.1 Signatures

Here we present all the signature composing the model.

```
open util/boolean

    sig User {
        fiscalCode: one FiscalCode,
        uData: one UserData,
        hData: some HealthData
    }{
        no disj hd1, hd2: HealthData |
                hd1.acquisition = hd2.acquisition
    }

    sig FiscalCode{}

    // Gender and age.
    sig UserData{
        age : Int
    } {
        age > 0
    }

    //The health parameters acquired
    sig HealthData{
        bpm: Int,
        dailySteps: Int,
        acquisition: AcquisitionData
    } { bpm > 0 and dailySteps > 0 }

    //The information of the acquisition
    sig AcquisitionData{
        position: Location,
```

```
    time: Time,
    date: Date
}

//The location of gathering
sig Location{}

//The date of gathering
sig Date{}

//The time of sampling
sig Time{}

one sig AutomatedSOS{
    registeredUsers : set User
} {
    //All the users registered to AutomatedSOS are elderly people
    //(Therefore as it is expressed in the domain assumption, with an age
        ↪  grather than 65 years but for model semplification we use 5
        ↪ as limit)
    registeredUsers.elderUser
}

sig AmbulanceRequest{
    valueOutOfRange: Bool,
    bpm : Int,
    fiscalCode: lone FiscalCode,
    position: lone Location
}

check ambulanceRequestWhenBpmNotOutOfBound for 5
check noAmbulanceWhenBpmOutOfBound for 5
run show for 5
```

4.2.2 Facts

In this section are defined all the constraints needed in order to create a consistent model of the ambulance requests.

```
//Each fiscal code is related to one user
fact noFiscalCodeWithoutUser{
    no fc: FiscalCode | all u: User | !(fc in u.fiscalCode)
}

//Each UserData is related to one User
fact noUserDataWithoutUser{
    no ud: UserData | all u: User | !(ud in  u.uData)
}

// Each health data corresponds to one user
fact noHealthDataWithoutAUser{
    no hd : HealthData | all u: User |
            !(hd in u.hData)
}

//Each acquisition data corresponds to a health data
fact noAcquisitionDataWithoutHealthData{
    no aq : AcquisitionData | all hd: HealthData |
            !(aq in hd.acquisition)
}

fact noTimeWithoutAcquisition{
    no t: Time | all aq: AcquisitionData | !(t in aq.time)
}
```

```
fact noDateWithoutAcquisition{
    no d: Date | all aq: AcquisitionData | !(d in aq.date)
}

fact noLocationWithoudAcquisition{
    no l: Location | all aq: AcquisitionData | !(l in aq.position)
}

//The fiscal code is unique
fact uniqueUser{
    all disj u1, u2: User | ( u1.fiscalCode ≠ u2.fiscalCode)
}

//No Health Data Acquire from the same User at the same time
fact differentAcquisition{
    no hd1, hd2: HealthData | one u: User |
            (hd1 in u.hData) and (hd2 in u.hData) and hd1.acquisition =
                ↪ hd2.acquisition
}


//No request for the same event to the same user
fact allRequestAreUnique{
    all disj r1, r2: AmbulanceRequest |
            r1.valueOutOfRange ≠ r2.valueOutOfRange or r1.bpm ≠ r2.bpm or
                ↪  r1.fiscalCode ≠ r2.fiscalCode or r1.position ≠ r2.
                ↪ position
}


// The bpm in the request refer to the bpm of a user registered to ASOS
    ↪ so with a age grather than 65 (6 in the model)
fact requestHasTheBpmOfTheUser{
    all r: AmbulanceRequest |
            (r.bpm in AutomatedSOS.registeredUsers.hData.bpm) and (r.
                ↪ fiscalCode in AutomatedSOS.registeredUsers.fiscalCode
                ↪ )
}

// In the model all the elderly people considered are registered.
fact allElderlyUserRegistered{
    all u: User |
            u.elderUser iff ( u in AutomatedSOS.registeredUsers)
}

//Ambulance request is attended only if the bpm is out of range
fact ambulanceOnlyIfBpmOutOfRange{
    all r: AmbulanceRequest |
            r.valueOutOfRange.isTrue iff r.bpm.outOfRange
}

fact userConnectedIfThereIsARequest{
    all r :AmbulanceRequest |
            r.valueOutOfRange.isTrue iff ((r.bpm in AutomatedSOS.
                ↪ registeredUsers.hData.bpm) and (r.fiscalCode in
                ↪ AutomatedSOS.registeredUsers.fiscalCode) )
}

// Location and fiscalCode of the old user are in the request only if his
    ↪  bpm are out of bound
fact noLocAndFCWhenBpmOk{
    all r: AmbulanceRequest |
            (#r.position ≠ 0 and #r.fiscalCode ≠ 0) iff r.valueOutOfRange
                ↪ .isTrue
}

fact allUserWithBpmOutOfRangeHasARequest{
    all u: User, r :AmbulanceRequest |
```

```
            u.hData.bpm.outOfRange implies (u.fiscalCode in r.fiscalCode)
}
```

4.2.3 Asserts

At the end this two asserts verify if the model built respects all the constraints to guarantee the requirements before explained.

```
assert ambulanceRequestWhenBpmNotOutOfBound{
    no request: AmbulanceRequest |
            (request.valueOutOfRange.isTrue and !request.bpm.outOfRange)
}

assert noAmbulanceWhenBpmOutOfBound{
    no request: AmbulanceRequest |
            (!request.valueOutOfRange.isTrue and request.bpm.outOfRange)
}

pred outOfRange [bpm: Int]{
    //The limit under the domain assumption are 130 and 40 but for
        ↪ semplifing the model we choose smaller value, 4 and 2
    bpm > 4 or bpm < 2
}

//This pred return true if the user is old (We consider old people, who
    ↪ has an age greater or equal than 65,
//but for semplification in the model we assume 5 as the limit of old
    ↪ people age
pred elderUser [ u: User ] {
    u.uData.age > 5
}

pred show{}
```

4.3 Track4Run Alloy analysis

In the Track4Run service is very important that the application allows the organizer to plan a race and define the path, users of Data4Help to enroll to the race and spectator, not registered people, to track the participant during the run. The below model is built to verify all these properties and verify one of them by the assertion.

4.3.1 Signatures

Here we present all the signature composing the model.

```
// This service is under Data4Help so all the Track4Run users are user
    ↪ also of Data4Help
one sig Track4Run{
    users: some Runner
}

sig Runner{
    id: Int,
    // For semplification we consider just the partecipation to one race
    race: Int,
    position: some Position
}
```

```
sig Visitor{
    // The followed race (we assume for the model that a visitor can
        ↪ follow just a race
    race: Race,
    participants: some Runner
} {
    race.id = participants.race
}

sig Organizer{
    // We assume that an organizer could organize just one race, to
        ↪ semplify the model.
    race: Race
}

sig Race{
    id: Int,
    path: Path,
    date: Date,
    time: Time,
    place: Place,
    participants: some Runner
}{
    participants.race = id and #participants > 1
}

sig Position{
    location: Location,
    time: Time
}

// A path of a race
sig Path{}

//The date of the race
sig Date{}

//The time of the race
sig Time{}

//The place of the race
sig Place{}

// The location of a partecipant
sig Location{}
```

4.3.2 Facts

In this section are defined all the constraints needed in order to create a consistent model of Track4Run.

```
// All Runners are registered to Track4Run
fact allRunnerRegisteredToT4R{
    all r: Runner |
            r in Track4Run.users
}

// All runners are different
fact allRunnersAreDifferent{
    all disj r1, r2: Runner |
            r1 ≠ r2
}

// Each runner can not partecipate to different races at the same time
```

```
fact noRunnerAtDifferentRaceAtTheSameTime{
    all disj rc1, rc2: Race | no rn : Runner |
            rc1.date = rc2.date and rc1.time = rc2.time and (rn in rc1.
                ↪ participants) and (rn in rc2.participants)
}

//No time without race or postion
fact noTimeWithoutPositionOrRace{
    no t: Time | all p: Position, r: Race |
            !(t in p.time) or !(t in r.time)
}

//All the date related to a race
fact noDateWithoutARace{
    no d: Date | all r: Race|
            !(d in r.date)
}

//All the race have an organizer
fact allOrganizerAssociatedToAtMostOneRace{
    no r: Race | all o: Organizer |
            !(r in o.race)
}

// All the runners partecipate to a race
fact noRunnersWithoutARace{
    no r: Runner | all rc: Race |
            !(r in rc.participants)
}

// All the visitor watch a race
fact noVIsitorWithoutARace{
    no v: Visitor | all r: Race |
            !(v.race in r)
}

// No location not related to any position
fact noLocWithoutPositionOrRace{
    no l: Location | all p: Position |
            !(l in p.location)
}

//No position not related to any runner
fact noPosWithoutRunner{
    no p: Position | all r: Runner |
            !(p in r.position)
}

//Position of different runners at different race can be the same
fact noSamePositionInDifferentRaces{
    no disj r1,r2: Runner |
            r1.race ≠ r2.race and ((r1.position in r2.position) or (r2.
                ↪ position in r1.position))
}

// All paths are related to one race
fact noPathWithoutRace{
    no p: Path | all r: Race |
            !(p in r.path)
}

// Each race has a unique ID
fact allRaceDifferentID{
    all disj r1,r2: Race |
            r1.id ≠ r2.id
}

// All the race has one organizer
```

```
fact allRaceOneOrganizer{
    #Race = #Organizer
}

// A spectator can see the position of every partecipants
fact aVisitorLookToEveryPartecipant{
    all v: Visitor |
            #v.participants = #v.race.participants
}
```

4.3.3 Asserts

At the end this assert verify if the model built respects the most critical constraint. The others can be visualize clearly on the graph in the Result Section.

```
// This assertion check if a visitor looking to a race track only the
    ↪ partecipants to that race.
assert noVisitorLookingToParticipantsOfDifferentRace{
    no v:Visitor |
            !(v.participants in v.race.participants)
}

// This predicates allow to visualize the model
pred show{}
```

4.4 Results

This section introduces all the results provide by Alloy through the predicates above. 'Check' allows the analysis of the assertion and the verification of them, then 'show' display the model and verify its consistency.

4.4.1 Data4Help results

These predicates for Data4Help verify if a single request return the data only if the user's answer is positive, and if, in case of multiple request, the data are provided only if the number of people corresponding to some specific filter is greater or equal than 1000 (3 in the model for semplification).

```
check singleRequest for 10
check multipleRequest for 10 but exactly 8 User
run show for 10 but exactly 8 User
```

Executing "Check singleRequest for 10"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
58783 vars. 2790 primary vars. 137702 clauses. 187ms.
No counterexample found. Assertion may be valid. 99ms.

Executing "Check multipleRequest for 10 but exactly 8 User"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
53998 vars. 2666 primary vars. 124431 clauses. 121ms.
No counterexample found. Assertion may be valid. 7794ms.

Executing "Run show for 10 but exactly 8 User"
Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
52839 vars. 2648 primary vars. 120606 clauses. 110ms.
Instance found. Predicate is consistent. 172ms.

3 commands were executed. The results are:
#1: No counterexample found. singleRequest may be valid.
#2: No counterexample found. multipleRequest may be valid.
#3: **Instance found.** show is consistent.

Figure 4.1:

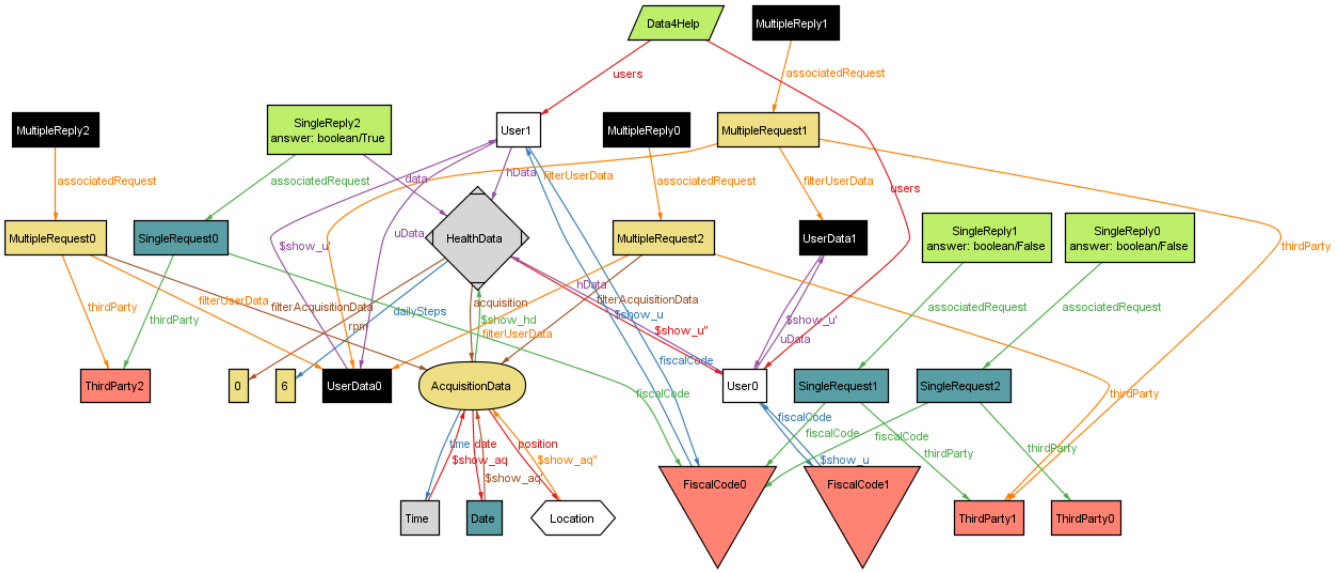

Figure 4.2:

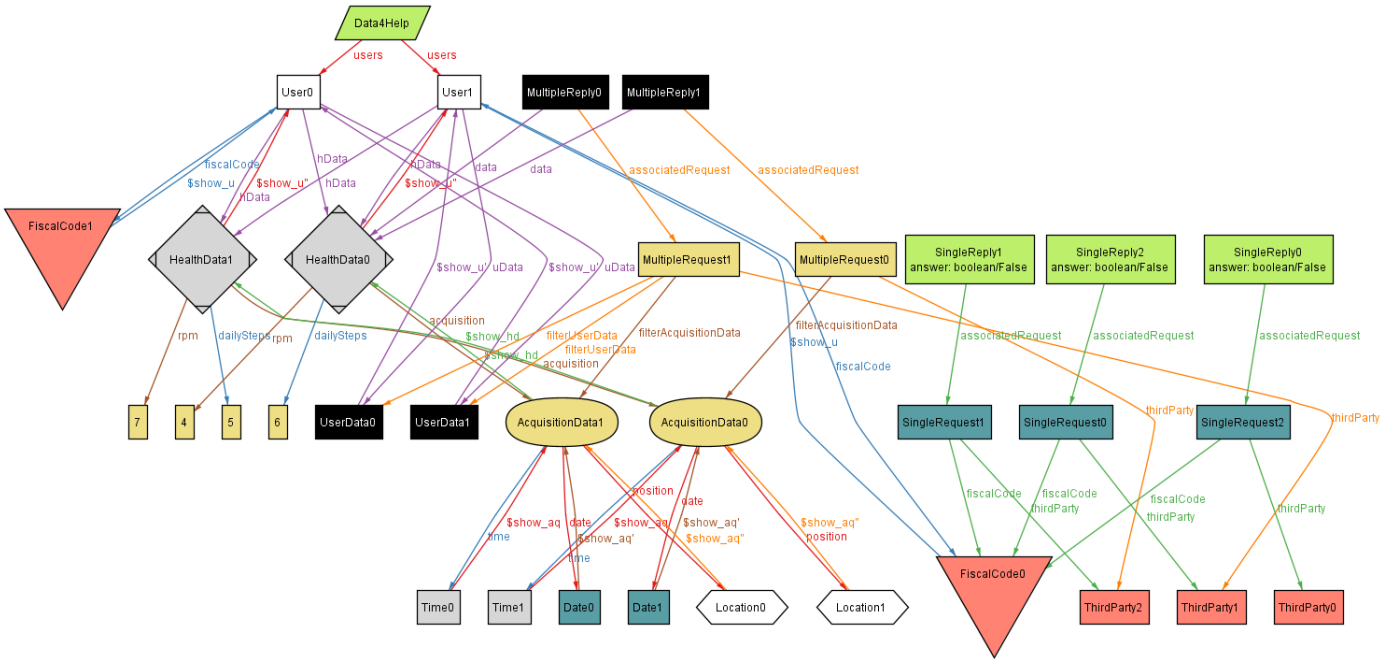

Figure 4.3:

4.4.2 AutomatedSOS results

On the other hand, the predicates above verify if the model of AutomatedSOS respect the constraint. Therefore, it proves the presence of an ambulance request if and only if a registered user has an hearth frequency too high or too low in relation to some thresholds.

```
check ambulanceRequestWhenBpmNotOutOfBound for 10
check noAmbulanceWhenBpmOutOfBound for 10
run show for 10
```

```
Executing "Check ambulanceRequestWhenBpmNotOutOfBound for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  48900 vars. 2370 primary vars. 109175 clauses. 320ms.
  No counterexample found. Assertion may be valid. 156ms.

Executing "Check noAmbulanceWhenBpmOutOfBound for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  48898 vars. 2370 primary vars. 109136 clauses. 156ms.
  No counterexample found. Assertion may be valid. 94ms.

Executing "Run show for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  48470 vars. 2360 primary vars. 107816 clauses. 125ms.
  Instance found. Predicate is consistent. 250ms.

3 commands were executed. The results are:
  #1: No counterexample found. ambulanceRequestWhenBpmNotOutOfBound may be valid.
  #2: No counterexample found. noAmbulanceWhenBpmOutOfBound may be valid.
  #3: Instance found. show is consistent.
```

Figure 4.4:

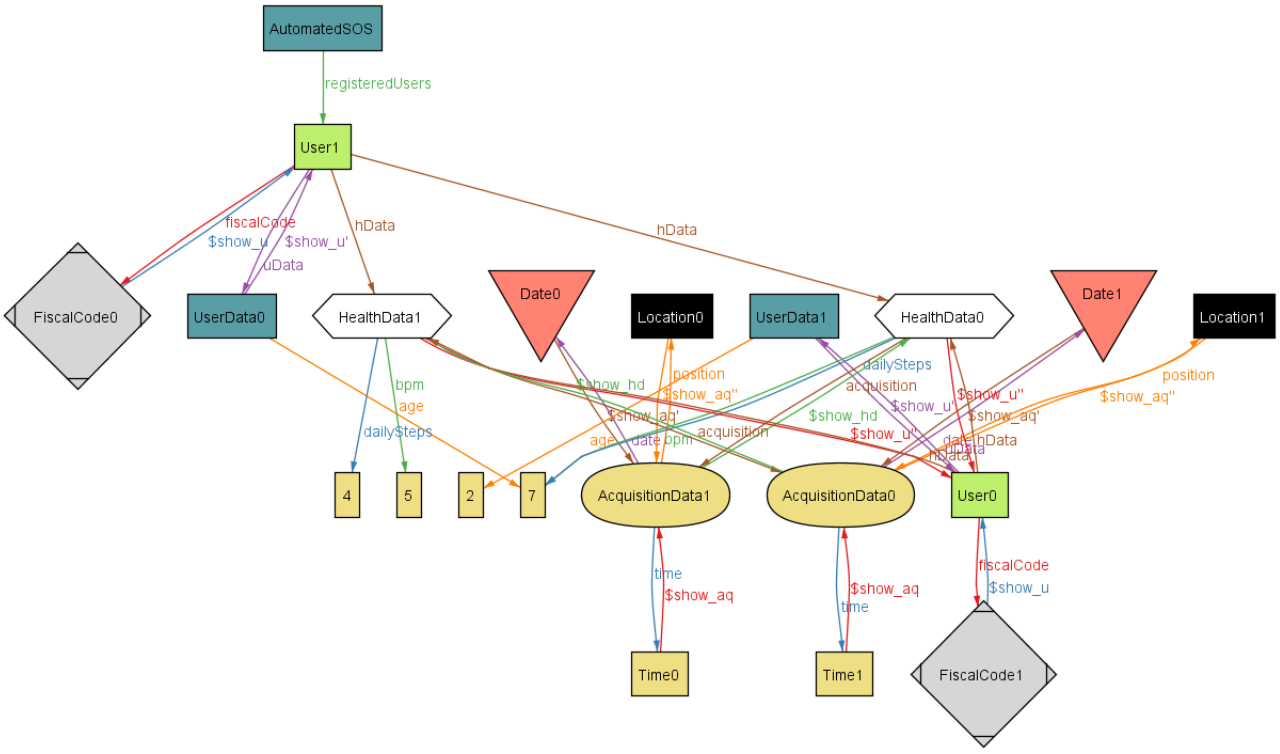

Figure 4.5:

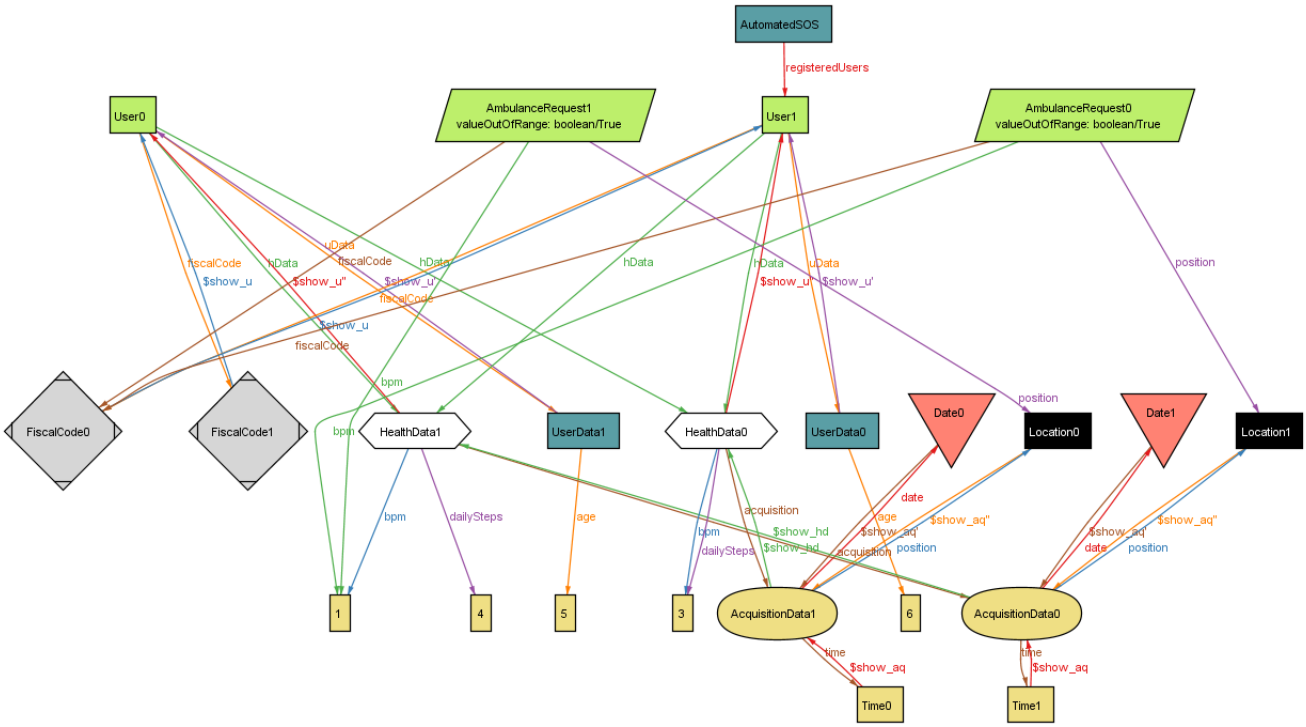

Figure 4.6:

4.4.3 Track4Run results

In the last service, Track4Run, the predicate verifies that a visitor can track only the participants enrolled to the race followed by him.

```
check noVisitorLookingToParticipantsOfDifferentRace for 10
run show for 10
```

```
Executing "Check noVisitorLookingToParticipantsOfDifferentRace for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  44868 vars. 2600 primary vars. 79955 clauses. 99ms.
  No counterexample found. Assertion may be valid. 10952ms.

Executing "Run show for 10"
  Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
  44448 vars. 2590 primary vars. 79190 clauses. 107ms.
  Instance found. Predicate is consistent. 471ms.

2 commands were executed. The results are:
  #1: No counterexample found. noVisitorLookingToParticipantsOfDifferentRace may be valid.
  #2: Instance found. show is consistent.
```

Figure 4.7:

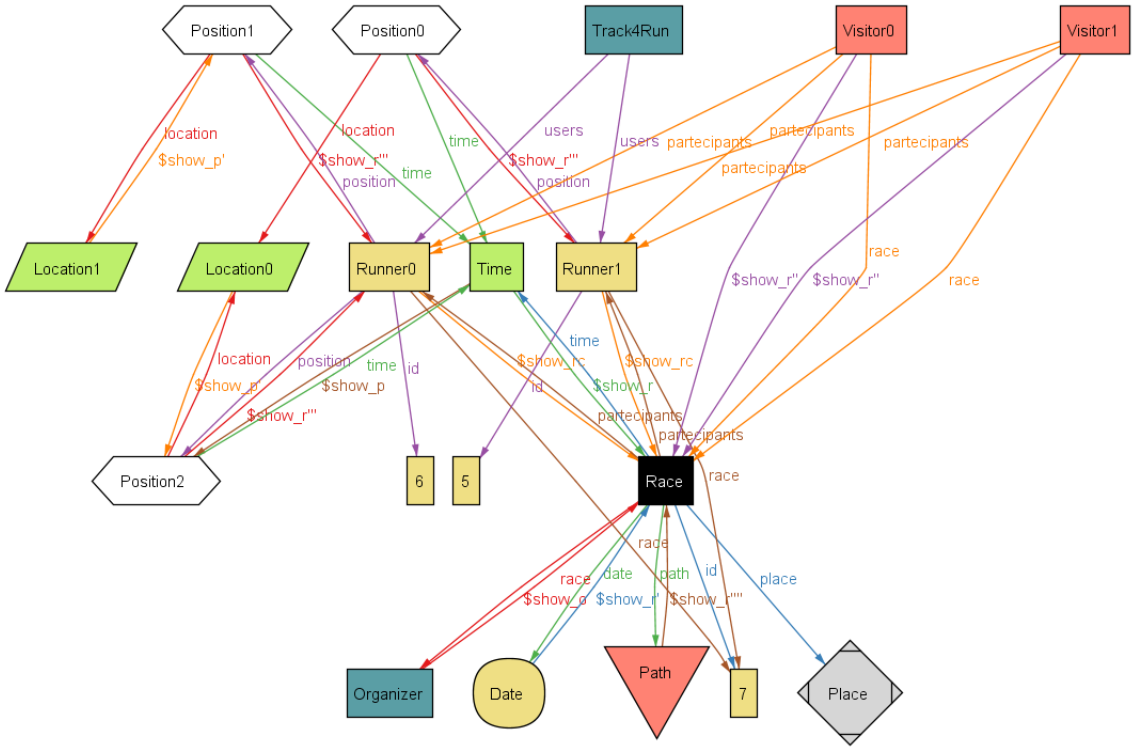

Figure 4.8:

The following pictures shows the result of each predicates and a model represents through a simplified graph.

Chapter 5

Effort Spent

5.1 Alessia Buccoliero

DATE	Number of hours	TOPIC
15/10/2018	2	Work Organisation
16/10/2018	3	General discussion and introduction writing
21/10/2018	3,5	Goals and domain assumptions sketch
23/10/2018	4,5	Goals and domain assumptions revision and requirements analysis discussion
24/10/2018	1,5	Goals, domain assumptions and requirements review
26/10/2018	3,5	Class diagram and use cases definition
27/10/2018	3	Chapter 2: Overall description
28/10/2018	1,5	Chapter 2: Overall description
30/10/2018	1	Use Case
1/11/2018	2	Chapter 3 : Use case (table)
2/11/2018	2	Chapter 3 : Use case (table)
5/11/2018	1	Chapter 3 : Sequence diagrams

Table 5.1: Alessia Buccoliero's working hours

5.2 Emilio Corvino

DATE	Number of hours	TOPIC
15/10/2018	2	Work Organisation
16/10/2018	3	General discussion and introduction writing
21/10/2018	3,5	Goals and domain assumptions sketch
23/10/2018	4,5	Goals and domain assumptions revision and requirements analysis discussion
24/10/2018	1,5	Goals, domain assumptions and requirements review
26/10/2018	3,5	Class diagram and use cases definition
27/10/2018	1,5	Chapter 1: Introduction (Purpose)
28/10/2018	1,5	Chapter 1: Introduction (Scope)
29/10/2018	2,5	Chapter 1: Introduction
30/10/2018	1	Chapter 1: Introduction, finalisation and refinement
31/10/2018	0,5	Chapter 2: Assumptions, dependencies and constraints
02/11/2018	1,5	Chapter 2 revision and Chapter 3 use cases writing
03/11/2018	2	Chapter 3 use cases writing
04/11/2018	2	Chapter 3 use cases finalisation
07/11/2018	1	Chapter 3 sequence and activity diagrams composition
08/11/2018	1	Chapter 3 sequence and activity diagrams finalisation

Table 5.2: Emilio Corvino's working hours

5.3 Gianluca Drappo

DATE	Number of hours	TOPIC
15/10/2018	2	Work Organisation
16/10/2018	3	General discussion and introduction writing
21/10/2018	3,5	Goals and domain assumption sketch
23/10/2018	4,5	Goals and domain assumptions revision and requirement analysis discussion
24/10/2018	1,5	Goals, domain assumptions and requirements review
26/10/2018	3,5	Class diagram and use cases definition
28/10/2018	2	D4H Alloy design
29/10/2018	1,5	Use case relative to G1.2 definition and D4H Alloy design
30/10/2018	4	D4H Alloy design and implementation
02/11/2018	5	D4H Alloy modelling
03/11/2018	4	D4H Alloy debugging
04/11/2018	6	D4H Alloy debugging and design ASOS
05/11/2018	2,5	ASOS Alloy implementation and debugging
06/11/2018	3	T4R Alloy design and implementation
07/11/2018	2,5	T4R Alloy implementation, debugging and document writing

Table 5.3: Gianluca Drappo's working hours