TrackMe project - Argiro Anna Sofia,
Battaglia Gabriele, Bernardo Casasole

Requirement Analysis and Specification Document

Deliverable:	RASD
Title:	Requirement Analysis and Verification Document
Authors:	Argiro Anna Sofia, Battaglia Gabriele, Bernardo Casasole
Version:	1.2
Date:	10-November-2018
Download page:	https://github.com/BernardoCasasole/ArgiroBattagliaCasasole.git

Contents

1. Introduction

1.1 Purpose

1.1.1 Goals

G1 Allow users to properly use the services they wish to employ

G2 Allow visitor to register as individual or third-party user

G3 Allow individual users to monitor their location and health parameters

G4 Allow third-party users to request the data on specific users

G5 Allow individual users to approve or deny the specific request for their data

G6 Allow third-party users to request data on anonymized groups of individual users

G7 Call an ambulance if the system detects a critical health condition

G8 Allow third-party users to organize running competitions

G9 Allow Individual users to enrol in existing running competitions as participants

G10 Allow Individual users to subscribe in existing running competition as spectators to monitor underway competitions

1.2 Scope

The service Data4Help is designed to monitor the location and health status of individual registered to it using wearable devices. Upon registration individual users agree to the acquisition and usage of data by TrackMe and will be able to do see their own data. The collected data is available to be requested by registered third-party users. The first possibility is requesting data on a specific individual user, for which it is required an identification code of the individual user(Social security number, fiscal code, etc.); the second possibility is accessing anonymized data on groups of individual users, so the system will have to allow third-party users to filter the users by age, geographic area, weight, etc. while keeping the data properly anonymized. For both possibilities third-party users can simply acquire the data stored or subscribe to receive the data as soon as it is produced.

AutomatedSOS is a service designed to provide emergency health support for elderly people, to be integrated on top of Data4Help. Those individual users parameters about blood pressure, heartbeat and blood oxygenation are constantly checked: whenever they represent a critical health condition, AutomatedSOS, within 5 seconds from the drop of those values, contacts an emergency service asking to send an ambulance at the user location.

The main purpose of the Track4Run is to support running competition organizer and jogging lovers in convening great sport events with little effort. Namely Track4Run is a service-to-be focused on organizing running competition and on monitoring competitors. A user of Track4Run will also have to be a user of Data4Help.

Third-party users will be able to:

- organize a running competition specifying time, path of the competition, restrictions on participants and message for who wants to enroll;
- see the participants;
- cancel a run they have organized.

Individual users will be able to:

- discover new nearby runs upon their creation by an app notification;
- be notified if a run they are participants in is canceled;
- search for organized runs and register to participate in a future run or spectate an ongoing run.

During the run:

- organizer, participants and spectators will be able to see the name and positions of the participants;
- the system must be able to distinguish between actual participants and individual users who registered to the run but are not competing, simply checking if their position is on the run path.

1.3 Definitions

- *Visitor*:someone that has not registered in yet;
- *User*: a person, third-party or user, that has registered;
- *Individual User*: every registered person from whom the system collects data;
- *Run Participant*: an individual who wants to take part in a run;
- *Run Spectator*: an individual user who wants to spectate a run;
- *Third-Party User*:every entity registered with the purpose to request data for external use;
- *Run Organizer*: third-party users that wants to organize a run;
- *Emergency Service*: external participant that receives and take charge of the request for an ambulance;
- *Identification code*: a unique code for individual users or third-party users which identify them; for individual users it can be the Social security number, fiscal code, etc.; for third-party users it can be also the VAT number.
- *Stored Data*: the data on a IU collected so far.
- *Data Request*: a request for data made from a TPU.
- *Stored Data Request*: a data request for stored data.
- *Subscription Request*: a request for subscribing to newly generated data.

1.4 Acronyms

- RASD: Requirement Analysis and Specification Document.
- API: Application Programming Interface
- TPU: Third-party User
- RO: Run Organizer
- RS: Run Spectator
- RP: Run Participant
- ES: Emergency Service
- ID: Identification code
- D4H: Data4Help
- ASOS: AutomatedSOS
- T4R: Track4Run

1.5 Abbreviations

- Gn: n-goal
- Dn: n-Domain assumption
- Rn: n-Requirement

1.6 Revision history

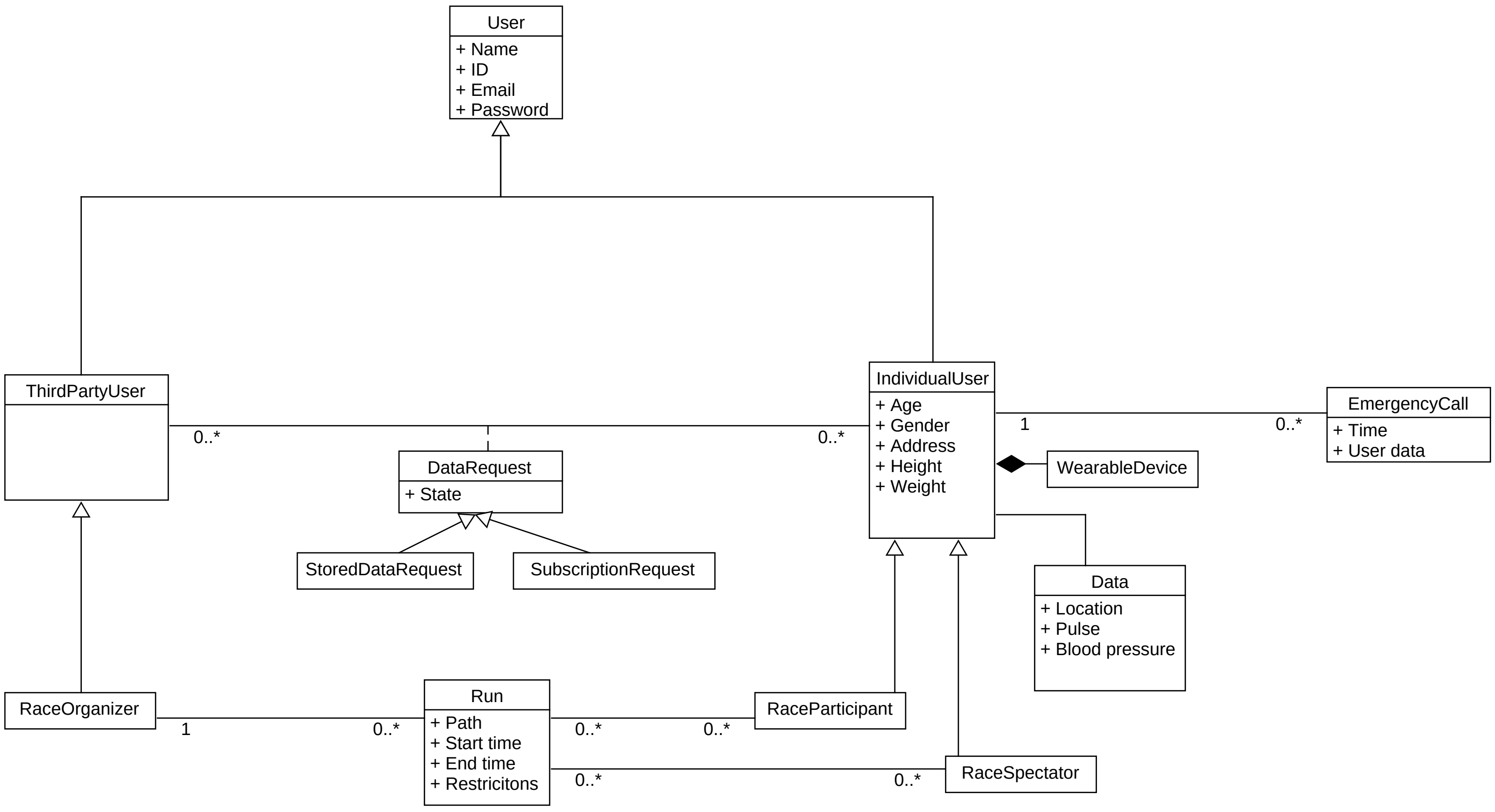
1.7 Reference Documents

1.8 Document Structure

2. Overview

2.1 Product perspective

The services are going to be used by Third party users interested in having access to statistical, individual data or in organizing competitions. In order to provide those services there will be a web page and, mobile application and a server application.
Individual users interact with the system via mobile application while third-party users interact via web page.
All services provided by Data4Help are based on the collection of data associated to an individual user: since his registration, data coming from users device are stored in a database on the server and continuously updated.
API documentation for Third-parties use will be available on the web page.


User
+ Name
+ ID
+ Email
+ Password
ThirdPartyUser
IndividualUser
+ Age
+ Gender
+ Address
+ Height
+ Weight
EmergencyCall
+ Time
+ User data
WearableDevice
DataRequest
+ State
StoredDataRequest
SubscriptionRequest
Data
+ Location
+ Pulse
+ Blood pressure
RaceOrganizer
Run
+ Path
+ Start time
+ End time
+ Restricitons
RaceParticipant
RaceSpectator
0..*
0..*
1
0..*
1
0..*
0..*
0..*
0..*
0..*

2.2 Product functions

2.2.1 Data management

By mobile application, individual users upon registration will provide their personal data (name, age, gender, address, ID, weight, height) into the system. Data4Help will grant access to their personal and collected data to third-party users if requested anonymously. If not, it will warn the individual user that a third-party user is questing access and ask for their approval; furthermore, the system will keep a record of third-party users subscribed to their data and it will allow the individual user to cancel the approval.

2.2.2 Data access

The Data4Help system upon registration of third-party users will require their ID and then will allow them to request saved data on individual users. Data4Help will allow them to request access to data on a specific individual user after providing a the ID of the individual user; it will notify them after the request is sent and when the user decides upon its approval.
Third-party users will be able also to request anonymized data by specifying a group of people according to relevant filters as age, weight, geographical area, etc. Data4Help will analyse the request and decide whether it is possible to properly anonymize the requested data and will notify the third-party upon its decision. If possible, will grant access to the requested data. Third-party users will be able to interact with the system by web interface or using the system APIs.

2.2.3 Data subscription

The Data4Help system will allow registered third-party users to subscribe to data on users. This service will provide data as soon as it is produced to the third-party without the need for constants approvals. In case a third-party subscribe to have live data for a specific user the system will act as with a stored data. If the user cancels a previously granted access Data4Help will warn the third-party users that had access to those data. Third-party users will be able to request data by web interface or using the released Data4Help APIs.

2.2.4 Emergency call

The Data4Help system will be able to forward a request for help to the Emergency Service within 5 seconds from the moment AutomatedSOS detects a dangerous drop in health parameters values. The Emergency Service takes charge of the request and sends an ambulance.

2.2.5 Running competition organization

The Track4Run system will allow users to organize running competition. It will require to a RO:

- time of the competition
- path of the competition
- restrictions on participants
- an optional message for who wants to enroll

The system provides the RO with data on the enrolled users.

2.2.6 Running competition monitoring

The Track4Run system will allow users to monitor a running competition; it is mandatory for a RS to subscribe to the run as a spectator. The system will provide the RS with names and real-time position of all the participants.

2.2.7 Running competition enrolling

The Track4Run system will allow users to enrol to an already existing competition. Track4Run will provide participants them with the final ranking and their personal data throughout the run: rate per kilometer, instantaneous speed, the missing and ran distance, the position.

2.3 User charateristics

- *Visitor*: a person, third-party, that has not registered in yet and has access only to registration;
- *User*: a person, third-party or user, that has registered;
- *Individual user*: every registered person from whom the system collects location and health data. The user has access to all data collected since his registration and to the AutomatedSOS and Track4Run services;
- *Run participant*: an individual who wants to enrol and take part in a run;
- *Run spectator*: an individual user who wants to spectate a run, namely to see all information on the run;
- *Third-party user*: every entity registered with the purpose to request data from Data4Help for external use or to organize a running competition exploiting Track4Run services;
- *Run organizer*: third-party users that wants to organize a run and receive all information on it and the participants
- *Emergency service*: external participant that receives and take charge of the request for an ambulance.

2.4 Assumptions

D1 The age, height, weight, address, gender and name provided by the user on themselves are correct

D2 The ID is unique

D3 Email address in unique

D4 The email is currently in use

D5 The kinds acd accuracy of data collected on the users health conditions is comprehensive and enough to determine their health status and their location whithin 10 meters

D6 The user has wearable device connected to a smartphone

D7 The device can successfully contact the Emergency Service 24/7

D8 Emergency Service takes charge of every request sending an ambulance

D9 The ROs organize only authorized running competitions respecting laws and common sense

2.5 Constraints

2.6 Dependencies

3. Specific Requirements

3.1 External Interface Requirements

3.2 Functional Requirements

3.2.1 Scenarios

Scenario 1

Alice's gym woluld like to collect data on their members so they asked Alice to install the Data4Help application. She downloaded it, entered her name, age, ID, password etc., she accepted the data sharing policy and, after receiving an email, confirmed her email address. Now she recived a notification from the app, asking if she wants to accept or deny the request for her data, made from her gym, which she accepts.

Scenario 2

Bob, historic user of Data4Help, is getting older and older so he wants to activate the emergency service, AutomatedSOS. He logs in the application and request the activation of the service. Incidentally the very next day, while working from home, he has a cardiac arrest. Immediatly the application calls the emergency service that send an ambulance to his house, saving his life.

Scenario 3

Charles and Devon are passionate runners and love to compete one with the other. To be able to find more opportunities to race they installe the Data4Help application and activate the Track4Run service. Soon they enroll in a race as partecipants but unfortunatly Devon the day before of the race get injured, so he forfeit from the run. The day after, wishing to know how his friend Charles is doing he opens the app and spectates the run. After Charles arrives first he gets a call from Devon telling him that next time he will arrive second.

Scenario 4

EvilCorp wants to collect data on the inhabitants of a small town. Felix the supervisor registers the company on the website, entering name, ID(VAT), etc. and waits for a confirmation email. Then he enters the website again and enters the data required to make a request for data on individual users living in the area. Unfortunately for EvilCorp there not enough users to properly anonymize the data so the request is rejected and Felix is fired.

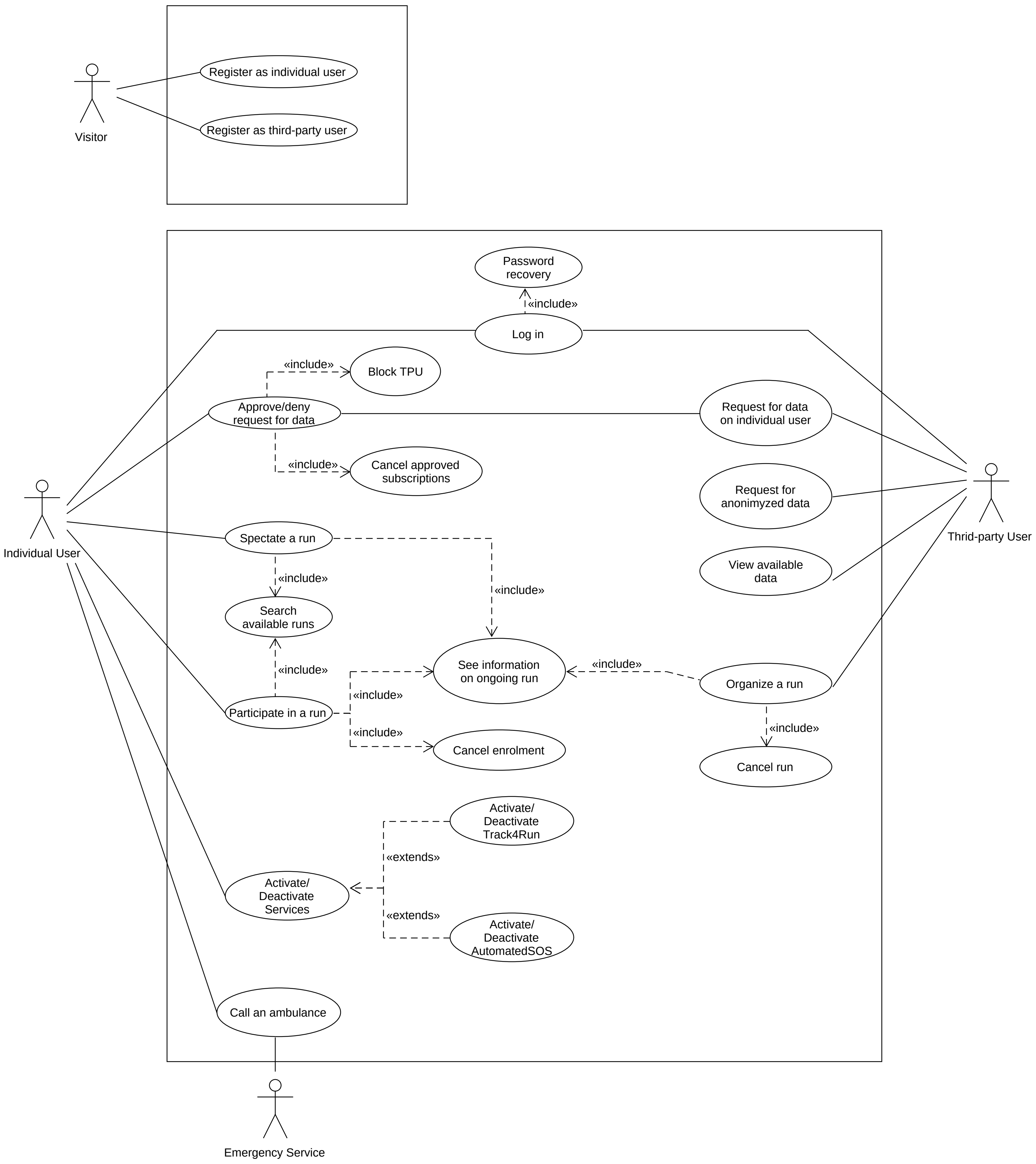
Scenario 5

GymForEveryone wants to personalize more the workout session of its premium members. In order to do so Helen, the manager, log in the gym account and sends, one by one, the request to the data of the premium members. The day after some of them have accepted the request while some of them denied it. With the data obtained she starts creating new workout sessions.

Scenario 6

IHealth wants to monitor large groups of people in order to collect data for research. In ordet to do that its researchers created a company account created via website. Jhon the scientist then makes a request to monitor the data of people living in Milan to evaluate the stress levels. Being able to properly anonymize the data, the system give him access to the live data. After some days, shocked by the results, start funding a campaign to urge Milan to work less and relax more.

3.2.2 Use cases


Visitor
Register as individual user
Register as third-party user
Password recovery
«include»
Log in
«include»
Block TPU
Approve/deny request for data
Request for data on individual user
«include»
Cancel approved subscriptions
Request for anonimyzed data
View available data
Individual User
Thrid-party User
Spectate a run
«include»
«include»
Search available runs
«include»
See information on ongoing run
«include»
Organize a run
Participate in a run
«include»
«include»
Cancel enrolment
«include»
Cancel run
Activate/ Deactivate Track4Run
«extends»
Activate/ Deactivate Services
«extends»
Activate/ Deactivate AutomatedSOS
Call an ambulance
Emergency Service

Registration as IU

ACTORS	Visitor
GOALS	**G2**
INPUT CONDITIONS	None
EVENT FLOW	1. The visitor downloads the mobile application 2. The visitor select the ”Sign in” option to register 3. The visitor fills all the mandatory fields (Name, ID, Age, Gender, Height, Weight, Address, Email, Password) 4. The visitor accepts the data policy 5. The system send an email to the Visitor 6. The Visitor confirms his email
OUTPUT CONDITIONS	The visitor is an IU
EXCEPTIONS	1. The email ot the ID is already in use 3. The visitor insert invalid data Exception cause the Visitor to be notified and the flow to go back to point 2.

Registration as TPU

ACTORS	Visitor
GOALS	**G2**
INPUT CONDITIONS	None
EVENT FLOW	1. The visitor opens the website 2. The visitor select the ”Sign in” option to register 3. The visitor fills all the mandatory fields (Name, ID(VAT), Address, Email, Password) 4. The system verify the VAT 5. The Visitor receive a confirmation email
OUTPUT CONDITIONS	The visitor becomes a TPU
EXCEPTIONS	1. The visitor is already a TPU 2. The email is already in use 3. The visitor doesn’t fill all the fields 4. The visitor insert invalid data Exception cause the Visitor to be notified and the flow to go back to point 2.

Login as IU

ACTORS	Individual User
GOALS	**G2**, **G3**
INPUT CONDITIONS	The IU is registered
EVENT FLOW	1. The IU opens the mobile application 2. The system shows them the home page 3. The IU insert email and password 4. The IU clicks the ”Log in” button 5. The system validate the fields entered 6. The system shows the IU health and location data
OUTPUT CONDITIONS	The IU is logged in
EXCEPTIONS	1. The IU enters invalid data Execeptions bring the flow back to point 2

Login as TPU

ACTORS	Third-Party User
GOALS	**G1**
INPUT CONDITIONS	The TPU is registered
EVENT FLOW	1. The TPU opens the website home page 2. The TPU insert email and password 3. The TPU clicks the ”Log in” button 4. The system validate the fields entered
OUTPUT CONDITIONS	The TPU is logged in
EXCEPTIONS	1. The TPU enters invalid data Execeptions bring the flow back to point 1

Password recovery

ACTORS	User
GOALS	**G1**
INPUT CONDITIONS	None
EVENT FLOW	1. The User opens the website 2. The User select the ”Forgot your password” option 3. The User provide their email 4. The system send a mail with a link to change password 5. The User follows the link and enters a new password
OUTPUT CONDITIONS	The User has changed password
EXCEPTIONS	1. The email doesn’t exist Exception cause the User to be notified and the flow to go back to point 1.

Request for IU data

ACTORS	Third-Party User, Individual User
GOALS	**G4**
INPUT CONDITIONS	The TPU is logged in
EVENT FLOW	1. The TPU opens the ”request data on IU” section 2. The TPU provide the IU ID 3. The system sends the IU a notification
OUTPUT CONDITIONS	The IU has a new request for data
EXCEPTIONS	1. The ID doesn’t match any IU Execeptions bring the flow back to point 1

Request Approval

ACTORS	Third-Party User, Individual User
GOALS	**G5**
INPUT CONDITIONS	The TPU is logged in
EVENT FLOW	1. The IU opens the ”request of data” section 2. The IU clicks on the ”Approve” button of a request 3. The system send a notification to the TPU that made the request
OUTPUT CONDITIONS	The request is approved
EXCEPTIONS	1. The IU doesn’t have any requests Execeptions bring the flow back to point 1

Request Denied

ACTORS	Third-Party User, Individual User
GOALS	**G5**
INPUT CONDITIONS	The TPU is logged in
EVENT FLOW	1. The IU opens the ”request of data” section 2. The IU clicks on the ”Deny” button of a request 3. The system send a notification to the TPU that made the request
OUTPUT CONDITIONS	The request is denied
EXCEPTIONS	1. The IU doesn’t have any requests Execeptions bring the flow back to point 1

TPU blocked

ACTORS	Third-Party User, Individual User
GOALS	**G5**
INPUT CONDITIONS	The IU is logged in
EVENT FLOW	1. The IU opens the ”request of data” section 2. The IU clicks on the ”Block” button of a request 3. The system send a notification to the TPU that made the request 5. The other requests from that TPU are denied
OUTPUT CONDITIONS	The TPU is blocked
EXCEPTIONS	1. The IU doesn’t have any requests Execeptions bring the flow back to point 1

Request for anonymized data

ACTORS	Third-Party User
GOALS	**G6**
INPUT CONDITIONS	The TPU is logged in
EVENT FLOW	1. The TPU opens the ”request anonymized data” section 2. The TPU provide the request parameters 3. The system evaluate the data requested are properly anonymized 4. The system sends a notification to the TPU
OUTPUT CONDITIONS	The TPU has access to the data
EXCEPTIONS	1. The data cannot be properly anonymized 2. The data entered is incorrect Execeptions bring the flow back to point 1

AutomatedSos activation

The activation of the Track4Run is omitted since is equivalent

ACTORS	IU
GOALS	**G1**
INPUT CONDITIONS	The IU is alredy registered and has not yet activated the AutomatedSOS service
EVENT FLOW	1. The IU opens the application 2. The IU opens the AutomatedSOS section 3. The system show the relevant information on the system 4. The IU clicks on the button to activate the service
OUTPUT CONDITIONS	The IU joined the service
EXCEPTIONS	None

AutomatedSos deactivation

ACTORS	IU
GOALS	**G1**
INPUT CONDITIONS	The IU is alredy registered and has activated the AutomatedSOS service
EVENT FLOW	1. The IU opens the application 2. The IU opens the AutomatedSOS section 3. The IU open the options menu 4. The IU clicks on the button to deactivate the service
OUTPUT CONDITIONS	The service is deactivated
EXCEPTIONS	None

Track4Run deactivation

ACTORS	IU
GOALS	**G1**
INPUT CONDITIONS	The IU is alredy registered and has activated the Track4Run service
EVENT FLOW	1. The IU opens the application 2. The IU opens the Track4Run section 3. The IU open the options menu 4. The IU clicks on the button to deactivate the service 5. The system unregister the IU as participant in any run they enrolled
OUTPUT CONDITIONS	The service is deactivated
EXCEPTIONS	None

Emergency call

ACTORS	IU, Emergency Service
GOALS	**G7**
INPUT CONDITIONS	IU has activated the AutomatedSOS service
EVENT FLOW	1. The system register a critical health condition 2. The system prompts a warning and a ”Don’t call” button for a few seconds 3. The system contacts the ES and gives him all relevant information
OUTPUT CONDITIONS	The ES is in charge on the call
EXCEPTIONS	1. The IU clicks the ”Don’t call” button Exception cause the IU to be notified and the flow to go back to point 1.

Organize a run

ACTORS	TPU
GOALS	**G8**
INPUT CONDITIONS	TPU is logged in the website
EVENT FLOW	1. The TPU opens the ”Organize” section 2. The TPU is shown a map to choose the path 3. The TPU enters a time, duration, restriction on participants and a message for the participants 4. The TPU confirms the creation
OUTPUT CONDITIONS	The run is organized
EXCEPTIONS	None

Cancel run

ACTORS	TPU, IU
GOALS	**G8**
INPUT CONDITIONS	The TPU is on the home page
EVENT FLOW	1. The TPU open the ”Organized runs” section 2. The TPU selects a run 3. The system shows the enrolled participants and their number 4. The TPU clicks the ”Cancel run” button 5. The Participants are notified by the system
OUTPUT CONDITIONS	The run is canceled
EXCEPTIONS	1. The TPU hasn’t organized any run Exception cause the flow to go back to point 1.

Spectate a run

ACTORS	IU
GOALS	**G10**
INPUT CONDITIONS	IU is in the Track4Run section of the application
EVENT FLOW	1. The system show a list of available runs, from closest to farthest 2. The IU selects one run 3. The IU clicks the ”Spectate” button 4. The system shows the map and all info on the participants
OUTPUT CONDITIONS	The IU is spectating the run
EXCEPTIONS	1. There are no run to spectate 2. The selected run has not yet started Exception cause the flow to go back to point 1.

Participate in a run

ACTORS	IU
GOALS	**G9**
INPUT CONDITIONS	The IU is in the Track4Run section of the application
EVENT FLOW	1. The system show a list of available runs, from closest to farthest 2. The IU selects one run 3. The system shows date, duration, path and the message of the organizer 4. The IU clicks the ”Pariticipate” button 5. The system adds the IU to the participants
OUTPUT CONDITIONS	The IU is a participating in the run
EXCEPTIONS	1. There are no run to take part in 2. The selected run has already started 3. The run restrictions prevent the IU from partecipating Exception cause the flow to go back to point 1.

Cancel enrolement

ACTORS	IU
GOALS	**G1**, **G9**
INPUT CONDITIONS	The IU is in the Track4Run section of the application
EVENT FLOW	1. The IU opens the section of a run he is enrolled in 2. The IU clicks on ”Unsuscribe” option
OUTPUT CONDITIONS	The IU is not longer a participant in the run
EXCEPTIONS	1. There are no run to unsuscribe from Exception cause the flow to go back to point 1.

3.2.3 Requirements mapping

G1 Allow users to properly use the services they wish to employ

R1 The Users must be able to log in

R2 An IU must be able to opt in and out of any additional service besides D4H

G2 Allow visitor to register as individual or third-party user

D1, **D2**, **D3**, **D4**

R4 A visitor must be able to register to D4H

R4.1 A visitor must insert name, age, gender, height, weight, ID, address, email and password

R4.2 A third party must insert name, ID, email and password

G3 Allow individual users to monitor their location and health parameters

D5, **D6**

R5 The system must provide to logged IU their current location and health parameters

R6 The system must collect and keep track of the location and health data

G4 Allow third-party users to request the data on specific users

D2

R7 A logged in TPU must be able to request the stored data or subscribe to newly produced data, after providing an ID of the user

R7.1 A TPU must be warned whether the data they requested is available

R8 A TPU must be able to obtain the data it has access to

G5 Allow individual users to approve or deny the specific request for their data

R9 The IU must be notified when a TPU request their data

R10 An IU must be able to accept the request on their data

R11.1 An IU must be able to deny access to their data

R11.2 An IU must be able to deny block a TPU to automatically deny their future requests for data

R12 User must be able to cancel the TPU subscriptions to their data that they previously accepted

G6 Allow third-party users to request data on anonymized groups of individual users

R13.1 A logged in TPU must be able to request the stored data or subscribe to newly produced data, after providing the specifics of the group it is interested in

R13.2 A TPU must be warned whether it is possible to properly anonymize the data requested

R8 A TPU must be able to obtain the data it has access to

G7 Call an ambulance if the system detects a critical health condition

D5, **D7**, **D8**

R14 When a critical condition is registered, the system should contact the ES and provide it location and health status of the IU

R15 The IU must be able to cancel the call

G8 Allow third-party users to organize running competitions

D9

R16.1 A TPU must be able to create a new run, specifying time, duration, path, restriction for enrolment

R16.2 A run organize must be able to must be able to cancel a run

R17 Run organizers, participants and spectators must be able to cancel a run see the status of the ongoing run

G9 Allow Individual users to enrol in existing running competitions as participants

D9

R18.1 An IU must be able to enrol in a future run

R18.2A participant must be able to cancel their inscription to the run

R17 Run organizers, participants and spectators must be able to cancel a run see the status of the ongoing run

G10 Allow Individual users to subscribe in existing running competition as spectators to monitor underway competitions

D9

R19 An IU must be able to spectate a ongoing run

R17o Run organizers, participants and spectators must be able to cancel a run see the status of the ongoing run

3.3 Performance Requirements

Data4help mobile application must respond within 5 seconds from the reception of a datum that reports a parameter that is under the threshold of health range. The response indeed corresponds to take contact with Emergency Service that takes charge of the requests and sends an ambulance.
Real time data available on server are necessary, for this reason Data4Help mobile application needs to have a worst-case latency under 30 ms that does not consider the time spent to reach the server over the network. This latency value only refers to the internal application process which starts from the reception of a datum and ends with its forwarding.
System availability must be at least 99.995% which means that no more than 26.30 minutes of downtime in a year are allowed.
Failure events cannot exceed three times over a year as reported in system reliability.

3.4 Design Contraints

3.4.1 Standards Compliance

Data4Help website will be developed taking in account Standards Compliance defined by the World Wide Web Consortium (W3C) with the purpose of enhancing interoperability among different browsers.

Data4Help website needs its compliance to be tested, before the release, submitting the website URL on the Validator Tool (validator.w3.org) and consulting errors reported: fully compliance (meaning zero errors on Validator Tool) its really hard to achieve and in most of the cases its not necessary, moreover there are cases in which its needed to break some of W3C rules to make something work in a specific desired way on every browser.

The website to release needs the result coming from Validator Tool to have at most 10 errors and indefinite number of warnings.

3.5 Software System Attributes

3.5.1 Reliability

Reliability is a parameter to evaluate the quality of a software (or hardware) system that can be measured by different reliability metrics (hardware metrics are not suitable for softwares):

for intermittent demands its useful the POFOD (probability of failure on demand) that is the likelihood the system will fail when a request is made, while ROCOF (rate of occurrence of failures) is used when the system has to process a large number of similar requests in a short time. This metric permits to select time units (such as row execution time, calendar time, number of transactions). Reliability is defined indeed as the measure of how long the system performs its intended function.

The demands rate for Data4Help system is likely to be unpredictable according to different options user can activate and the intensity of the interaction the user intends to have with the application or web page. For these reasons ROCOF metric is more appropriate to evaluate Data4Help system reliability: the time unit selected is the year and the occurrence of failure over this period must not exceed two times.

3.5.2 Availability

Availability is a parameter particularly relevant for continuously running systems: its a measure of the percentage of time the system is available for use and considers the restart and the repair time. Availability can be calculated as the Mean Time Between Failure (MTBF, uptime) divided by the sum of MTBF and the Mean Time To Repair (MTTR, downtime including restart).

Taking in account the fact that Data4Help operates in health services it has been selected a four-and-a-half-nine availability that means a percentage of 99.995, a downtime per year of 26.30 minutes, 2.19 minutes per month.

3.5.3 Security

Security of a system reflects the ability to guarantee confidentiality, integrity of data and authenticity over treats from both accidental and malicious events.

Data4Help authentication process (made from user identification and password) preserves confidentiality allowing users to access dedicated resources only: sensitive data such as passwords need to be encrypted implementing Hash-with-Salt or any stronger mechanism for encryption.

Data4Help uses TLS (transport layer security cryptographic protocol) to encrypt communications between the user and server(s).

Backups, kept both on cloud and on hard disks, need to be fully encrypted to prevent unauthorized access. When connections are established, HTTPS (HTTP + SSL certificate at application layer) is selected over HTTP.

Keeping data consistent over time is the concern behind Integrity: Data4Help avails itself of Web Application Firewalls (WAFs); to reduce human induced integrity errors, it should be implemented a detecting algorithm such as Damm or Luhn algorithm, while to ensure logic integrity a run-time sanity checks needs to be present (such as SQL CHECK constraints). The use of ZFS (or equivalent) should be taken in account to include an extensive protection against data corruption.

3.5.4 Maintainability

Maintainability is a measure of how easily and rapidly a software system can be corrected, improved and adapted to a new environment: its important for the system quality to be concerned about maintainability from the very beginning of the development, later adjustment can be very expensive.

To achieve this quality Data4Help software will avoid high dependencies in the code and use specifics Patterns wherever its appropriate; the development should be iterative and incremental (such as Agile Methodology), Waterfall approach could compromise high maintainability and shouldnt be applied.

3.5.5 Portability

The concern behind porting is to build a program executable and usable over different environments having effort to (re)adapt the software significantly inferior to the effort for new implementation.

The separation between the logic and the interface of the software, that needs to be done from the very beginning of development, is essential to ensure Data4Help to be portable.

Portability can also be affected by the choice of programming language and libraries: language portability its kind of a debated issue, java has a bunch of standard libraries and should be portable and distributed enough. Independence from hardware enhances Data4Help portability.

4. Formal Analysis Using Alloy

4.1 Data4Help

The first alloy model focus on the lifecycle of data requests performed by TPUs on IUs, on the access from TPUs to approved data and their being blocked, namely:

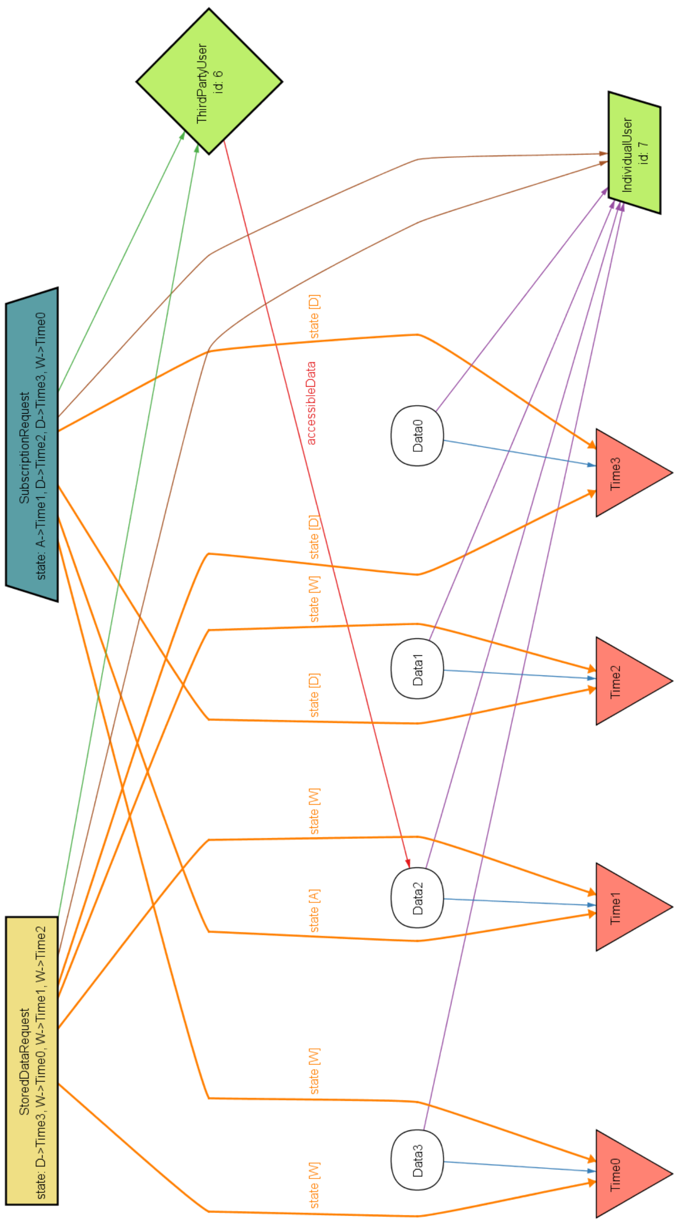
- a request must be created waiting for approval and must then go in a deny or approve state, after which it never changes state, unless it is a subscription request whose approval can be revocked, causing the request to go from an approval state ti a deny state;
- a TPU has access only to data for which exists either a stored data request approved after the data was collected or a subscription request approved at the time the data was produced
- a TPU cannot make a request for data of an individual user if it is blocked

To do so we considered a time frame for which the requests have the set of state they were in every instant. The IUs have instead the set of blocked TPUs; this is considered already filled with the TPUs that the IU blocked and never change in the time frame. The TPUs have a set of data it has access to, after the time frame.

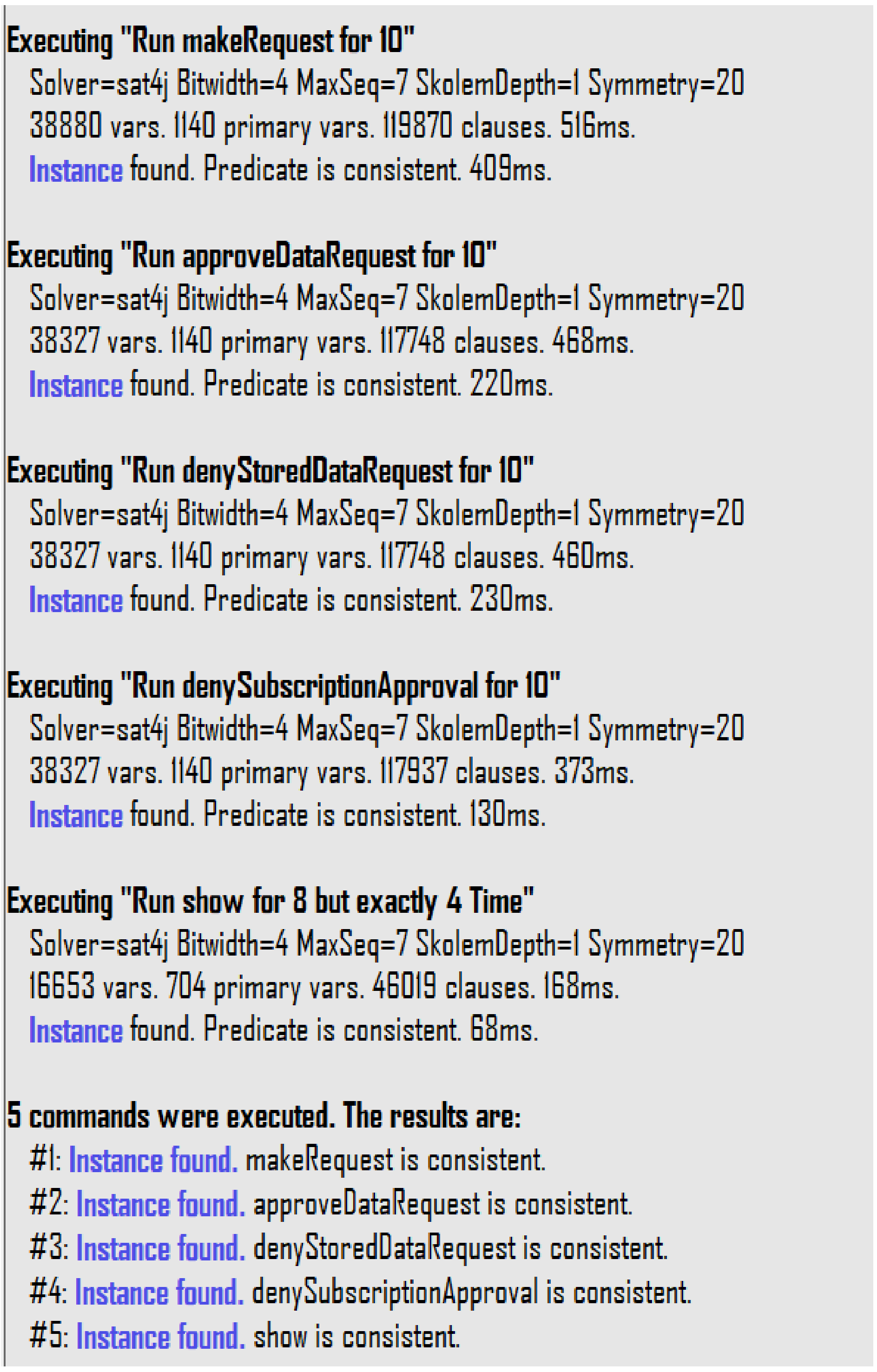
4.1.1 Alloy model

```
open util/time

/* Users
*****************************************************************/
abstract sig User{
  id: one Int
}
sig ThirdPartyUser extends User{
  accessibleData: set Data /*The data accessible from the TPUs is the data
      is has access to after the time frame.*/
}
sig IndividualUser extends User{
  blockedTPUs: set ThirdPartyUser /*The data accessible from the TPUs is
      the data is has access to after the time frame.*/
}
sig Data{
  user: one IndividualUser,
  time: one Time
}

/* Request
*****************************************************************/
abstract sig DataRequest{
  thirdPartyUser: one ThirdPartyUser,
  individualUser: one IndividualUser,
  stateSet: set RequestState -> Time /*The requests contain the set of
     state they were in the time frame considered.*/
}
sig StoredDataRequest extends DataRequest{}
sig SubscriptionRequest extends DataRequest{}

enum RequestState{W, A, D}
/*WAITING, APPROVED, DENYED*/


/* Unicity facts
*******************************************************************/
fact idUnique{
  no disjoint u1, u2: User | u1.id = u2.id
}

/*Data requests cycle facts
*******************************************************************/
fact dataRequestState{
  /*A state is defined for every time istant*/
  all dr: DataRequest | all t: Time | one state: RequestState |
     state in dr.stateSet.t
  /*Created in "Waiting"*/
  all dr: DataRequest | all t: Time | some t0: Time |
    gte[t, t0] && dr.stateSet.t0 = W
  /*When the request is "Waiting", it has never changed state*/
  all dr: DataRequest | all t1, t2: Time |
    (gte[t2, t1] && dr.stateSet.t2 = W) => (dr.stateSet.t1 = W)
  /*Once the request is "Denyed" it can't change state again*/
  all dr: DataRequest | all t1, t2: Time |
    (gte[t2, t1] && dr.stateSet.t1 = D) => (dr.stateSet.t2 = D)
}/*Allowed state sequences: {W -> D}{W -> A}{W -> A -> D}*/

fact storedDataRequest{
  /*Once the request is "Accepted" it can't change state again*/
  all sdr: StoredDataRequest | all t1, t2: Time |
    (gte[t2, t1] && sdr.stateSet.t1 = A) => (sdr.stateSet.t2 = A)
}/*Restricted allowed state sequences to: {W -> D}{W -> A}*/

/*Data access facts
*******************************************************************/
fact accessibleData{
  all tpu: ThirdPartyUser | all data: Data |
    /*Data regarding time t is accessible if*/
    (data in tpu.accessibleData) <=>
    /*exist a stored data request approved after time t*/
    ((some sdr: StoredDataRequest | some t: Time |
      (sdr.individualUser = data.user) &&
      (sdr.thirdPartyUser = tpu) &&
      (sdr.stateSet.t = A) &&
      (sdr.stateSet.(data.time) = W)
      )
    ||
    /*exist a subscription request approved at time t*/
    ((some sr: SubscriptionRequest |
        (sr.individualUser = data.user) &&
        (sr.thirdPartyUser = tpu) &&
        (sr.stateSet.(data.time) = A)
        )
     ))
}

/*For each individual user all data exists*/
fact completeData{
  all us: IndividualUser | all t: Time | one data: Data | data.user = us
     && data.time = t
}

/*Data request cycle pred
*********************************************************************/
pred makeRequest[iu: IndividualUser, tpu: ThirdPartyUser, t: Time]{
  //pre-conditions
  not isBlocked[iu, tpu]
  //post-conditions
  one dr: DataRequest | dr.thirdPartyUser = tpu && dr.individualUser = iu
     && dr.stateSet.(t.next) = W
}
pred isBlocked[iu: IndividualUser, tpu: ThirdPartyUser]{
  tpu in iu.blockedTPUs
}
pred approveDataRequest[tpu: ThirdPartyUser, dr: DataRequest, t: Time]{
  //pre-conditions
  dr.stateSet.t = W
  //post-conditions
  dr.stateSet.(t.next) = A
}
pred denyStoredDataRequest[tpu: ThirdPartyUser, sdr: StoredDataRequest, t:
    Time]{
  //pre-conditions
  sdr.stateSet.t = W
  //post-conditions
  sdr.stateSet.(t.next) = D
}
pred denySubscriptionApproval[tpu: ThirdPartyUser, sr:
    SubscriptionRequest, t: Time]{
  //pre-conditions
  not sr.stateSet.t = D
  //post-conditions
  sr.stateSet.(t.next) = D
}

pred show{
  #(ThirdPartyUser) = 1
  #(IndividualUser) = 2
  #(StoredDataRequest) = 2
  #(SubscriptionRequest) = 2
  /*At least one tpu has access to some data*/
  (some tpu: ThirdPartyUser | #(tpu.accessibleData) > 0)
  (some iu: IndividualUser | some tpu: ThirdPartyUser | some t: Time |
     makeRequest[iu, tpu, t])
  (some dr: DataRequest | some tpu: ThirdPartyUser | some t: Time |
     approveDataRequest[tpu, dr, t])
  (some sdr: StoredDataRequest | some tpu: ThirdPartyUser | some t: Time |
     denyStoredDataRequest[tpu, sdr, t])
  (some sr: SubscriptionRequest | some tpu: ThirdPartyUser | some t: Time
     | denySubscriptionApproval[tpu, sr, t])
}


/*Run
*******************************************************************/
run makeRequest for 10
run approveDataRequest for 10
run denyStoredDataRequest for 10
run denySubscriptionApproval for 10

run show for 8 but exactly 4 Time
```

4.1.2 Generated world



4.1.3 Alloy Result



```
Executing "Run makeRequest for 10"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   38880 vars. 1140 primary vars. 119870 clauses. 516ms.
   Instance found. Predicate is consistent. 409ms.

Executing "Run approveDataRequest for 10"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   38327 vars. 1140 primary vars. 117748 clauses. 468ms.
   Instance found. Predicate is consistent. 220ms.

Executing "Run denyStoredDataRequest for 10"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   38327 vars. 1140 primary vars. 117748 clauses. 460ms.
   Instance found. Predicate is consistent. 230ms.

Executing "Run denySubscriptionApproval for 10"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   38327 vars. 1140 primary vars. 117937 clauses. 373ms.
   Instance found. Predicate is consistent. 130ms.

Executing "Run show for 8 but exactly 4 Time"
   Solver=sat4j Bitwidth=4 MaxSeq=7 SkolemDepth=1 Symmetry=20
   16653 vars. 704 primary vars. 46019 clauses. 168ms.
   Instance found. Predicate is consistent. 68ms.

5 commands were executed. The results are:
   #1: Instance found. makeRequest is consistent.
   #2: Instance found. approveDataRequest is consistent.
   #3: Instance found. denyStoredDataRequest is consistent.
   #4: Instance found. denySubscriptionApproval is consistent.
   #5: Instance found. show is consistent.
```

4.2 AutomatedSOS

The second alloy model focus on the lifecycle of an emergency call:

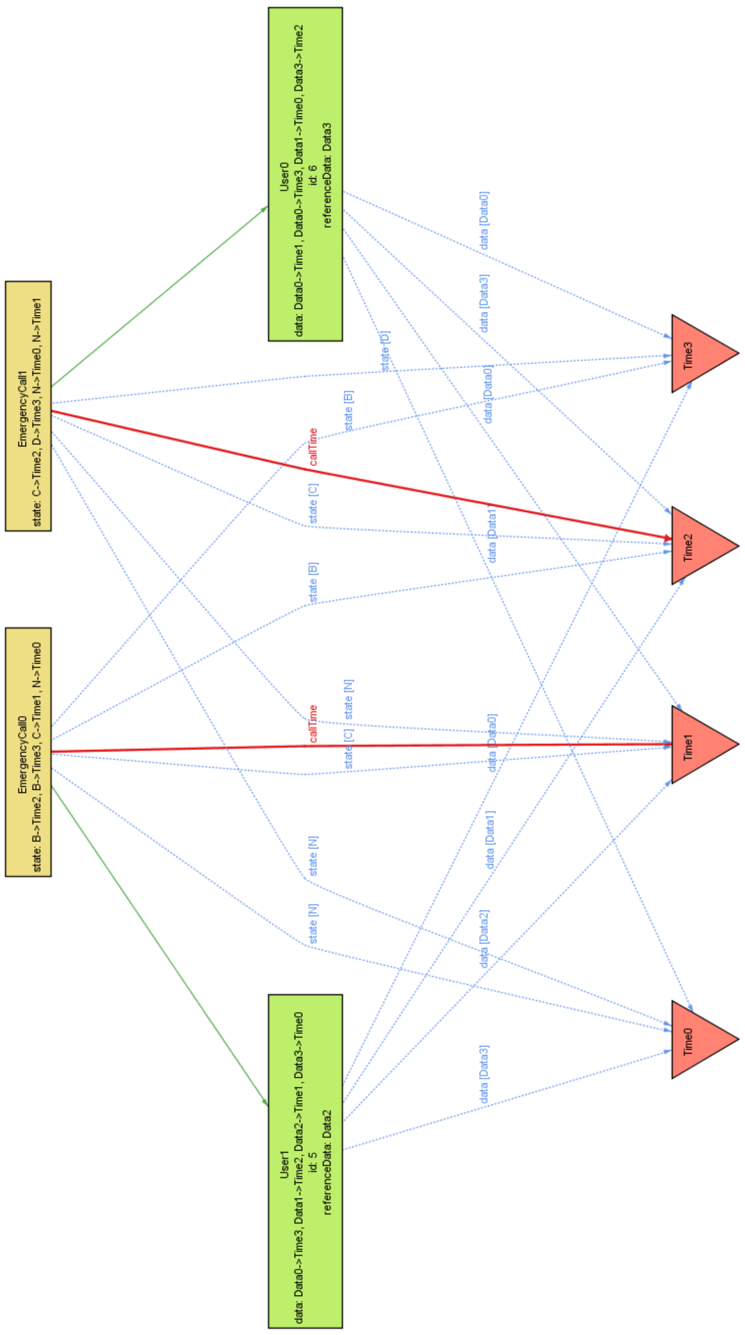
- a call must end after it has contacted the ES or after it was blocked by the IU
- for every time interval in which the health was critical exactly one call must have been made

For simplicity sake, this model considers that an abulance is called every time the status drops ti critical.

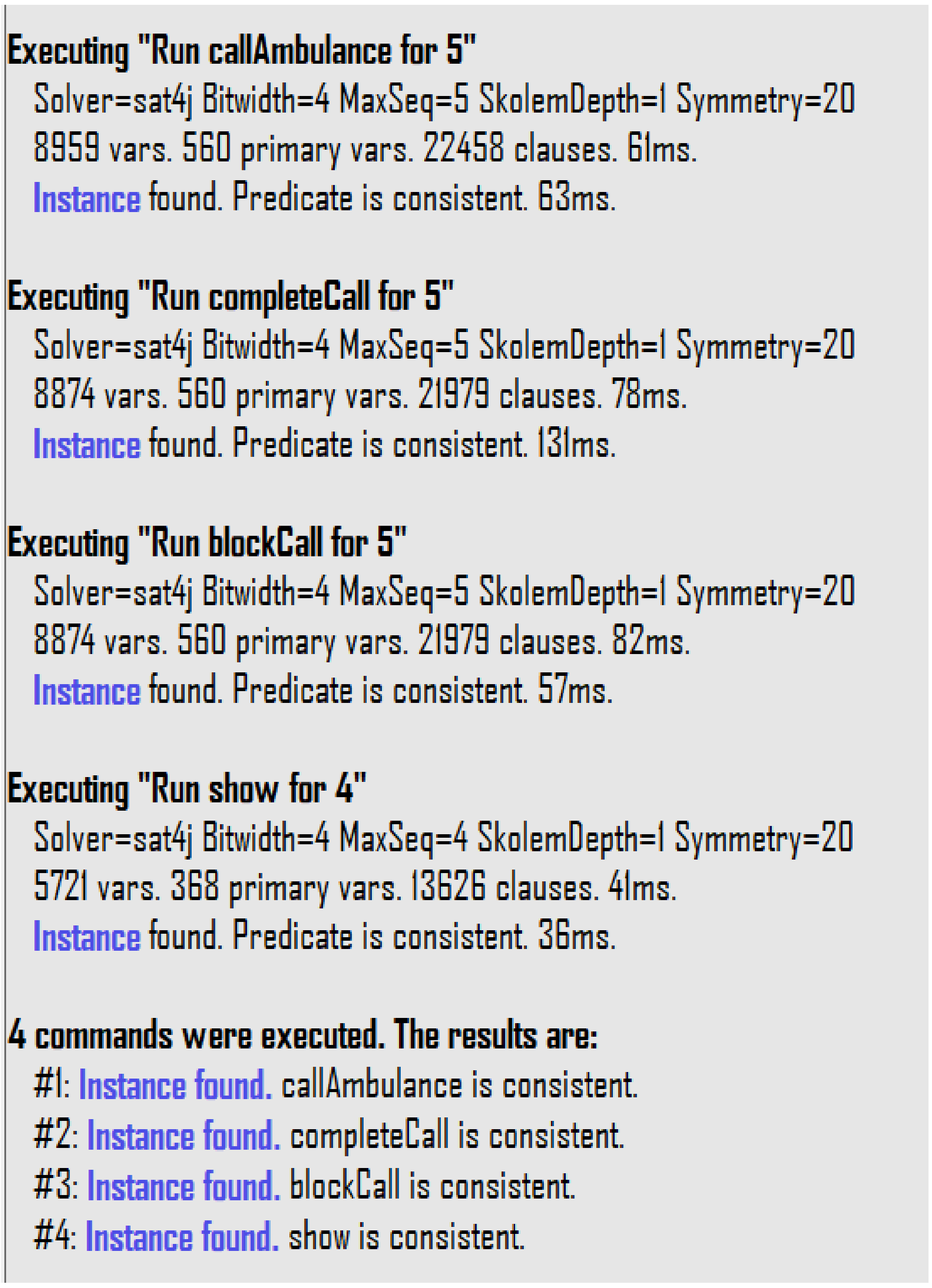
4.2.1 Alloy model

```
open util/time



/* Signatures
*****************************************************************************/
abstract sig User{
  id: one Int,
  locationSet: set Location -> Time,
  dataSet: set Data -> Time,
  referenceData: set Data /*Set of data for critical health condition*/
}{
  #(referenceData) > 0
  all t: Time | one data: Data | dataSet.t = data
}
sig Data{}
sig Location{}

sig EmergencyCall{
  user: one User,
  location: one Location,
  callTime: one Time,
  stateSet: set CallState one -> Time
}{
  location = user.locationSet.callTime
  all t: Time | one cs: CallState | stateSet.t = cs
}
enum CallState{N, C, B, D}
//NOT_STARTED, CALLING, BLOCKED, DONE

/* Facts
*****************************************************************************/
fact idUnique{
  no disjoint u1, u2: User | u1.id = u2.id
}

fact callStates{
  /*A state is defined for every time istant after callTime*/
  all ec: EmergencyCall | all t: Time | one state: CallState |
     state in ec.stateSet.t
  /*Created in "Calling"*/
  all ec: EmergencyCall |
     (ec.stateSet.(ec.callTime) = C) && (no t: Time | (lt[t, ec.callTime])
        && (ec.stateSet.t = C))
  /*When the call is "not started", it has never changed state*/
  all ec: EmergencyCall | all t1, t2: Time |
    (gte[t2, t1] && ec.stateSet.t2 = N) => (ec.stateSet.t1 = N)
  /*Once the call is "Blocked" it can't change state again*/
   all ec: EmergencyCall | all t1, t2: Time |
    (gte[t2, t1] && ec.stateSet.t1 = B) => (ec.stateSet.t2 = B)
  /*Once the call is "Done" it can't change state again*/
   all ec: EmergencyCall | all t1, t2: Time |
    (gte[t2, t1] && ec.stateSet.t1 = D) => (ec.stateSet.t2 = D)

}/*Allowed state sequences: {N -> C -> B}{N -> C -> D}*/

/*For every instant in wich the user health went critical a call has been
    made when health dropped*/
fact whenCriticalThereIsACall{
  all us: User | all t: Time |
  (us.dataSet.t in us.referenceData) => (
  one ec: EmergencyCall | all t1: Time |
    (ec.user = us) &&
    (lte[ec.callTime, t]) &&
    ((gte[t1, ec.callTime] && lte[t1, t]) => (us.dataSet.t1 in
        us.referenceData))
  )
}

/*No calls are made when health is not critical*/
fact {
  all ec: EmergencyCall | one us: User |
  (ec.user = us) &&
  (us.dataSet.(ec.callTime) in us.referenceData)
}

/* Predicates
*********************************************************************/
pred callAmbulance[us: User, t: Time]{
  //pre-conditions
  //post-conditions
  one ec: EmergencyCall |
    (ec.user = us) &&
    (ec.callTime = t.next) &&
    (ec.stateSet.(t.next) = C)
}

pred completeCall[ec: EmergencyCall, t: Time]{
  //pre-conditions
  ec.stateSet.t = C
  //post-conditions
  ec.stateSet.(t.next) = D
}

pred blockCall[ec: EmergencyCall, t: Time]{
  //pre-conditions
  ec.stateSet.t = C
  //post-conditions
  ec.stateSet.(t.next) = B
}

pred show {
    #(User) = 2
   #(EmergencyCall) = 2
   (all us: User | some t: Time | callAmbulance[us, t])
   (some ec: EmergencyCall | some t: Time | completeCall[ec, t])
   (some ec: EmergencyCall | some t: Time | blockCall[ec, t])
}

/* Run
*******************************************************************/
run callAmbulance for 5
run completeCall for 5
run blockCall for 5

run show for 5
```

4.2.2 Generated world



4.2.3 Alloy Result



```
Executing "Run callAmbulance for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8959 vars. 560 primary vars. 22458 clauses. 61ms.
   Instance found. Predicate is consistent. 63ms.

Executing "Run completeCall for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8874 vars. 560 primary vars. 21979 clauses. 78ms.
   Instance found. Predicate is consistent. 131ms.

Executing "Run blockCall for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   8874 vars. 560 primary vars. 21979 clauses. 82ms.
   Instance found. Predicate is consistent. 57ms.

Executing "Run show for 4"
   Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20
   5721 vars. 368 primary vars. 13626 clauses. 41ms.
   Instance found. Predicate is consistent. 36ms.

4 commands were executed. The results are:
   #1: Instance found. callAmbulance is consistent.
   #2: Instance found. completeCall is consistent.
   #3: Instance found. blockCall is consistent.
   #4: Instance found. show is consistent.
```

4.3 Track4Run

The third alloy model focus on the lifecycle of a run and the restriction on participants:

- every organized run must either be canceled before they start or successfuly end;
- IUs can enroll to participate or spectate only during specific state of the run, respectively when the run has been organized but not started and when the run has started

For simplicity sake, this model considers that an abulance is called every time the status drops ti critical.

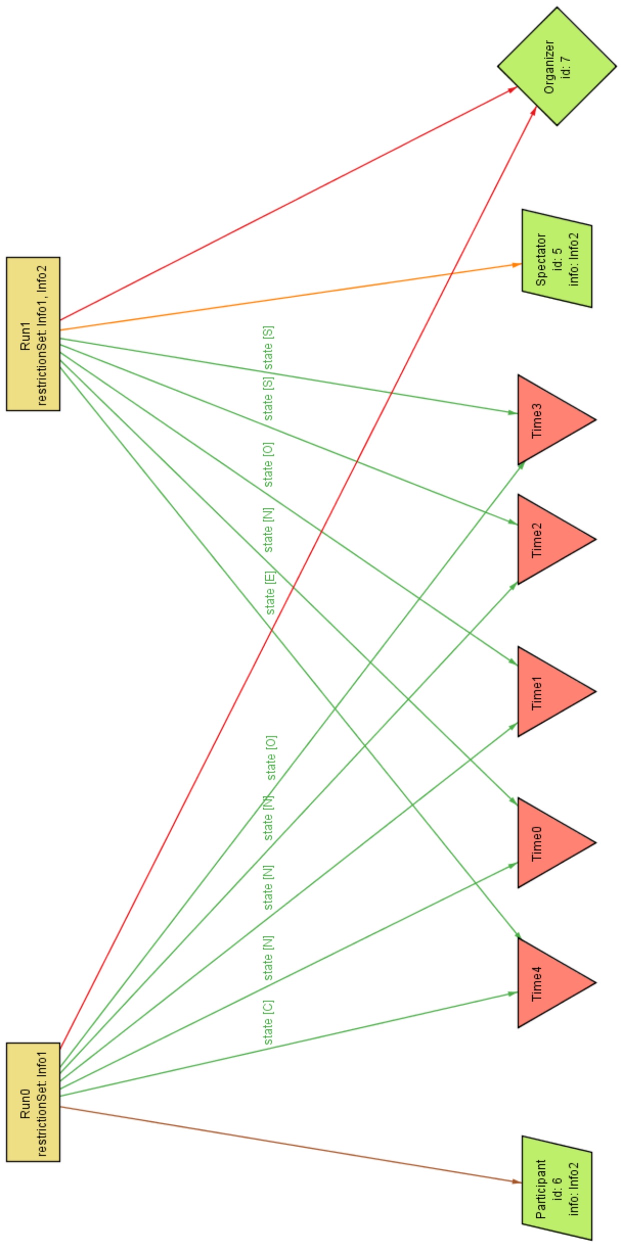
Alloy model

```
open util/time

/* Users
*******************************************************************/
abstract sig User{
  id: one Int
}
abstract sig ThirdPartyUser extends User{}
abstract sig IndividualUser extends User{
  info: one Info
}
sig Info{}

sig Organizer extends ThirdPartyUser{}

sig Participant extends IndividualUser{}
sig Spectator extends IndividualUser{}

sig Run{
  organizer: one Organizer,
  stateSet: RunState one -> Time, /*state of the run for every instant in
     the time frame considered*/
  /*Users who spectated the run*/
  spectatorSet: set Spectator,
  /*Users who are participants at the end of the time frame considered*/
  participantSet: set Participant,
  /*These two set represent the number of participants and spactators for
     every instant in the time frame considered*/
  participantNum: set Int one -> Time,
  spectatorNum: set Int one -> Time,
  restrictionSet: set Info //restriction are modeled as a set of not
     allowed info
}{
  all t: Time | one i: Int | participantNum.t = i
  all t: Time | one i: Int | spectatorNum.t = i
  /*at the end of the time frame the number of participant is equal to
     participantNum*/
  all t: Time | (no t0: Time | gt[t0, t]) => participantNum.t =
     #(participantSet)
}
enum RunState{N, O, S, E, C}
//NOT_CREATED, ORGANIZED, STARTED, ENDED, CANCELD

/* Unicity facts
```

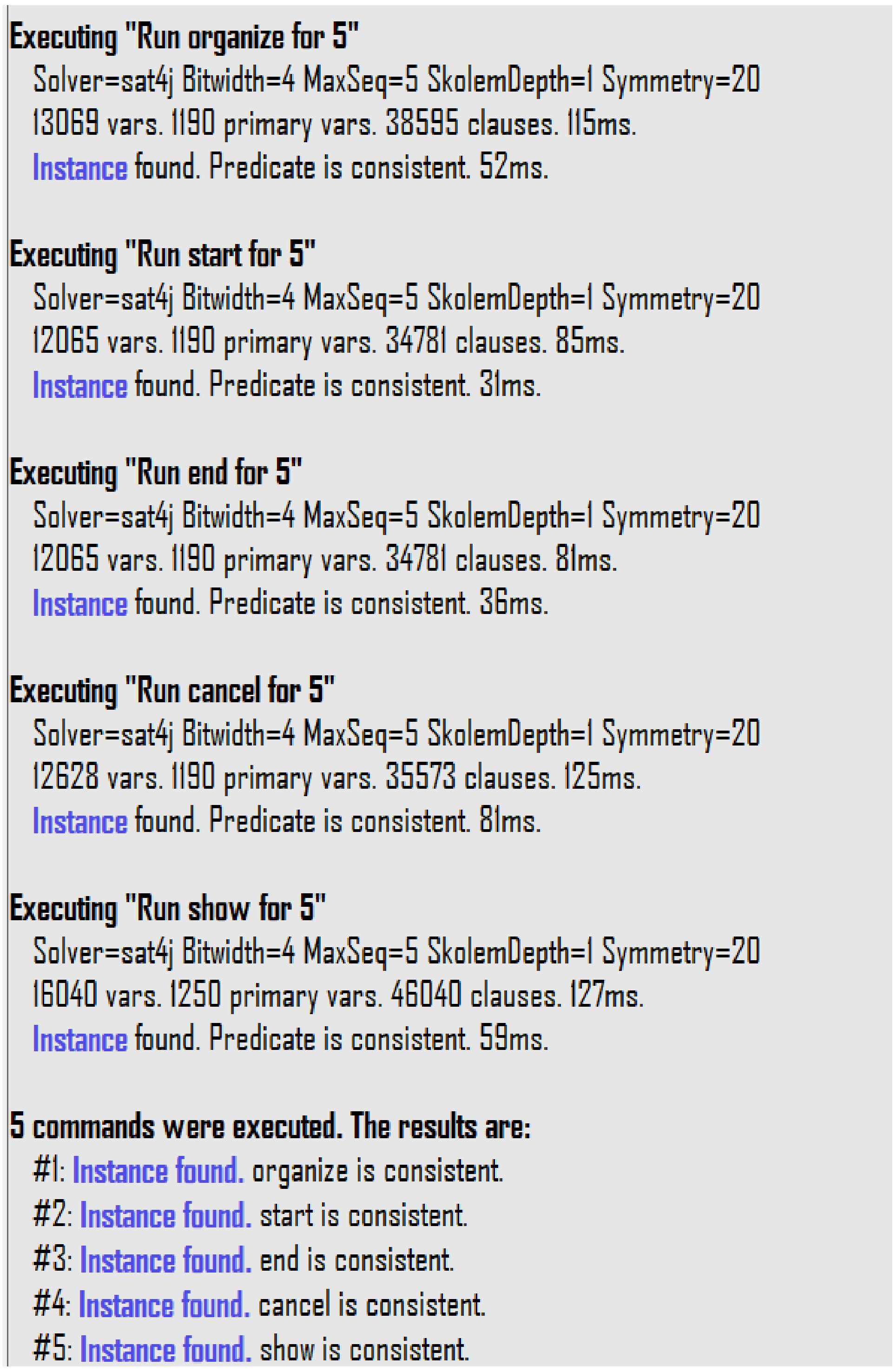
```
*******************************************************************/
fact idUnique{
  no disjoint u1, u2: User | u1.id = u2.id
}

/* Run states facts
*******************************************************************/
fact runStates{
  /*A state is defined for every time istant after callTime*/
  all r: Run | all t: Time | one state: RunState |
     state in r.stateSet.t
  /*A run must have been in the "Non created" state at least once*/
  all r: Run | some t: Time | r.stateSet.t = N
  /*When the run is "Not created" yet, it has never changed state */
  all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t2 = N) => (r.stateSet.t1 = N)
  /*If "Organized" has always been ''Not started'' or ''Organized''*/
  all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t2 = O) => (r.stateSet.t1 = N ||
       r.stateSet.t1 = O)
  /*If "Started" will "End"*/
  all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t1 = S) => (r.stateSet.t1 = S ||
       r.stateSet.t1 = E)
  /*Once the run is "Ended" it can't change state again*/
  all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t1 = E) => (r.stateSet.t2 = E)
  /*If "Canceled" has never "Started"*/
  all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t2 = C) => (not r.stateSet.t1 = S)
  /*Once the run is "Canceled" it can't change state again*/
   all r: Run | all t1, t2: Time |
    (gte[t2, t1] && r.stateSet.t1 = C) => (r.stateSet.t2 = C)

}/*Allowed state sequences: {N -> O -> S -> E}{N -> O -> C}*/


/* Joining facts
*******************************************************************/
fact openEnroll{
  /*If run is not in the "Organize" state none can enroll*/
  all r: Run | all t1, t2: Time |
  (not(r.stateSet.t1 = N) && not (r.stateSet.t1 = O) && not(r.stateSet.t2
     = N) && not (r.stateSet.t2 = O))
  =>
  (r.participantNum.t1 = r.participantNum.t2)

  /*If run is in the "Not Started" nobody enorlled*/
  all r: Run | all t: Time |
  (r.stateSet.t = N)
  =>
  (r.participantNum.t = 0)

  /*If run is in not in the "Started" nobody is spectating*/
  all r: Run | all t: Time |
  (not (r.stateSet.t = S))
  =>
  (r.spectatorNum.t = 0)
}

/* Run state pred
***********************************************************************/
pred organize[o: Organizer, t: Time]{
  one r: Run |
    (r.stateSet.t = N) &&
    (r.organizer = o) &&

    (r.stateSet.(t.next) = O) &&
    (r.participantNum.(t.next) = 0) &&
    (r.spectatorNum.(t.next) = 0)
}

pred start[r: Run, t: Time]{
  //pre-conditions
  (r.stateSet.t = O)
  //post-conditions
  (r.stateSet.(t.next) = S)
}

pred end[r: Run, t: Time]{
  //pre-conditions
  (r.stateSet.t = S)
  //post-conditions
  (r.stateSet.(t.next) = E)
}

pred cancel[r: Run, t: Time]{
  //pre-conditions
  (r.stateSet.t = O)
  //post-conditions
  (r.stateSet.(t.next) = C) &&
  (r.spectatorNum.(t.next) = 0)
}

/* Enroll Run pred
***********************************************************************/
pred enroll[p: Participant, r: Run, t: Time]{
  not (p.info in r.restrictionSet)
  r.participantNum.(t.next) = r.participantNum.t + 1
  p in r.participantSet
}

pred spectate[s: Spectator, r: Run, t: Time]{
  r.spectatorNum.(t.next) = r.spectatorNum.t + 1
  s in r.spectatorSet
}

/* Show
***********************************************************************/
pred show{
  #(ThirdPartyUser) = 1
  #(IndividualUser) = 2
  #(Info) = 3
  #(Run) = 2
```

```
   (some o: Organizer | some t: Time | organize[o, t])
   (some r: Run | some t: Time | start[r, t])
   (some r: Run | some t: Time | end[r, t])
   (some r: Run | some t: Time | cancel[r, t])
   (some r: Run | some t: Time | some p: Participant | enroll[p, r, t])
   (some r: Run | some t: Time | some s: Spectator | spectate[s, r, t])
}


/* Run
*******************************************************************/
run organize for 5
run start for 5
run end for 5
run cancel for 5

run show for 5
```

4.3.1 Generated world



4.3.2 Alloy Result



```
Executing "Run organize for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   13069 vars. 1190 primary vars. 38595 clauses. 115ms.
   Instance found. Predicate is consistent. 52ms.

Executing "Run start for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   12065 vars. 1190 primary vars. 34781 clauses. 85ms.
   Instance found. Predicate is consistent. 31ms.

Executing "Run end for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   12065 vars. 1190 primary vars. 34781 clauses. 81ms.
   Instance found. Predicate is consistent. 36ms.

Executing "Run cancel for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   12628 vars. 1190 primary vars. 35573 clauses. 125ms.
   Instance found. Predicate is consistent. 81ms.

Executing "Run show for 5"
   Solver=sat4j Bitwidth=4 MaxSeq=5 SkolemDepth=1 Symmetry=20
   16040 vars. 1250 primary vars. 46040 clauses. 127ms.
   Instance found. Predicate is consistent. 59ms.

5 commands were executed. The results are:
   #1: Instance found. organize is consistent.
   #2: Instance found. start is consistent.
   #3: Instance found. end is consistent.
   #4: Instance found. cancel is consistent.
   #5: Instance found. show is consistent.
```

5. Effort Spent