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 1.1
11/11/2018

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 provides 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 specific needs, 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 (or aggregated) 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).
- **Frequency and Granularity:** these terms are related to the subscriptions. The frequency represents the period of time that has to pass between updates, while the granularity represents how the data is aggregated (e.g. average bpm per hour/day/week; number of steps per hour/-day/week). Both of these parameters can be set by the Third-party after they check the box for the subscription.
- **Enterprise application:** this expression refers to the D4H application available to Third-parties on the App Store and Play Store. When the *Enterprise* adjective is not present, the document refers to the application downloaded by Individuals.
- **Organiser edition:** this expression refers to the Organisers' version of T4R application, available on the App Store and Play Store. When T4R is referenced as "application", without specifying anything, it is implied that the Participants' version of the application is being addressed. Note that both the editions have the possibility to track Participants without being registered.

1.3.2 Acronyms

- **API:** Application Programming Interface.
- **RASD:** Requirement Analysis and Specification Document.
- **UML:** Unified Modeling Language.
- **HTTPS:** HyperText Transfer Protocol over Secure Socket

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
1.1	Typography corrections, separation of D4H and T4R clients, mock-ups and diagrams update (figures 3.18, 3.22, 3.27)

Table 1.1: Revision history table

1.5 Reference Documents and Software Tools

1.5.1 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

1.5.2 Software Tools

1. *Overleaf*, an online LaTeX editor;
2. *starUML*, a UML tool to create class diagrams;
3. *Visio*, a diagramming and vector graphics application by Microsoft;
4. *Mockplus*, an all-in-one prototyping tool to create mobile app prototypes;
5. *Alloy Analyzer*, a solver that takes the constraints of a model and finds structures that satisfy them.

1.6 Document Structure

The structure of this RASD document 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

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 for each service obtained by running such model.

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

The Bibliography is provided at the end of the document. All the sources of information exploited are listed there.

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 smart-watches or fitness band 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. Also Individual can cancel subscriptions.

- 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 analyse data gathered by their *Data4Help* account. In case of critical health values, the system sends their 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 them available in the future to both Participants and Organisers.

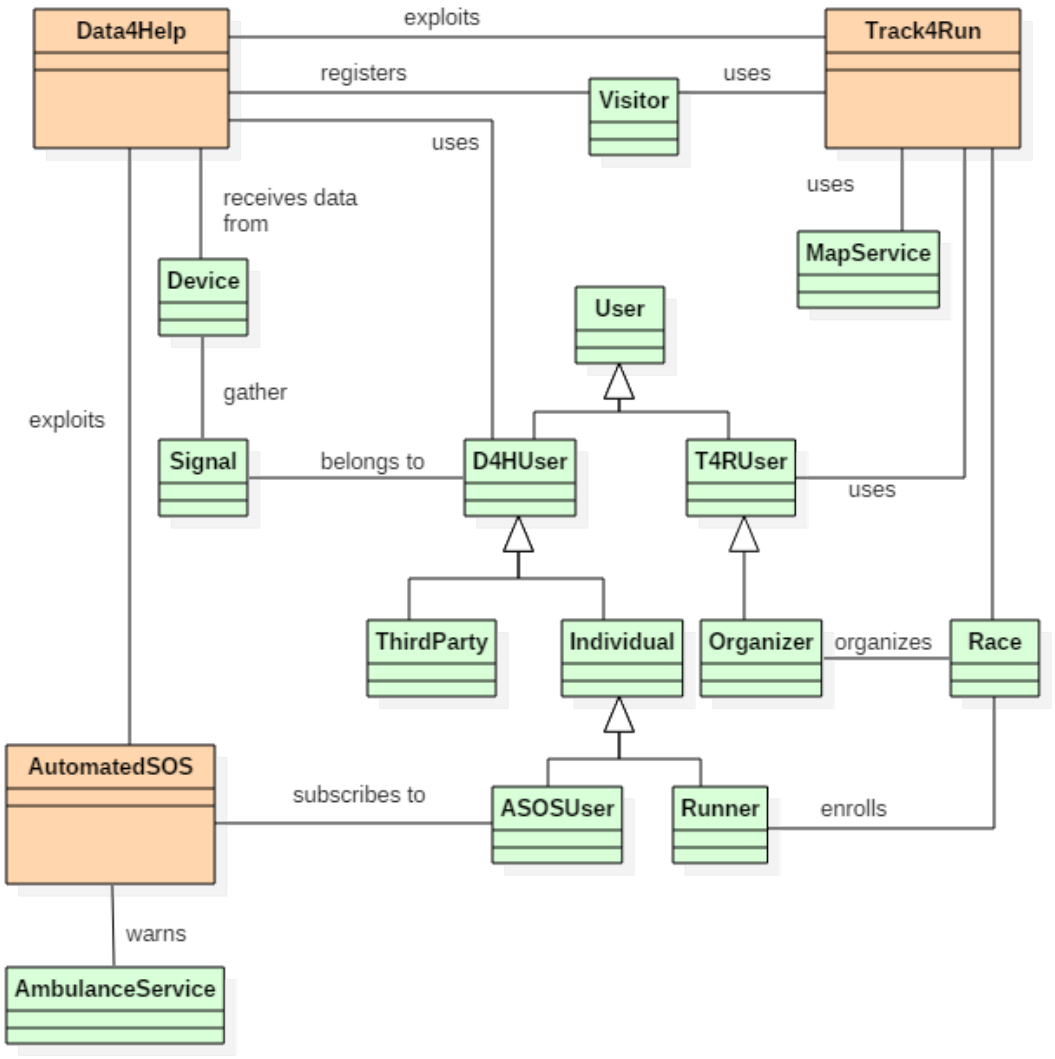


Figure 2.1: Class diagram of the entire system

2.2 Product functions

In this section there is an analysis of functional requirements (grouped by the three 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 system allows Third-parties to request data of Individuals and the latter to accept or refuse them.

[R1.4.1] If the Individual accepts, the system sends to the Third-party all the data gathered from the specified user.

[R1.4.2] If the Individual refuses or he/she is not registered, the system sends to the Third-party an error message.

[R1.5] 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.

[R1.5.1] If the request is approved, the saved data is provided to the Third-party.

[R1.5.2] If the request is not approved, an error message is sent to the Third-party.

[R1.5.3] In order to anonimize data, the application only sends information related to the health status of individuals (e.g. bpm, step, genre, age).

[R1.6] The application allows Third-parties to subscribe to certain data.

[R1.6.1] 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.

[R1.7] The application allows Individuals withdrawals of consent for accessing their data and Third-party unsubscriptions.

- [R1.7.1] 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.
- [R1.7.2] 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.

[R1.8] The system allows Third-parties to access to their request history and results.

[R1.9] 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 beyond the threshold.

[R2.2] The system gives the possibility to subscribe to the service upon request, only after checking the Individual 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. They should also be able to apply filters to ease the research (e.g.: place, time, date range).

[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 Users 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 location, 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 at the same place and 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 with the application 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, along with the date of the Elderly person in need of help.
- *Map service*: The applications need to interact with the map service, *Google Maps*, in order to let Individuals see gathered position (D4H) and Visitors and other users see on a map the position of all runners during the run (T4R).
- *Organisers*: They are companies that are registered to Track4Run service. The system provides them with a sign in interface and, after log in, lets them organise runs. They don't need to be registered to Data4Help.
- *Runners or Participants*: They are Individuals, the application allows them 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-parties are assumed to be companies that want to gather data, therefore they have a VAT code.

[D1.6] Third-parties know the fiscal code of the Individual they are looking for.

AutomatedSOS

[D2.1] An ambulance service capable of handling requests exists. It is also capable of receiving the personal data of the Elderly person and his location.

[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] 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 place where the run is held 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

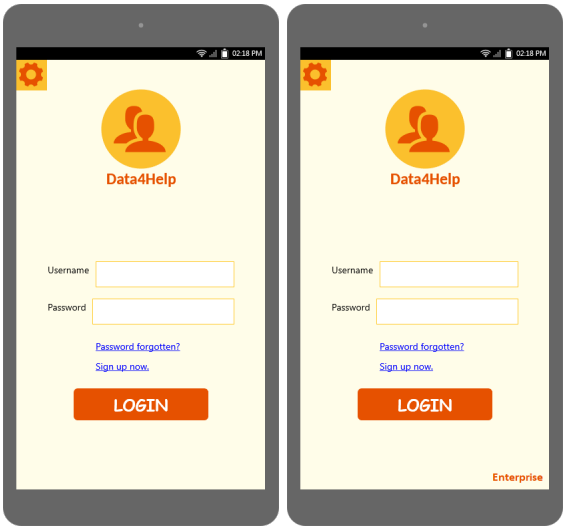
In this section we are going to better explain the relation between our software and the main actors of System.

3.1.1 User Interfaces

Data4Help interfaces

In the pictures below there are mockups for the D4H applications.

Picture 3.1 represents screens from the login pages of D4H. The registered users (both Individuals **Picture 3.1(a)** and Third-parties **Picture 3.1(b)**) can log into their D4H account writing in their username and password. If not yet registered, the user can access the *Registration pages* trough the link *Sign up now*.

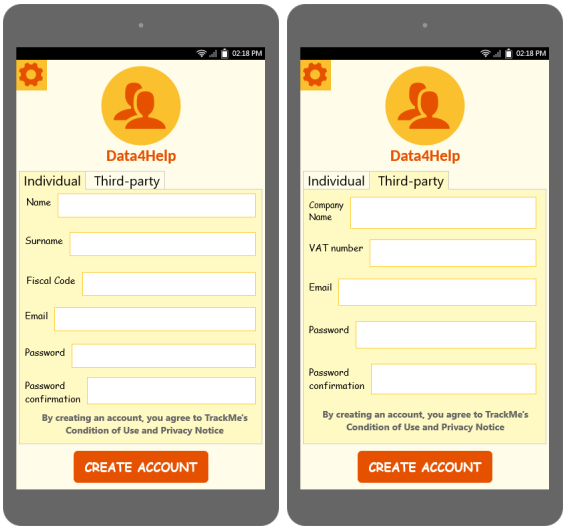


(a) D4H Individual Login page

(b) D4H Third-party Login page

Figure 3.1: D4H Login pages

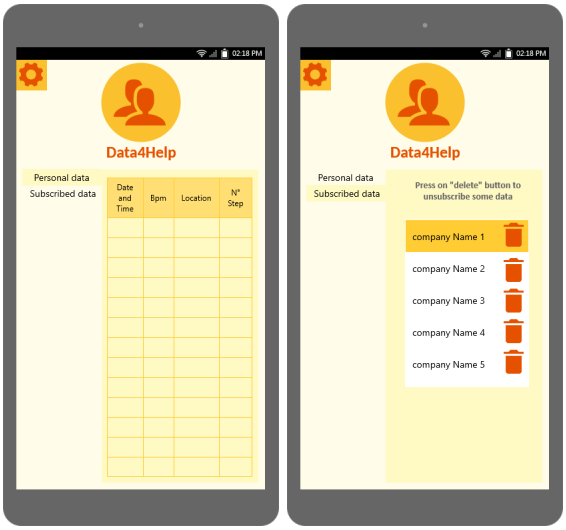
In **picture 3.2** there are two registration forms in order to create either an Individual account (**picture 3.2(a)**) or a Third-party account (**picture 3.2(b)**).



(a) Individual registration page (b) Third-party registration page

Figure 3.2: D4H Registration pages

In **picture 3.3** there are screens from an Individual account. In the first image (**picture 3.3(a)**) the registered user can look at his gathered data, collected in a table; in the *Location* columns there is a link to open the position on a map, retrieved from *Google Maps*. In the second image (**picture 3.3(b)**), instead, the application lets the Individual see a list of all Third-parties that are subscribed to his not-anonymized data and to delete the subscription at any moment.

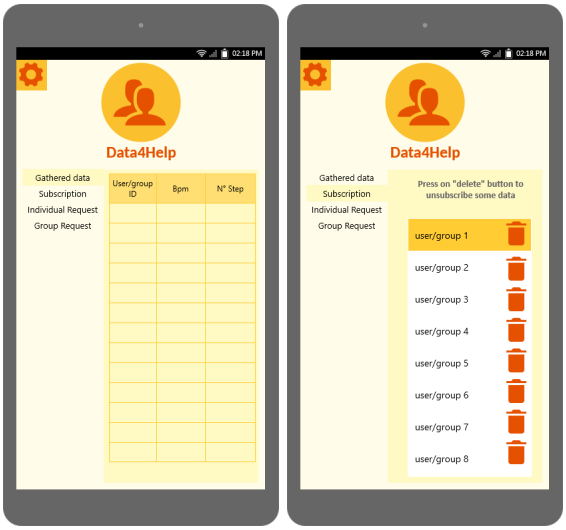


(a) Personal data (b) Subscribed data

Figure 3.3: D4H Individual account pages

In **picture 3.4** there are mockups for the Third-party account. It is composed of four pages:

1. the first (**picture 3.4(a)**) lets the Third-party access data of past requests;
2. in the second (**picture 3.4(b)**) there is a list of all Individuals or groups to which the Third-party is subscribed and it can unsubscribe simply by clicking on the "delete" button;
3. the third and the fourth are the two forms to fill in order to make new data requests (and subscriptions, selecting the check-box *Subscription*). In the Group request form (**picture 3.4(d)**) there are all the possible parameters according to which the request could be done. To compute an Individual request (**picture 3.4(c)**), instead, only the fiscal code is required.



(a) Gathered data (b) Subscription

(c) Individual request (d) Group request

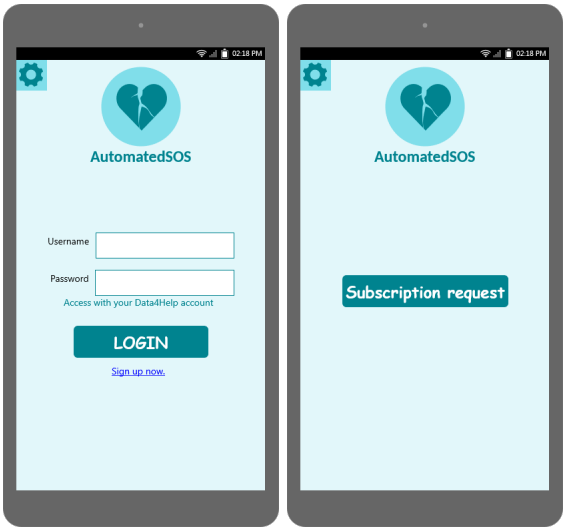
Figure 3.4: D4H Third party account pages

ASOS

In the pictures below there are mockups for ASOS. Since it is intended for Elderly people, the interface is kept as simple as possible: the functions of the application are done in background.

AutomatedSOS service provides a sign in interface (**picture 3.5(a)**) and,

after successful log in with a D4H account, an interface to make the subscription request (**picture 3.5(b)**).



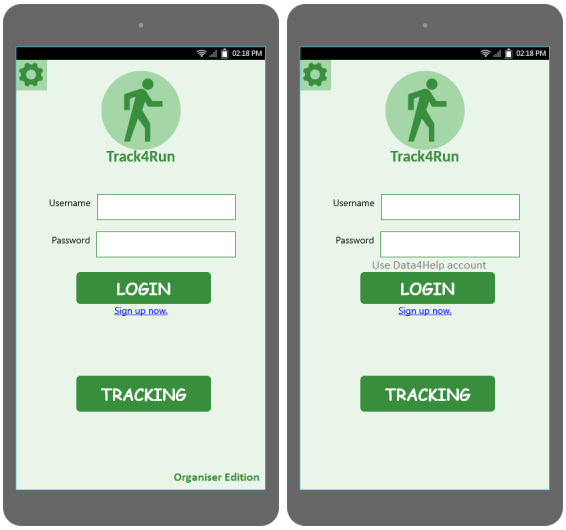
(a) Login page (b) Subscription request

Figure 3.5: D4H Individual account pages

T4R

In the pictures below there are mock-ups for T4R.

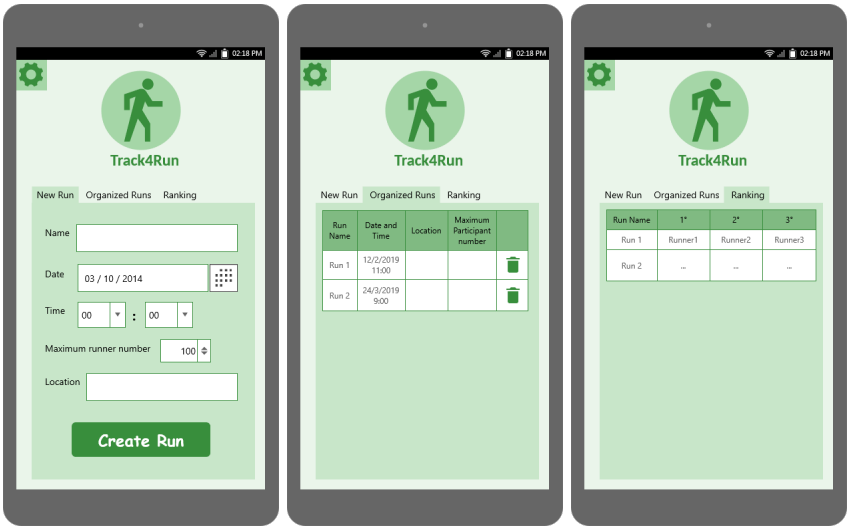
The first two images (**picture 3.6**) are a possible design of the T4R sign in page. The application gives the possibility to access and register as *Organiser* (**picture 3.6(a)**), to access as *Runner* (**picture 3.6(b)**) or to use the application as *Visitor* clicking on the "TRACKING" button (both picture 3.6(a) and 3.6(b)). Participants can log in through their D4H account.



(a) Organiser log in (b) Participant log in

Figure 3.6: T4R Log in pages

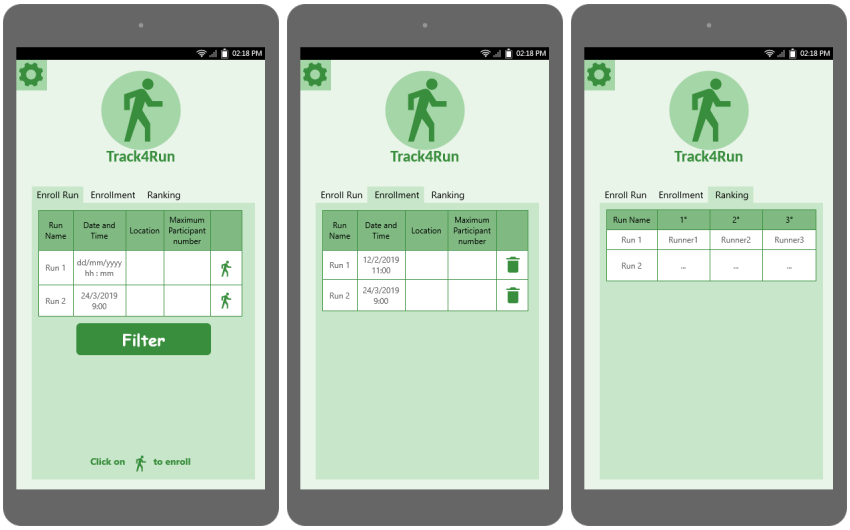
The following three screens are a representation of how an Organiser account in T4R looks like (in the Organiser edition). This Application provides a page to create a new run (**picture 3.7(a)**); a page with already created runs (**picture 3.7(b)**) and a page to see the rankings of past races (**picture 3.7(c)**).



(a) New Run Creation (b) Organised Run (c) Ranking

Figure 3.7: T4R Organiser account pages

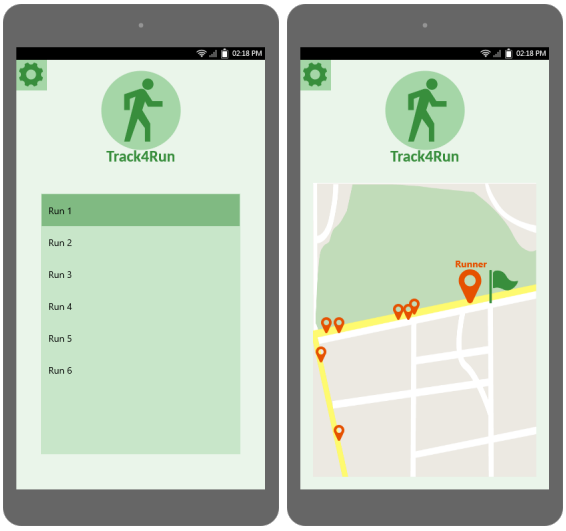
Picture 3.8 below shows the design of the Runner's account page. The application gives the possibility to enroll to a run (**picture 3.8(a)**) (filtering available race by clicking on *Filter* button), to see all runs where the runner is enrolled (**picture 3.8(b)**) and, as for the organizer account, to see the rankings of past races to which the user took part(**picture 3.8(c)**).



(a) New enrollment (b) Enrollment (c) Ranking

Figure 3.8: T4R runner account pages

The following screens (**picture 3.9**) show the pages accessed by a Visitor; they can only choose between current runs (**picture 3.9(a)**) and track runners on a map (**picture 3.9(b)**)



(a) Choosing run to track (b) Tracking

Figure 3.9: T4R visitor pages

3.1.2 Software Interfaces

In order to provide all services, our applications need to interact with these external software:

- *Map service* (Google Maps)
- *Ambulance service*

All these software will be accessed through specific APIs.

3.1.3 Communication Interfaces

D4H, *ASOS* and *T4R* will be developed mainly with the idea of being *Android* and *iOS* applications. In both cases it is mandatory to have a Bluetooth connection in order to receive data from external device (smart-watch or fitness band) and a working Internet connection, to send data to the main Application Server (to compute requests, to create run, to enroll a run).

3.2 Functional Requirements

This section is split into 3 parts, one for each application. In each of them there is a scenario describing a realistic situation that may occur in the world and how the actors interact with the software. Every scenario is then generalised in an use case diagram, several use cases, sequence and activity diagrams from the UML framework.

3.2.1 Data4Help

Consider the following scenario:

RestAssured is an insurance company that wants to offer a personalised service to its customers. In order to do so, it registers to *Data4Help* as a Third-party through the Enterprise Android application, providing the needed data. Luigi is a new customer of RestAssured. The company has a special offer dedicated to the clients that accept to be tracked through *Data4Help*, so he downloads the application on his device and registers to the service as an Individual, also associating his smart-watch. RestAssured then looks for him in the application inserting his fiscal code and sends him a request for the access to his data, including a subscription request, with updates wanted every week. Luigi accepts the request, so *Data4Help* starts sending his data to RestAssured.

After a couple of months, however, Luigi no longer wishes to benefit from RestAssured's special offer, because he is concerned about his privacy and finds out through *Data4Help* that he is giving a lot of data to RestAssured. Therefore, he accesses the subscription page on *Data4Help* and cancels the company's subscription. RestAssured receives the email that notifies Luigi's refusal of further provision of data and cancels his offer.

Having a large client base, RestAssured decides to perform a statistical study on the health status of his customers. The company wants to start from Milan, so it performs an aggregated research on *Data4Help*, setting the location to Milan and the time frame to the last month. Since the research produced more than 1000 results, *Data4Help* sends the result back to the company, making the data anonymous.

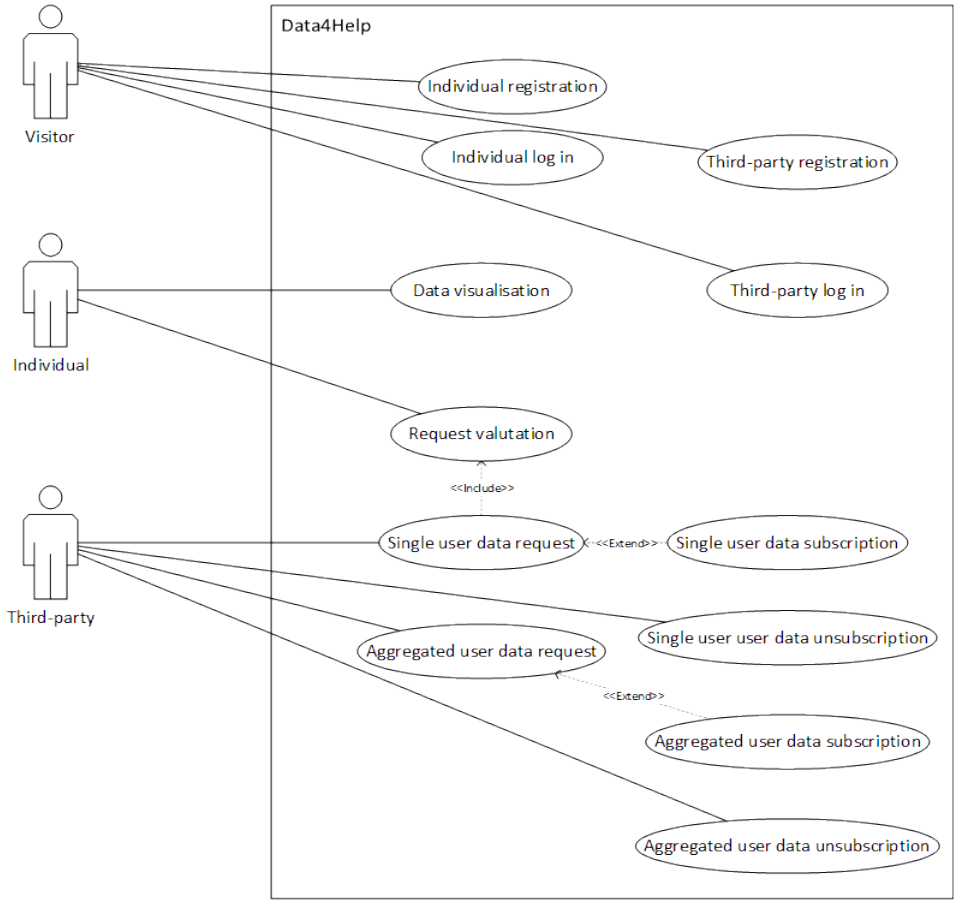


Figure 3.10: Use case diagram relative to *Data4Help*.

Individual registration and 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
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 "*Sign up now*" 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 he did not confirmed his email. In these situations, the system will show the user an error message.

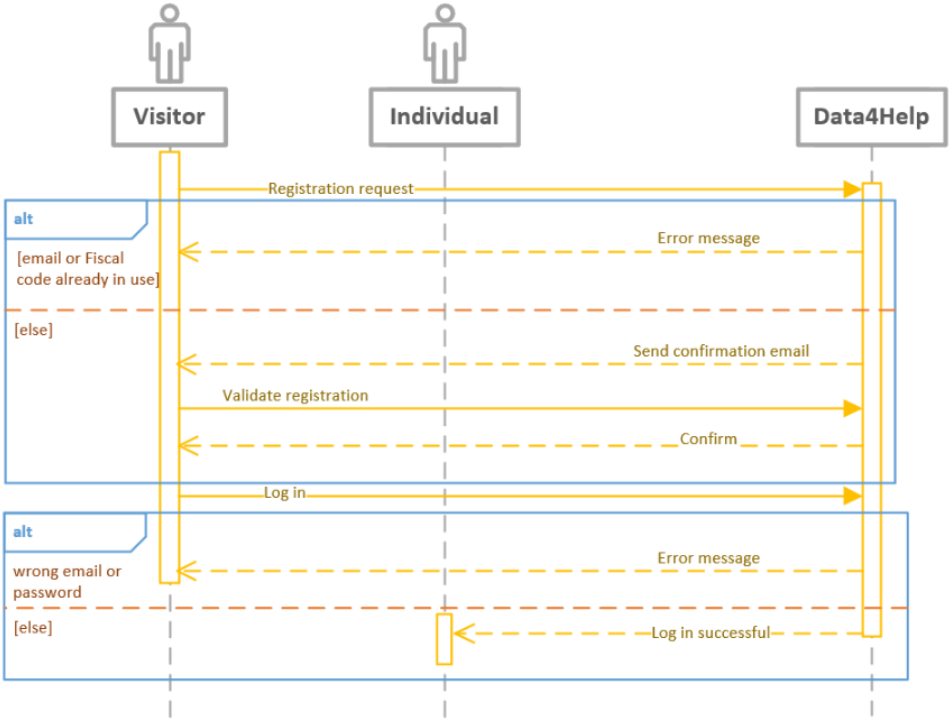


Figure 3.11: Sequence diagram representing the registration/log in phase of an Individual.

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
Entry Condition	There is no entry condition.
Events Flow	1. The Visitor accesses the registration page of the Enterprise application. 2. The Visitor clicks on the "*Sign up now*" 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.

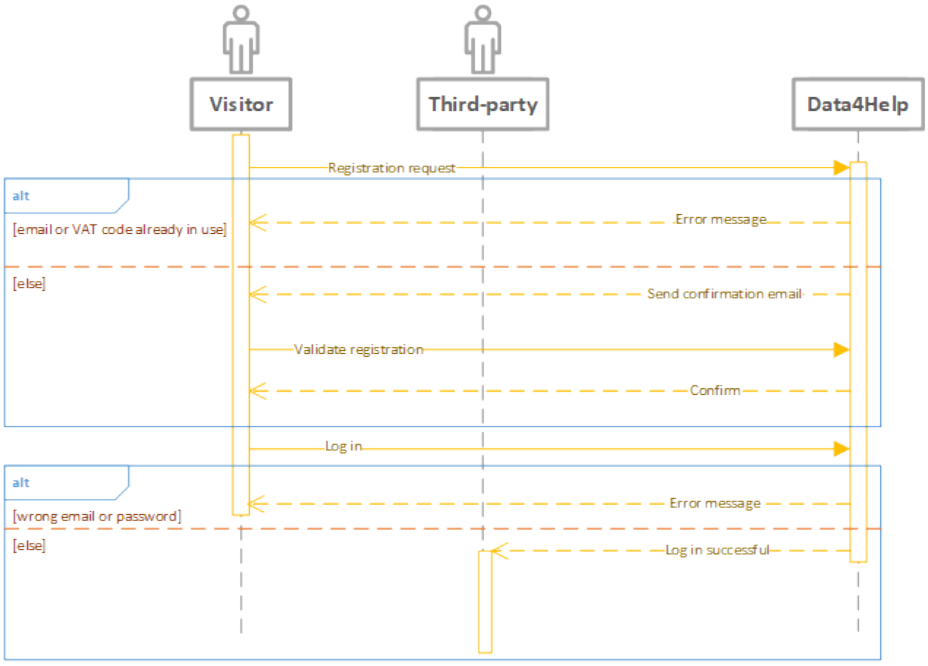


Figure 3.12: Sequence diagram representing the registration/log in phase of a Third-party.

Single user data request and subscription

Third-parties can perform a request to Data4Help for a specific user by specifying his/her fiscal code. Data4Help forwards it to the Individual 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
Entry Condition	Third-party has to be registered to Data4Help and logged in.
Events Flow	1. The Third-Party selects "*Individual Request*" and it inserts the Fiscal Code of the desired person and, if it wants to subscribe, granularity and frequency of update. 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 Individual receives the request with the name of the company that is requesting his/her data. 5. The Individual can accept or refuse the request. By accepting it, he/she allows the System to send all his/her health data to the applicant. 6. The system send notification to the Third-party.
Exit Condition	The Third-party has the user's collected data and receives updates of them periodically, if it has a subscription.
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.

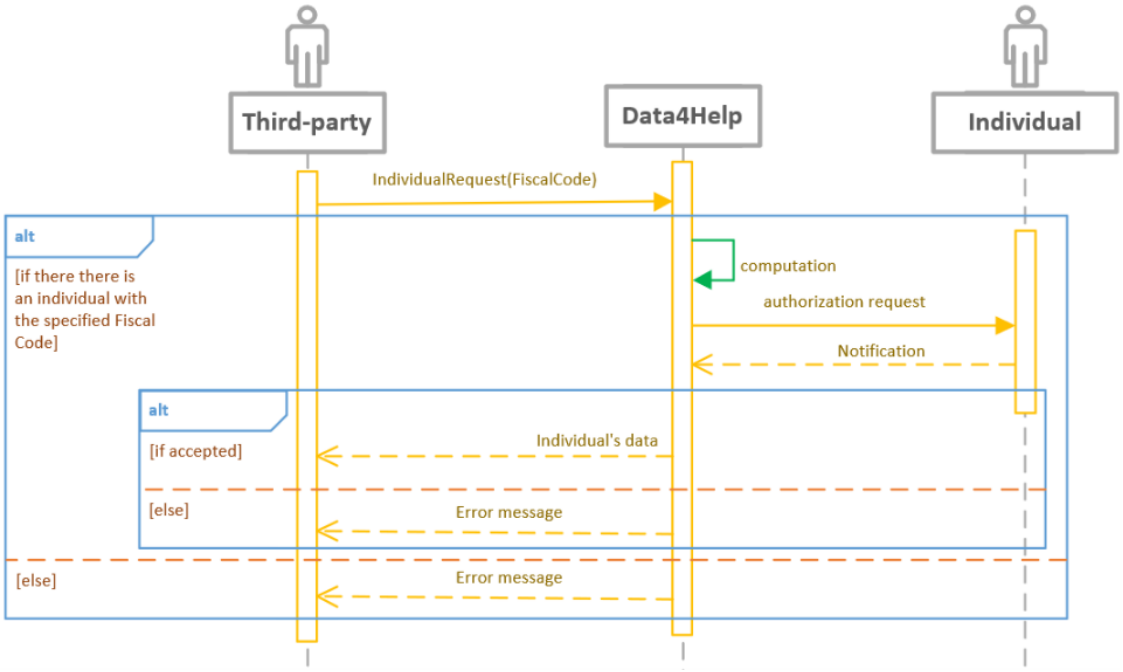


Figure 3.13: Sequence diagram representing the third-party Individual request.

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
Entry Condition	The Third-party is logged in.
Events Flow	1. The Third-party clicks on the "*Group request*" button. 2. The Third-party inserts the parameters of interest and, if it wants to subscribe, granularity and frequency of update. 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.

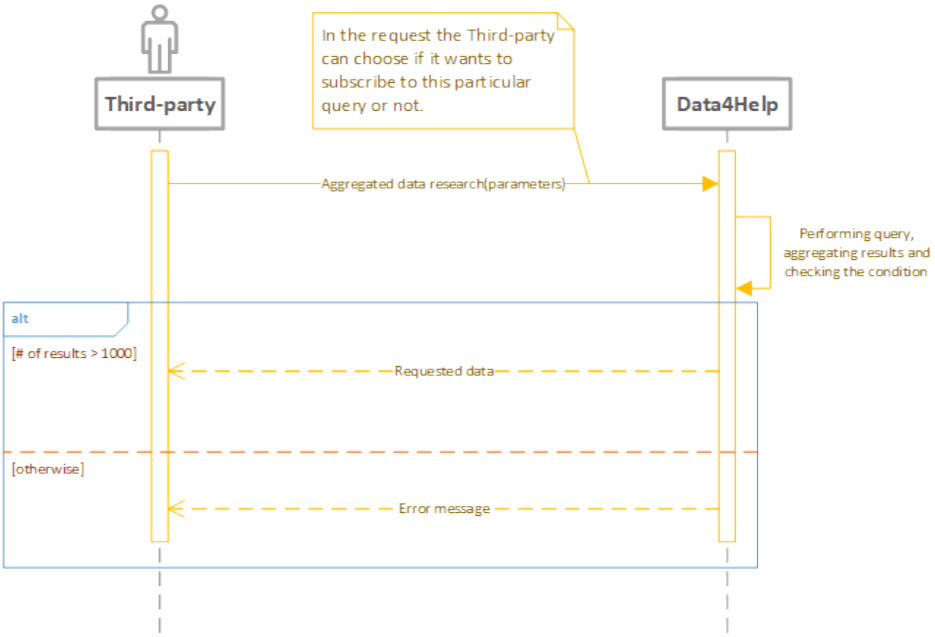


Figure 3.14: Sequence diagram representing the group requests performed by the Third-party.

Collected data during Subscription

After a successful subscription request, the system sends updates to the Third-party aggregating the data with the specified granularity and frequency.

Actors	• Third-party
Entry Condition	Third-party is logged in and has previously subscribed to the Individual's data.
Events Flow	1. When data arrived, the system checks if the Third-party is subscribed to single or aggregated user data. 2. If it is subscribed to aggregated user data, D4H evaluates the results. • If results are less than 1000, the system deletes the subscription and sends an error message to the Third-party. 3. The system sends updates to the Third-party aggregating the data with the specified granularity and frequency.
Exit Condition	Third-party receives updated data with specified granularity and frequency.
Exception	The system goes out from loop when *Single User data unsubscription* or *Aggregated user data unsubscription* starts, depending on what kind of data the Third-party is subscribed to.

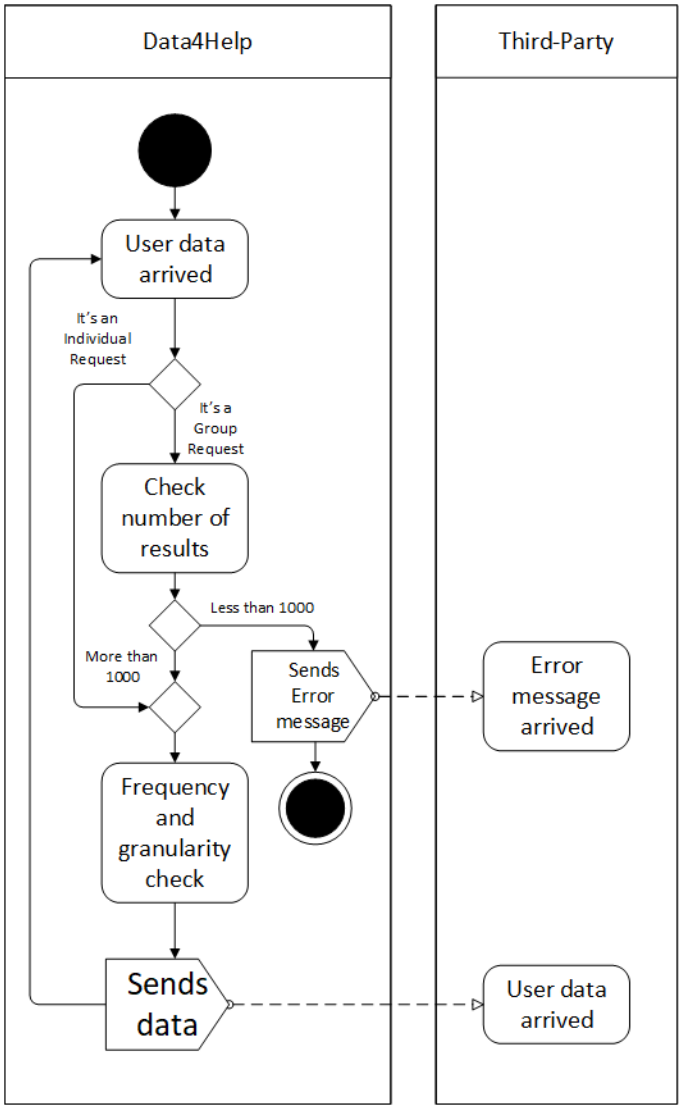


Figure 3.15: Activity diagram represents data sending during subscription.

Single User data unsubscription

After a successful subscription request, both Third-parties and Individuals can make a request in order to cancel the subscription to the specified data.

Actors	• Third-party • Individual
Entry Condition	Third-party is logged in and has previously subscribed to the Individual's data.
Events Flow	1. The Third-Party or Individual accesses to the subscription page and selects 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.

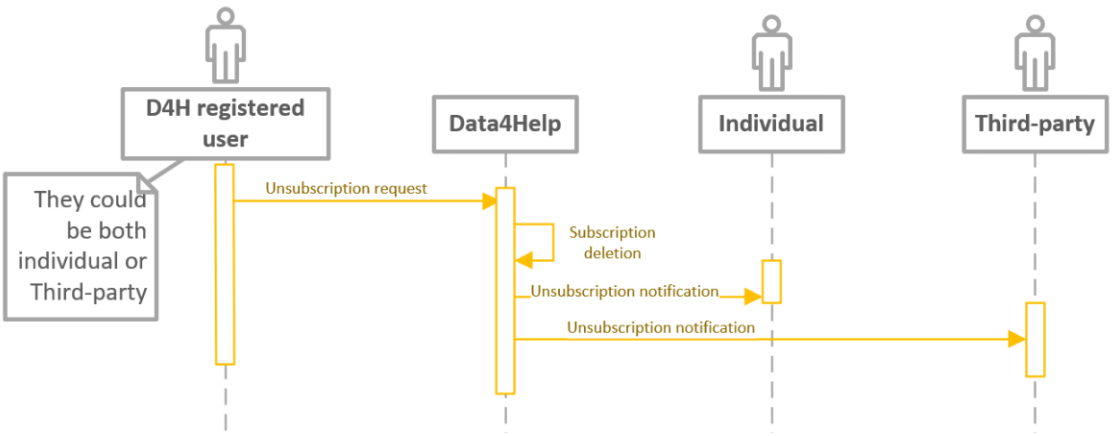


Figure 3.16: Sequence diagram representing the Unsubscription process for an individual data subscription.

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
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.

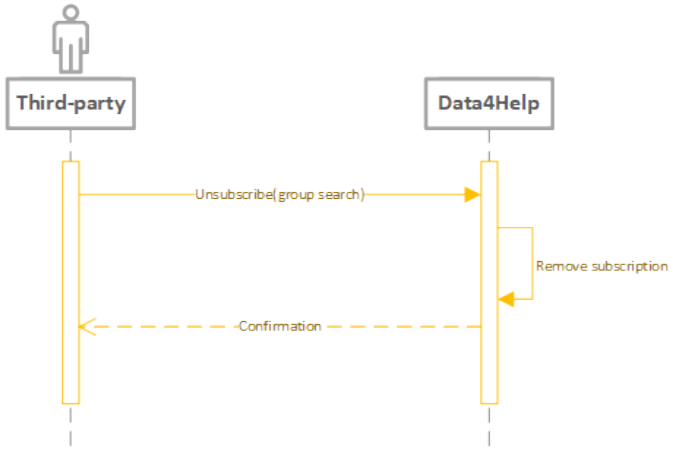


Figure 3.17: Sequence diagram representing the unsubscription from a group search by a Third-party.

Data Collection and Visualisation

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
Entry Condition	The user is correctly registered and logged into the D4H system.
Events Flow	1. Smart-watch 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 stored 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.

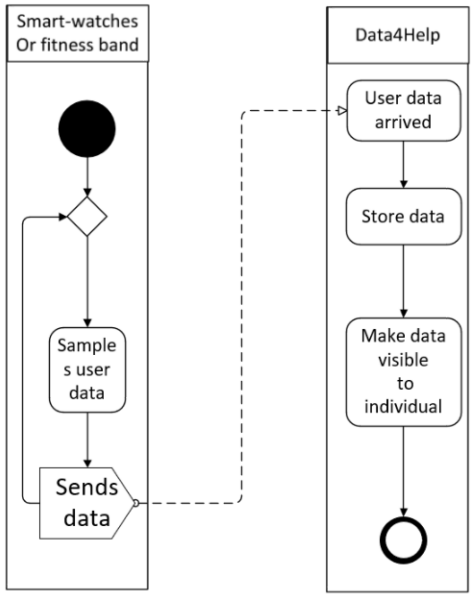


Figure 3.18: Activity diagram representing data storage process.

Third-parties request history

Third-parties are able to see all of the requests that they have done and still have access the data of the ones that were approved at the time.

Actors	• Third-party
Entry Condition	The Third-party is logged in and has made at least a request.
Events Flow	1. The Third-party clicks on the "*Request history*" tab. 2. The Third-party selects the request of which it wishes to see the details. 3. D4H shows the details (and the data, if the request was approved) to the Third-party.
Exit Condition	The Third-party is able to see the details of the selected request it has previously made.
Exception	If the Third-party has not performed any request, an error message is sent to it.

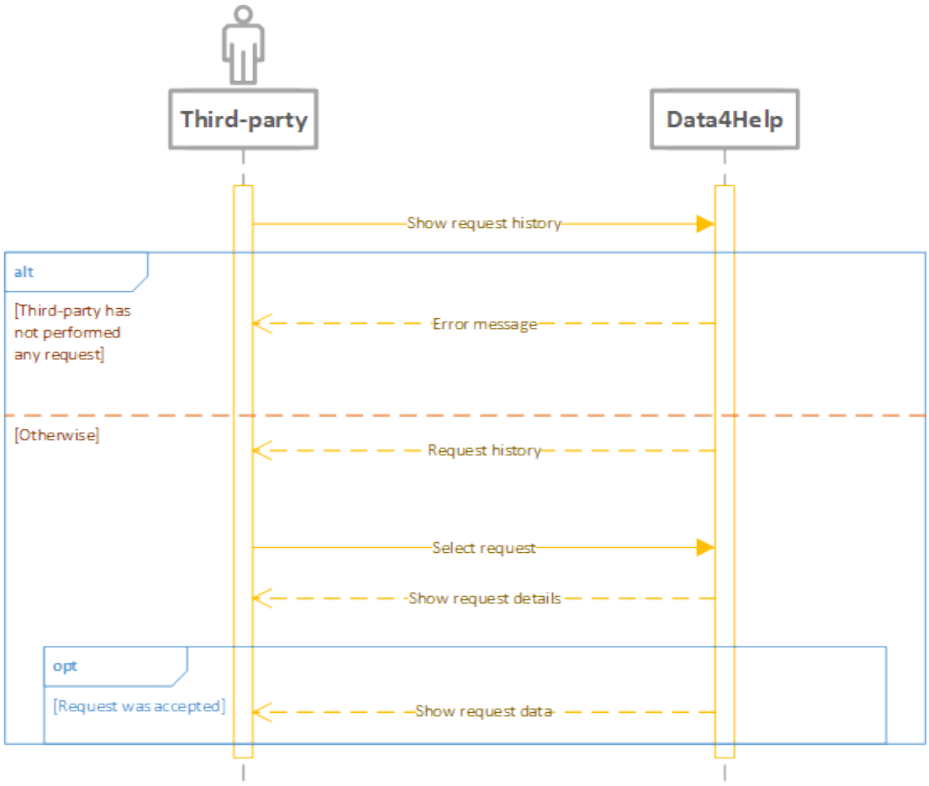


Figure 3.19: Sequence diagram representing the way a Third-party can access his request history.

3.2.2 AutomatedSOS

Consider the following scenario:

Leo is a 67 years old man, already registered to *Data4Help*. After the company has enabled *AutomatedSOS* service, he is asked if he wants to subscribe to it for an additional price. He accepts and then *TrackMe* sends him the instructions to follow in order to activate the service. Therefore, Leo opens *AutomatedSOS*. After that, he clicks the button to subscribe. *AutomatedSOS* evaluates and accepts the request (since Leo is 67).

One day, Leo is taken ill suddenly and his hearth rate goes below 40 bpm. The fitness band measures this value, sends it to *Data4Help* that forwards it to *AutomatedSOS*. The software recognises the measurement as critical and promptly sends to the ambulance service Leo's data and location. Within 5 minutes, an ambulance arrives at Leo's house and brings him to the hospital.

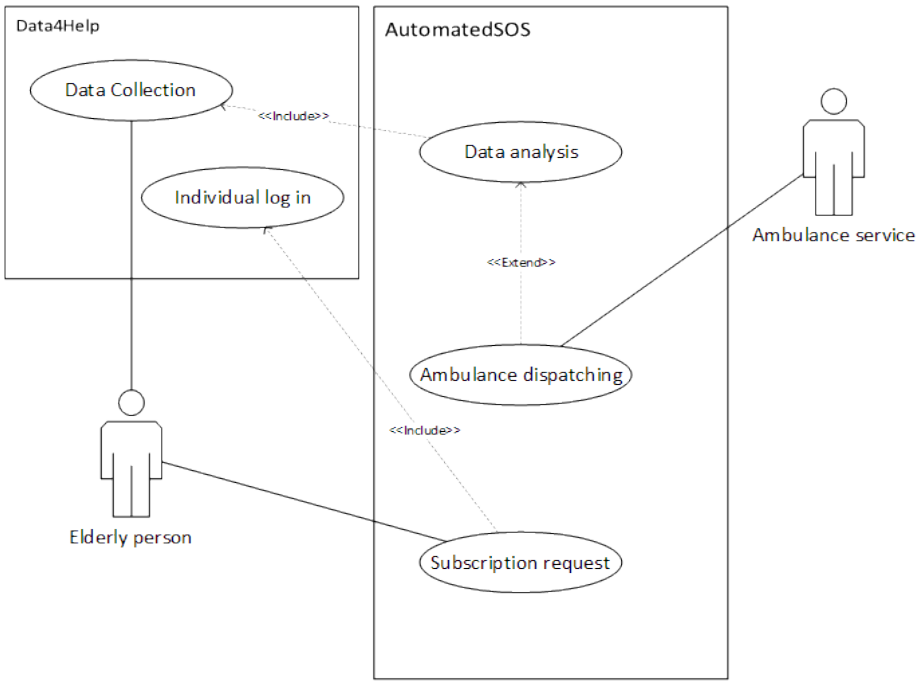


Figure 3.20: Use case diagram relative to *AutomatedSOS*. You can see the full use case diagram of D4H in the figure 3.10.

Subscription Request

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

Actors	• Individual
Entry Condition	The user has a D4H account and is logged in.
Events Flow	1. The Individual presses on "*AutomatedSOS service*", in order to make a new subscription request, then clicks on the "*Subscribe*" button. 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.

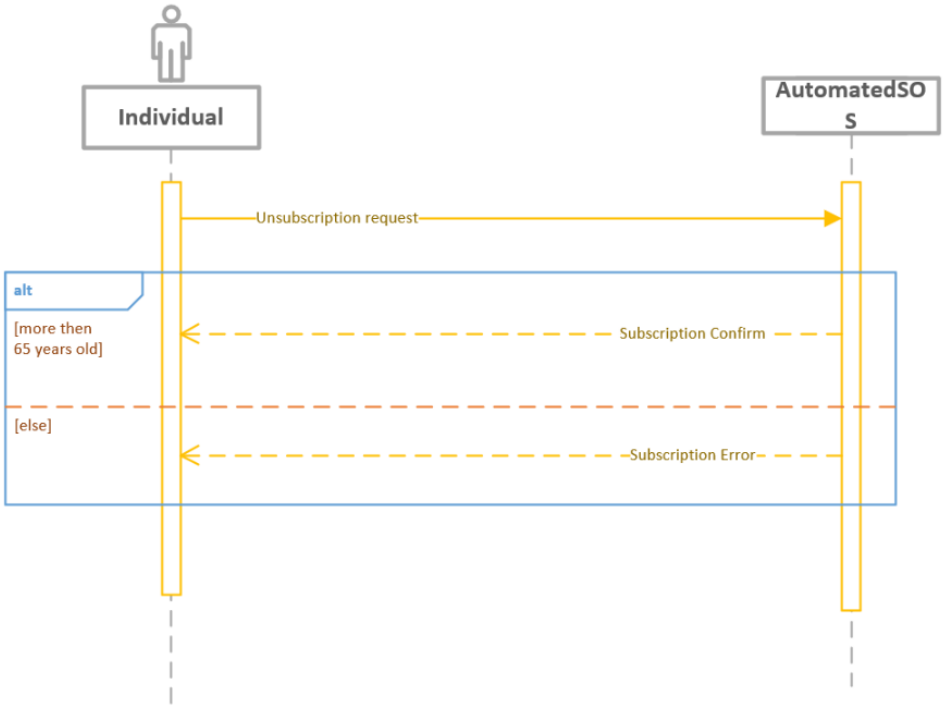


Figure 3.21: Sequence diagram representing the subscription request for ASOS service.

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
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 beyond 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.

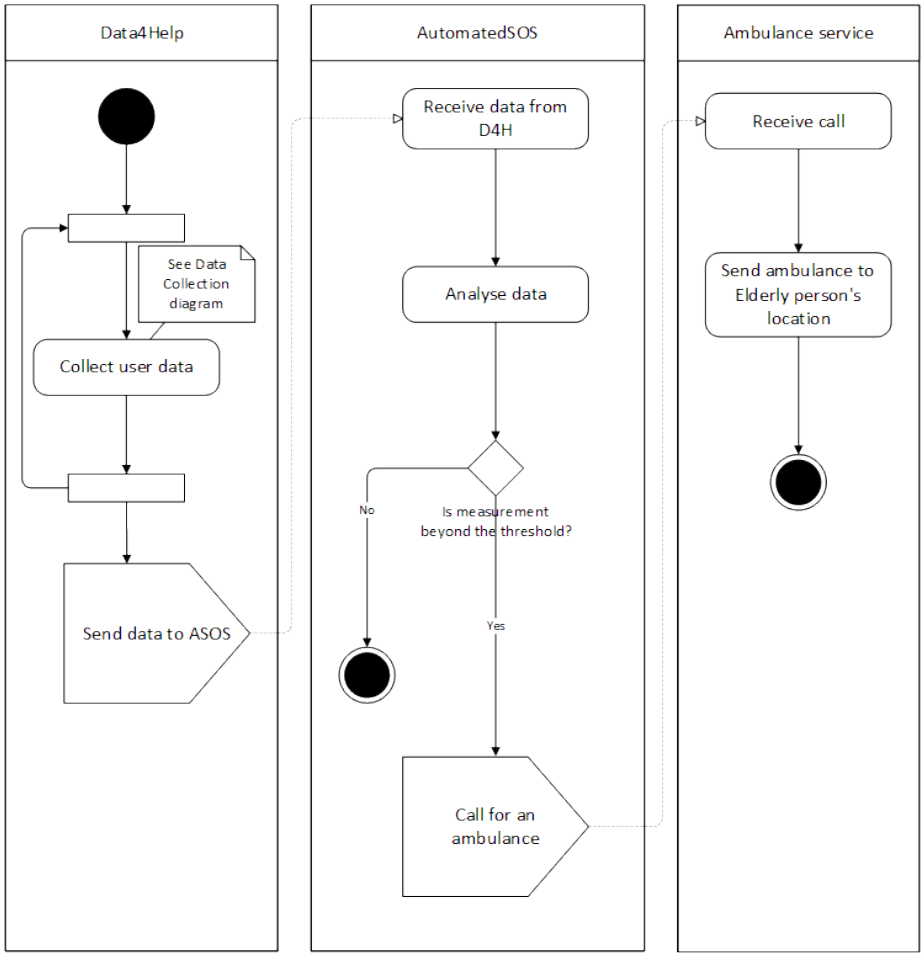


Figure 3.22: Activity diagram representing the call for an ambulance that ASOS performs when a critical measurement is collected by D4H. See figure 3.18 to see how D4H collects user data.

3.2.3 Track4Run

Consider the following scenario:

RunTogether is a group interested in the organisation of runs in Milan. After *TrackMe* has launched *Track4Run*, the company decides to download *Track4Run* Organiser edition and to register to the service. It then tries to publish a run held at Parco Sempione, on 11/11/2018 at 9.30, allowing 30 participants. Since there is another run scheduled at the same place and time, *Track4Run* shows an error message. RunTogether then decides to delay the competition until 12 and manages to publish it successfully. It then receives a confirmation mail with the summary of the process.

Matteo enjoys jogging and wants to try to compete in a race. Therefore, he downloads *Data4Help* application on his smart-phone and registers to the service. Then, he downloads *Track4Run* application and searches for the runs available in Milan in November. He then selects RunTogether's race, since it still has open slots, and enrols in it, pressing on the specific button. *Track4Run* sends him an email as confirmation. Fifteen minutes before the beginning of the race, Matteo is asked, through his D4H account, if he allows the acquisition of data, requested by T4R in order to track him during the race. He accepts.

The day of the race, Matteo's girlfriend, Jessica, wants to follow his route during the run. Therefore, she downloads *Track4Run*, without needing to register. She finds the run Matteo is enrolled in and selects it. *Track4Run* then requests the map of the area from Google Maps and sends it back to Jessica with the location of the participants updating in real time. Since she followed the race until the end, the rank is shown to her.

At the end of the race, Matteo wants to see the rank, so he opens *Track4Run*, taps on the rank history tab and finally selects the run he just attended. The system promptly shows the result to him.

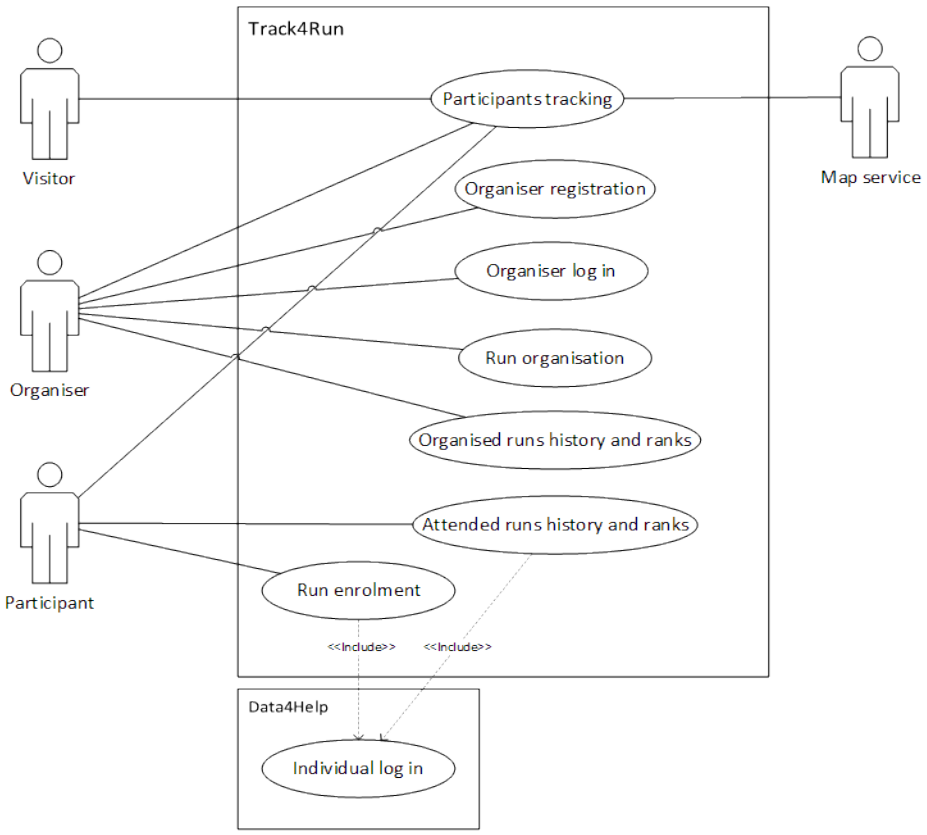


Figure 3.23: Use case diagram relative to *Track4Run*. You can see the full use case diagram of D4H in the figure 3.10.

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
Entry Condition	There is no entry condition.
Events Flow	1. The Visitor accesses the application log in page. 2. The Visitor clicks on the "*Sing up now*" link 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.

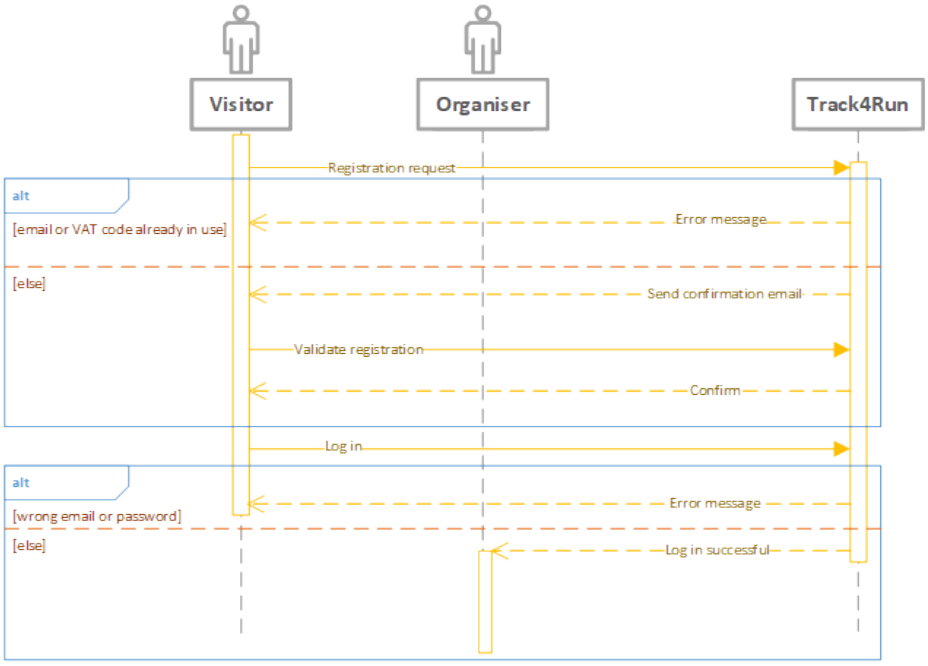


Figure 3.24: Sequence diagram representing the registration of an Organiser to T4R.

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
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.

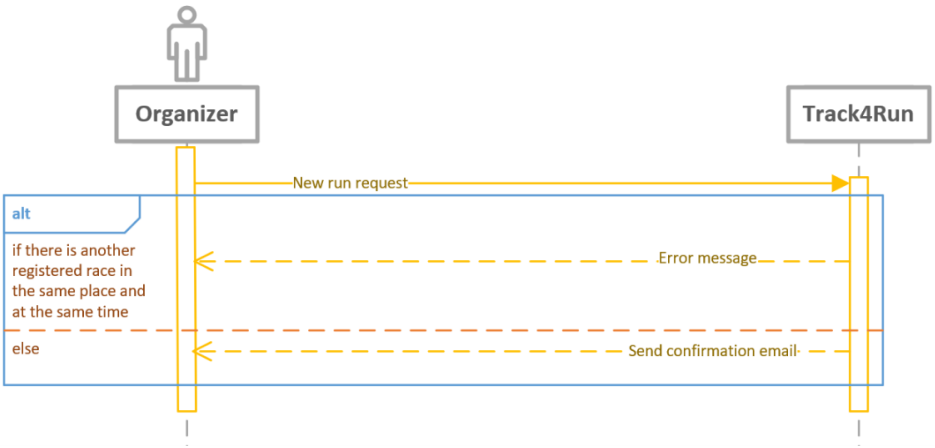


Figure 3.25: Sequence diagram representing the request in order to create a new run.

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
Entry Condition	The Participant is logged into T4R, using his D4H Individual account.
Events Flow	1. The participant clicks on the "*enroll run*" 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.

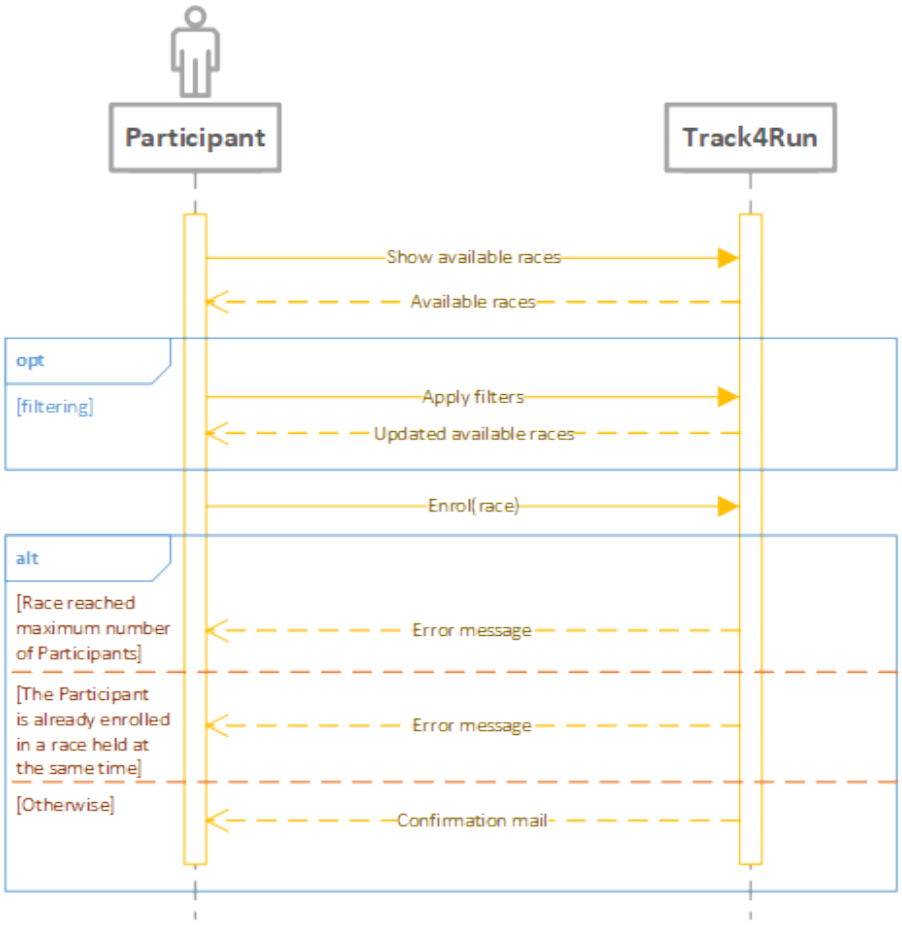


Figure 3.26: Sequence diagram representing the enrolment to a run of a Participant.

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. This can be done in both the Organisers edition and normal T4R application. In order to ask the permission to track the runners, T4R performs a single user data request through D4H for each of them. Therefore, in order to effectively participate to the run, each Participant has to accept the request, otherwise he gets removed from the race. The way the request is performed is shown in figure 3.13, where T4R acts as a Third-party.

Actors	• User • Map service
Entry Condition	There is no entry condition
Events Flow	1. The User launches the application. 2. They press on the "*Track runners*" button. 3. The system shows the list of current runs. 4. The User selects the run of interest. 5. The system asks for runners positions to D4H service. 6. D4H sends computed position to T4R. 7. T4R sends to Map service runners position 8. The map service sends to T4R map and runners location. 9. The system shows the map with all runners current position. 10. If the User follows the run until the end, T4R shows to him the rank.
Exit Condition	Users 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. • If the user disconnects, the system doesn't show Participant tracking anymore.

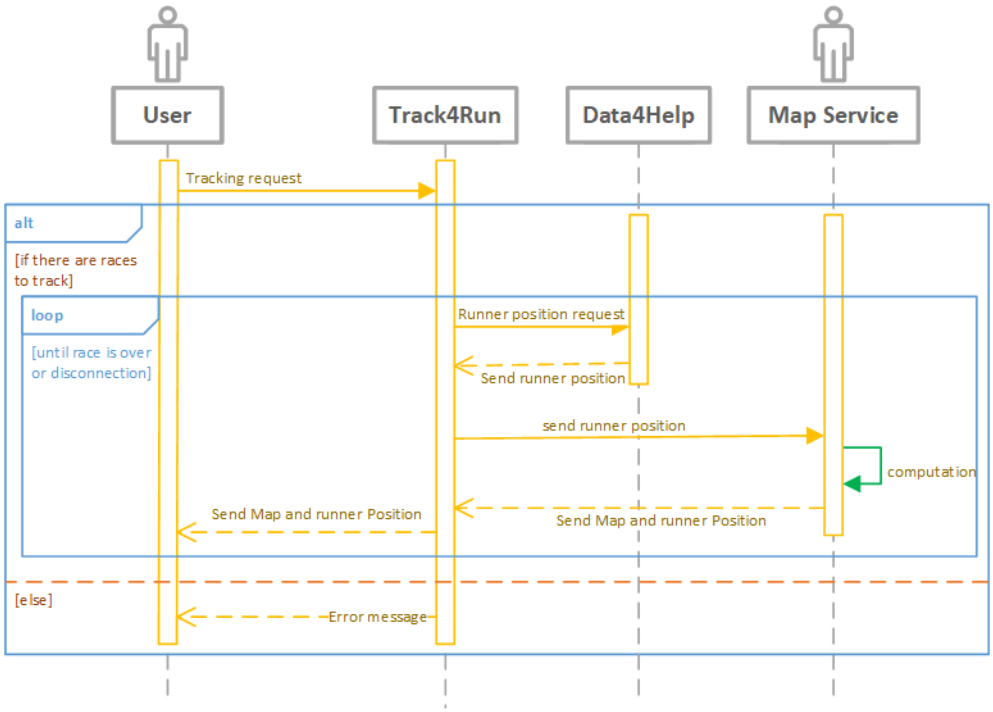


Figure 3.27: Sequence diagram representing the request in order to see participant position on a map during the run.

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. Of course, the two actors use the versione of the application dedicated to them.

Actors	• Participant / Organiser
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 "*Ranking*" tab. 2. 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.

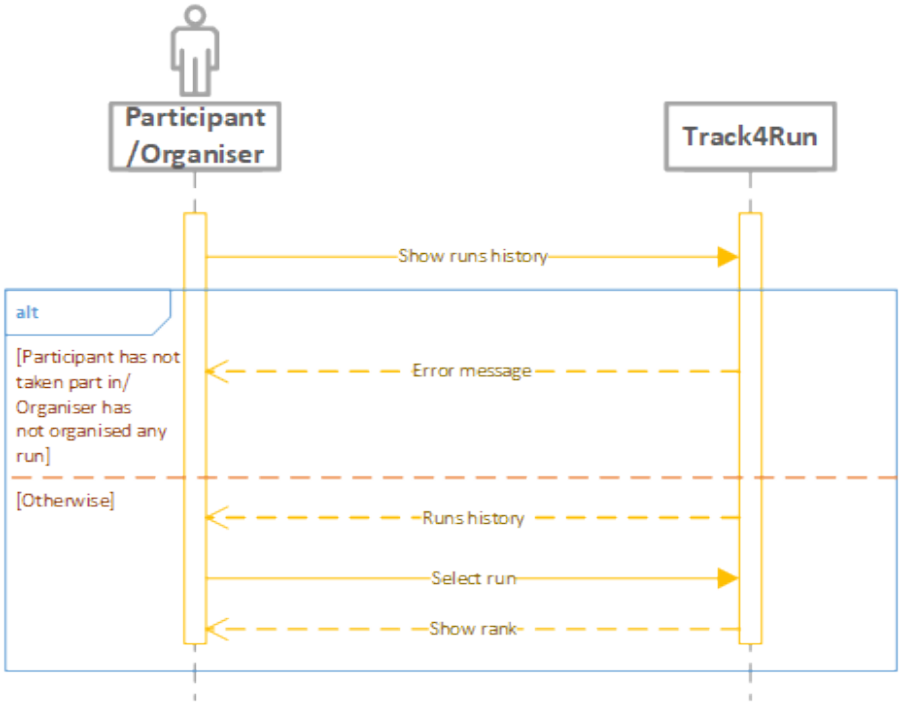


Figure 3.28: Sequence diagram representing request for the rank of a run.

3.2.4 Traceability matrix

In this section, we will keep track of the relationship between goals, requirements and use cases. In order to do this, we build for each analysed service *Traceability matrix*. It is important to highlight that the same use case could be linked to different requirements and the same requirement to different use cases.

Data4Help

GOAL	REQUIREMENT	USE CASE ID
[G1.1]	[R1.1]	Individual registration/log
[G1.1]	[R1.2]	Third-party registration and log in
[G1.1.1]	[R1.4]	Single user data request and subscription
[G1.1.3]	[R1.6]	Single user data request and subscription
[G1.2]	[R1.4.1]	Single user data request and subscription
[G1.2]	[R1.4.2]	Single user data request and subscription
[G1.1.3]	[R1.7]	Single User data unsubscription
[G1.1.3]	[R1.7.1]	Single User data unsubscription
[G1.1.3]	[R1.6]	Aggregated user data request and subscription
[G1.1.2]	[R1.5]	Aggregated user data request and subscription
[G1.1.2]	[R1.5.1]	Aggregated user data request and subscription
[G1.1.2]	[R1.5.2]	Aggregated user data request and subscription
[G1.1.2]	[R1.5.3]	Aggregated user data request and subscription
[G1.1.3]	[R1.7]	Aggregated user data unsubscription
[G1.1.3]	[R1.7.2]	Aggregated user data unsubscription
[G1.1.3]	[R1.6.1]	Collected data during Subscription
[G1.2]	[R1.3]	Data Collection and Visualisation
[G1.2]	[R1.9]	Data Collection and Visualisation
[G1.1]	[R1.8]	Third-parties request history

AutomatedSOS

GOAL	REQUIREMENT	USE CASE ID
[G2.1]	[R2.2]	Subscription Request
[G2.2]	[R2.1]	Ambulance dispatching

GOAL	REQUIREMENT	USE CASE ID
[G3.1]	[R3.6]	Organiser registration and log in
[G3.1]	[R3.7]	Run organisation
[G3.1]	[R3.9]	Run organisation
[G3.2]	[R3.1]	Enrolment to a run
[G3.2]	[R3.2]	Enrolment to a run
[G3.2]	[R3.8]	Enrolment to a run
[G3.3]	[R3.3]	Participant tracking
[G3.3]	[R3.4]	Participant tracking
[G3.1] [G3.2]	[R3.5]	Rank history

3.3 Design Constraints

3.3.1 Standards compliance

Since the software is initially meant to be used in Italy, it is important for it to be compliant towards the *General Data Protection Regulation* (GDPR) recently approved in Europe. This means that the software always has to:

- ask for the user approval when retrieving and using his/her data;
- allow him/her to know what data is being shared and with whom;
- make it possible for him/her to revoke the consent at any time.

3.3.2 Hardware limitations

In order to function properly, the three services need that some categories of users associate their smart-watch or fitness band to them. This concerns:

- Individuals for D4H;
- Elderly people for ASOS;
- Participants for T4R.

Note that both Elderly people and Participants must have a D4H account as Individuals in order to benefit from ASOS and T4R, respectively.

Moreover, the devices of all the users need to have an active internet connection in order for the software to work and bluetooth to communicate with smart-watches or fitness bands.

3.3.3 Operative Systems

The services will have to run on both Android and iOS. The Android version needs to work under Android 4.4 and above, while the iOS one has to support iOS 10 and above, so as to cover the majority of devices in circulation.

3.4 Software System Attributes

3.4.1 Performance requirements

The non functional requirements relative to performance of each service are listed below.

Data4Help

- The frequency of data acquisition must be of one every 30 seconds.
- Each request and response must be handled in less than 3 seconds.

AutomatedSOS

- The reaction time of the service, from the moment the parameters are measured to be beyond the threshold to when the ambulance is called, must be of less than 5 seconds.

Track4Run

- The tracking of the Participants must happen in real time.

3.4.2 Reliability

The software must be robust and fault tolerant: it is important that the system does not lose information if some component fails. This may require data redundancy.

3.4.3 Availability

The software must be available 24/7, a part from when it is under maintenance. These interventions have to be scheduled, when possible, and are expected to be performed once per month (e.g.: system upgrade, integration...).

3.4.4 Security

The software will handle private and sensitive data, therefore the level of security must be adequate. Information must be transmitted through HTTPS, passwords must be hashed and salted, databases must be encrypted with the latest standards.

3.4.5 Maintainability

The software should be organised in modules and be well documented, in order to make maintenance, upgrades and integration of new features easy.

3.4.6 Portability

The software must be able to run under different versions of Android and iOS environments, as already stated in section 3.3.3.

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 used to specify 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 Third Party that can be identified with a VAT number (in case of
    ↪ enterprise) or fiscal code (in case of individual).
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 if positive it 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 only 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 acquired 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 certificates 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 certificates that a reply for a multiple request contains data
    ↪  just if the number of users with that filter is grater than 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 Data4Help 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 explained before.

```
// This assertion checks 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 checks if a multiple request returns data if and only
    ↪ if the number of user with a
// specific requirement is grater 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 developed on top of Data4Help that allows the monitoring of the Elderly people, and sends 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 constraint.

4.2.1 Signatures

Here we present all the signatures 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 assumptions, 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
}
```

4.2.2 Facts

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

```
//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 (This is a
    ↪ simplification for the model, because we do not have a time and a
    ↪  date as identifiers for a request).
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 refers to the bpm of a user registered to ASOS
    ↪ so with an 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 explained before.

```
// This assert verify the absence of request with bpm not out of bound
assert ambulanceRequestWhenBpmNotOutOfBound{
    no request: AmbulanceRequest |
            (request.valueOutOfRange.isTrue and !request.bpm.outOfRange)
}

// This assert verify that when the bpm of a user are out of bound there
    ↪ is an ambulance request
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
        ↪ simplifying the model we choose smaller values, 4 and 2
    bpm > 4 or bpm < 2
}

//This pred returns true if the user is old (We consider old people, who
    ↪ have an age greater or equal than 65,
//but for simplification 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 Organizers to plan a race and define the path, Participants to enroll to the race and Spectators to track the participant during the run. The model below is built to verifies all these properties and verify one of them with 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 all the constraints needed in order to create a consistent model of Track4Run are defined.

```
// 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 are 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 partecipant
fact aVisitorLookToEveryPartecipant{
    all v: Visitor |
            #v.participants = #v.race.participants
}
```

4.3.3 Asserts

At the end this assert verifies if the model built respects the most critical constraint. The others can be visualised 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 provided by Alloy through the predicates above. 'Check' allows the analysis of the assertions and the verification of them, then 'show' displaies the model and verifies its consistency.

4.4.1 Data4Help results

These predicates for Data4Help verify if a single request returns 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 simplification).

```
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: This figure shows the results provided by the analyser after the execution of the predicates above.

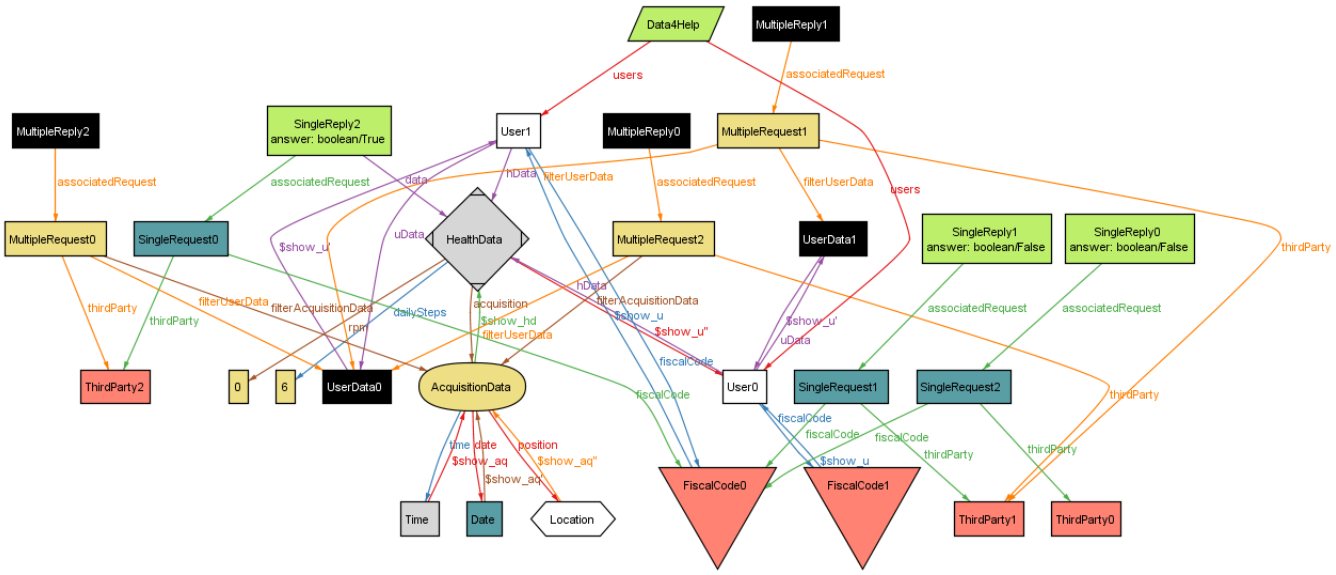


Figure 4.2: This figure represents a model of the world that includes: the case of single request accepted and refused and the case of multiple request refused.

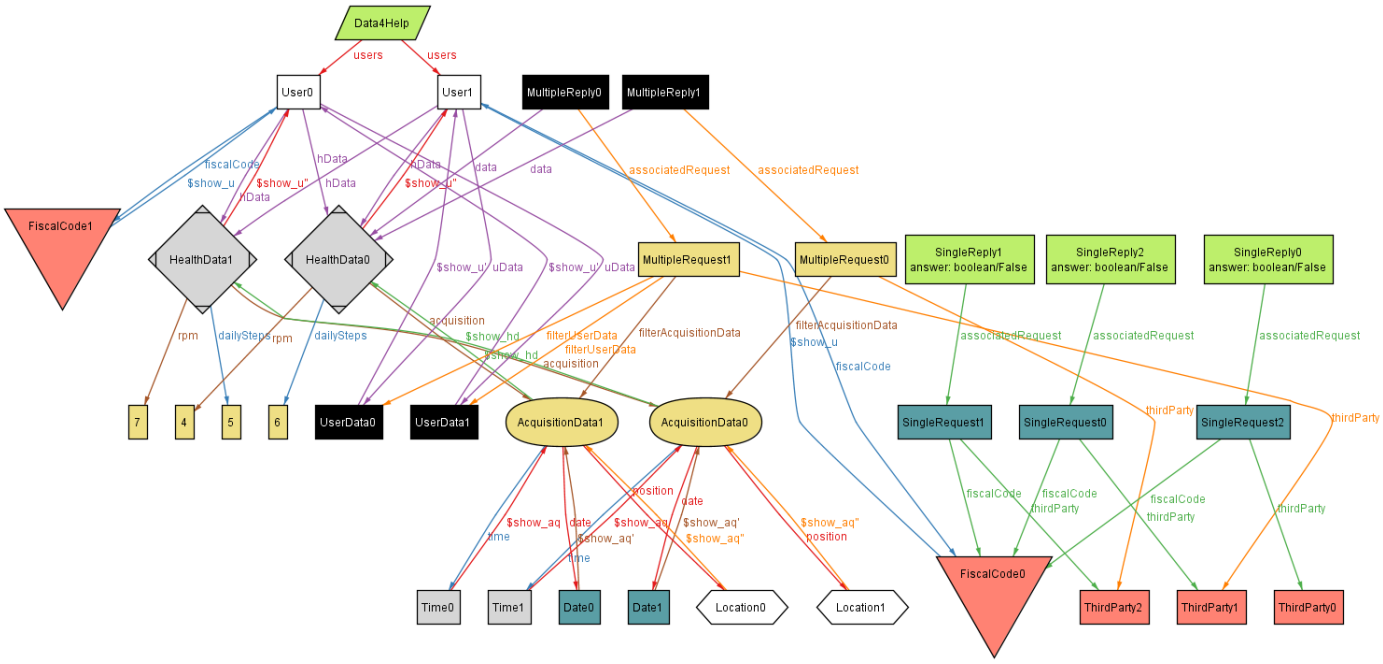


Figure 4.3: This figure represents a model of the world that includes the case of multiple request accepted.

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
```

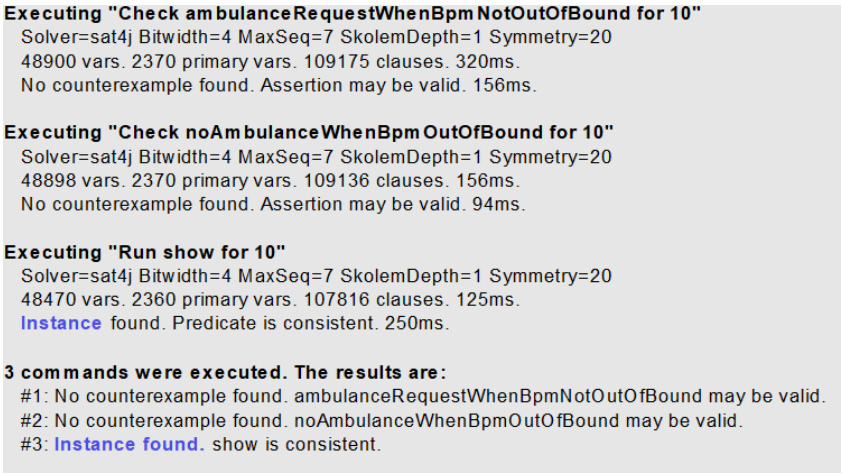


Figure 4.4: This figure shows the results provided by the analyser after the execution of the predicates above.

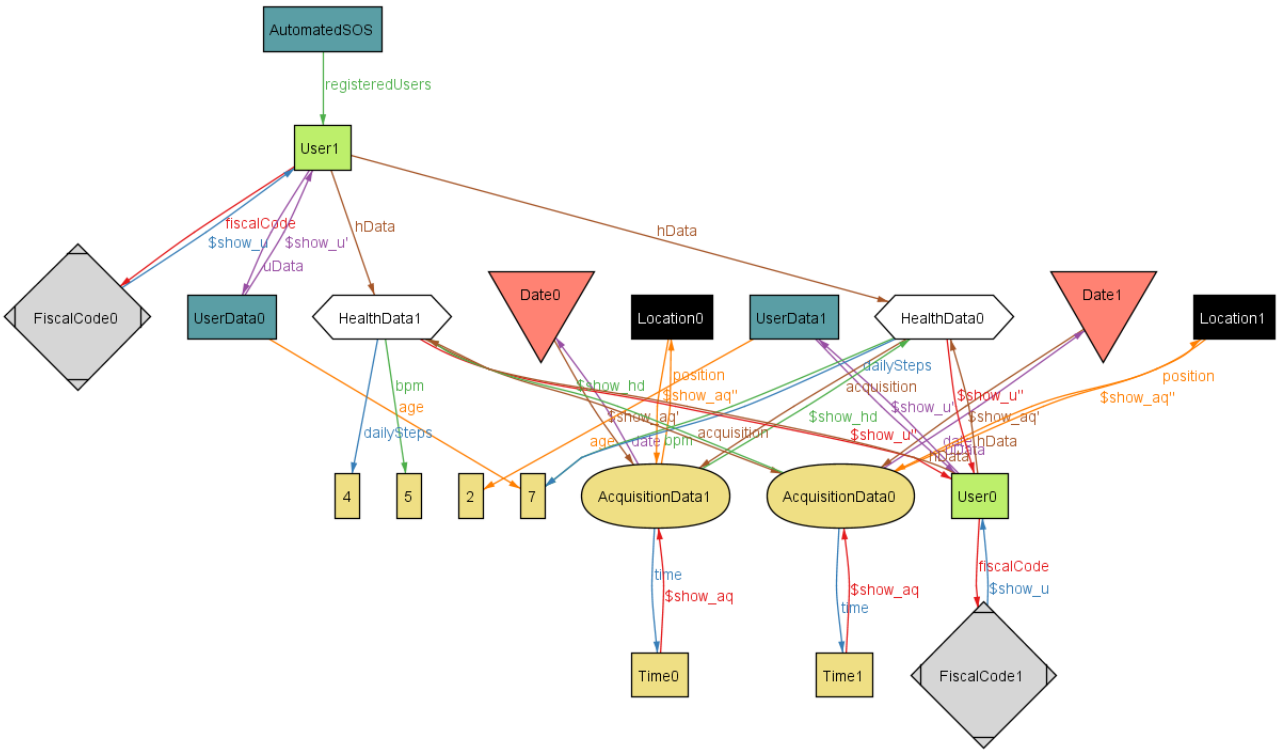


Figure 4.5: This figure shows an instance of the world where an individual has bpm inside the limits, then no ambulance requests have been created.

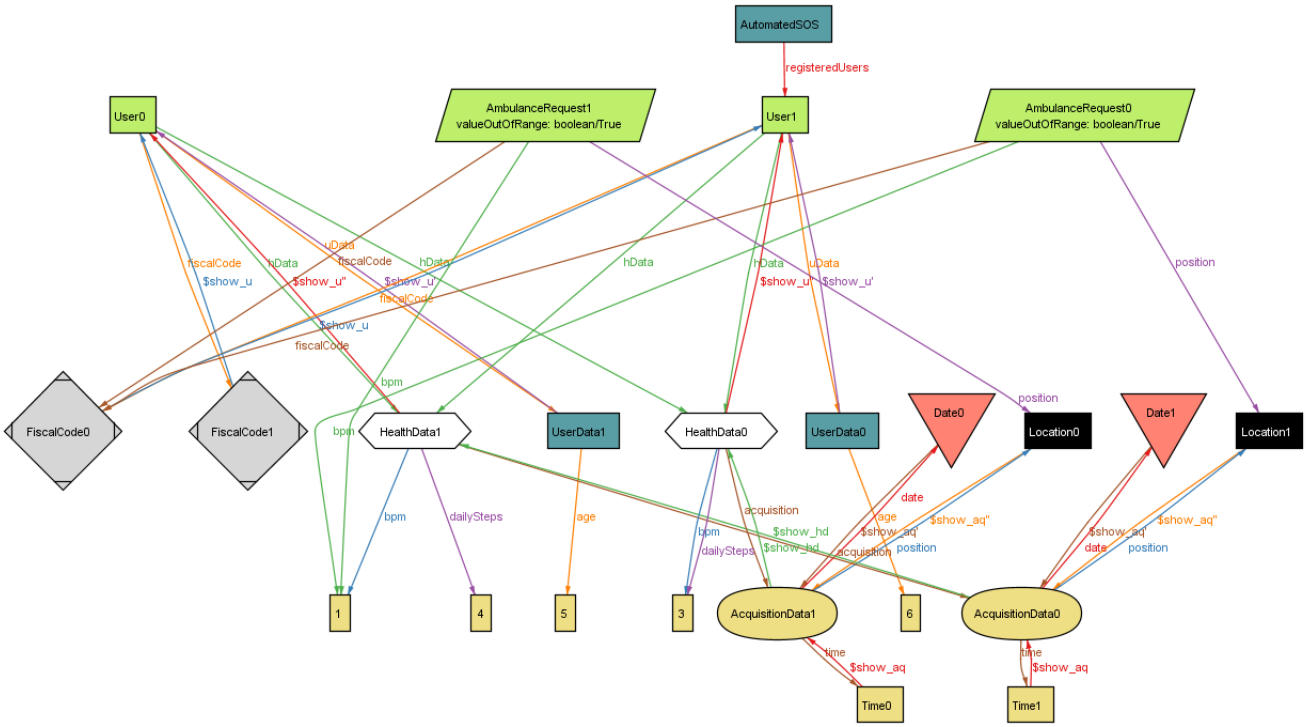


Figure 4.6: In this figure is represented the opposite case of the previous one, it displays the case where one Individual has Bpm out of critical range.

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: This figure shows the results provided by the analyser after the execution of the predicates above.

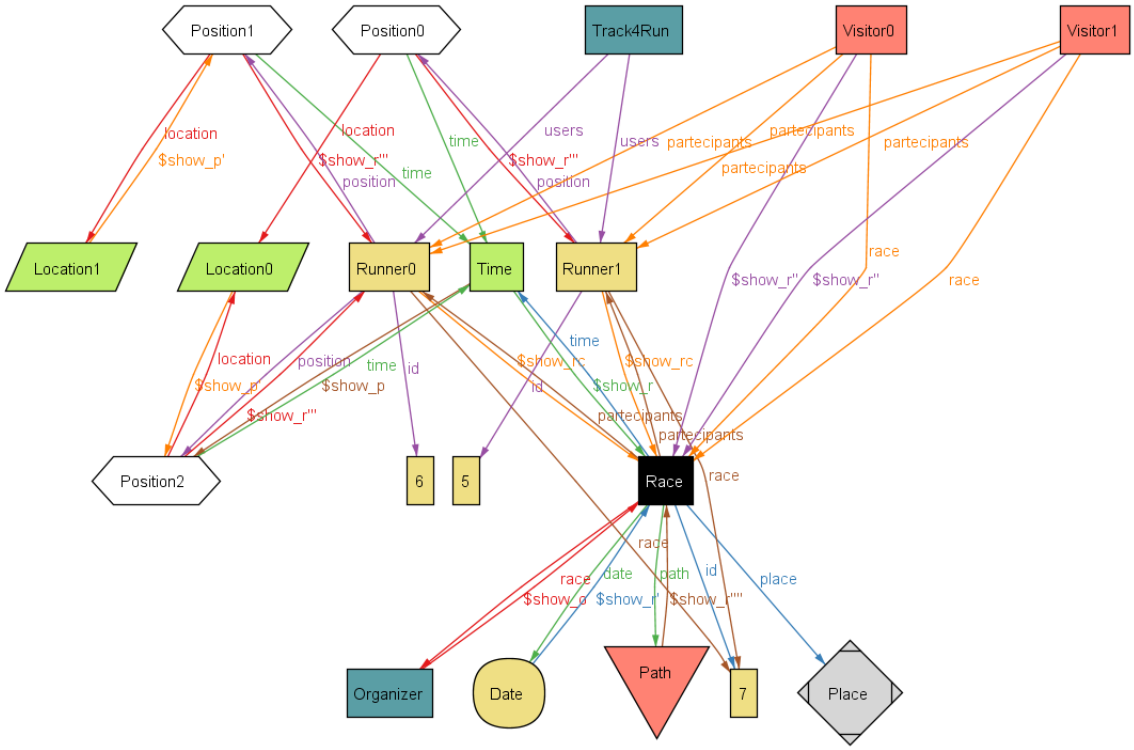


Figure 4.8: This figure represents an instance of the world create by the constraints imposed with the facts.

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
6/11/2018	2	Chapter 3 : Sequence diagrams
8/11/2018	3	Chapter 3 : mockups
9/11/2018	3	Chapter 3 : mockups
10/11/2018	2	Bibliogray and reference document
10/11/2018	4	Traceability matrix and use cases updates
11/11/2018	3	Final revision

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
09/11/2018	2	Chapter 3 scenarios and use cases revision
10/11/2018	2,5	Chapter 3 revision
11/11/2018	3	Final revision

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
11/11/2018	3	Final revision

Table 5.3: Gianluca Drappo's working hours

Bibliography

[1] Alloy mit tutorial. http://alloytools.org/tutorials/online/.

[2] Android distribution. https://developer.android.com/about/dashboards/.

[3] Gdpr, tutto ciò che c'è da sapere per essere in regola. https://www.agendadigitale.eu/cittadinanza-digitale/gdpr-tutto-cio-che-ce-da-sapere-per-essere-preparati/.

[4] ios distribution. https://developer.apple.com/support/app-store/.

[5] Latex forum to solve issues. https://tex.stackexchange.com.

[6] Latex guide. http://www.guit.sssup.it/downloads/LaTeX-facile.pdf.

[7] Latex tutorial. https://www.overleaf.com/learn.

[8] Mockplus tutorial. https://doc.mockplus.com/.

[9] Uml sequence diagram. https://www.ibm.com/developerworks/rational/library/3101.html.