Politecnico di Milano AA 2021-2022

Computer Science and Engineering

Software Engineering 2 Project

RASD

Requirement Analysis and Specification Document

version 1.0 - 10/11/2021

Blasucci Camilla - 907559

Buono Salvatore - 907445

1 INTRODUCTION

1.1 Purpose

This Requirement Analysis and Specification Document aims to present DREAM in detail.
It contains the process and the requirements to achieve a new food system where, through new technologies, the collection of data and the decision of further actions are made easier.
This document presents a detailed description of functional and non-functional requirements necessary to develop the system.

1.2 Goals

In the region of Telangana in India, COVID-19 has made worse the already noticeable problem regarding the food supply chains, highlighting the weaknesses of the marginalized communities and small holder farmers. This also brought up the importance of building a better food system with the assistance of the newest IT technologies.
Here are listed the main goals:

[G1] Allow a farmer to have a personal area

[G2] Allow a registered farmer to insert data about daily crops

[G3] Allow a registered farmer to publish and visualize contents on the forum

[G4] Allow a policy maker to view charts on farmers' performances

[G5] Allow a registered farmer to check the forecast

[G6] Allow a policy maker to alert a farmer of winning a good production prize

[G7] Allow a farmer to compare his work in a time frame

[G8] Allow a registered farmer to ask for help

1.3 Scope

Dream is a user-friendly application that is supposed to help the entire food supply chain, giving to all the characters taken in account an instrument to control and maximize the results in the agriculture.
Farmers can check their crops' development in time through charts present on their profile, insert the daily changes of the harvest, control the parameters of their soil and read some personalized suggestions to improve their work.
They can also find a forum to write their doubts or look through the different topics and comments.
The relationship between the farmers and the agronomists is improved through an option to ask for help easily and chat.
Policy makers can also take advantage of the application to control easily all the farmers and their work.
They can see for each farmer his data or look at overall charts for different topics.

Through the charts they can choose to whom give different rewards and show them in farmers' application.
They can also control the work of the agronomists thanks to charts and collections of data present on the app.
They have access to the forum like the farmers and both can access the forecast clicking a button on the homepage.
This application makes easier the interaction between the agronomical workers on the entire land and helps to make more efficient the overall farm process.
Here are listed the different phenomena:

World phenomena

W1. Farmer decides to look at the forecast
W2. Farmer decides to sow a particular crop
W3. Farmer harvests some crop
W4. Farmer finds crop damaged and wants help
W5. Policy maker wants to find out best farmer to give a reward
W6. Farmer has a doubt he wants to solve
W7. Farmer wants to know if he got better in the last three months
W8. Farmer wants to check the sensors' info
W9. Policy maker wants to see the results of the farmers in his area

Shared phenomena

S1. Farmer/ policy maker logins on the app
S2. Farmer/policy maker registers on the app
S3. Confirmation account validated by the system
S4. Farmer asks for help
S5. Farmer/policy maker checks forecast
S6. Policy maker looks at the chart
S7. Policy maker sends reward
S8. Farmer writes on the forum
S9. Farmer reads the forum
S10. Farmer looks at his data
S11. Farmer registers crop's data
S12. Farmer receives a prize notification

Machine phenomena

M1. Registration of data on the profile
M2. Creation of the different charts and graphs
M3. Update forecast data
M4. Notification sending for bad weather
M5. Notification sending for prize winning alert
M6. Post comments on forum
M7. Update sensors' data

1.4 Definitions, Abbreviations, Acronyms

Definitions

- **Farmer –** any Telengana farmer
- **Policy Maker –** any Telengana policy maker
- **Agronomist –** any Telengana agronomist
- **Crop –** term used to generalize the crop and the fields of a farmer
- **System –** the entire hardware and software entities that form the service. Also called app, application, DREAM

Acronyms

- **TSDPS -** Automatic Weather Station
- **DREAM -** Data-dRiven PrEdictive FArMing in Telangana
- **TCP –** Transmission Control Protocol

Abbreviations

- **G -** Goal
- **W -** World phenomena
- **M -** Machine phenomena
- **S -** Shared phenomena
- **D -** Dependency
- **R –** Requirement

1.5 Revision history

11th November 2021 - Version 1.0 started
23th December- Version 1.0 finished

1.6 Reference Documents

- Software Engineering 2- year 2021- prof. Di Nitto Elisabetta course slide
- Alloy documentation: https://alloytools.org
- Specification document resources: https://www.tsdps.telangana.gov.in/aws.jsp
- Specification document: "R&DD Assignment A.Y. 2021-2022"

1.7 Document Structure

Section 1

This section contains a view of the context and a brief description of the project to develop.
Also, an initial analysis of the different world phenomena can be found and finally some information on the glossary used and the references that helped writing the RASD.

Section 2

Here there are some examples of different possible scenarios with the actors involved in the project.
The scenarios describe both more common and problematic situations.
The different actors and other elements are connected in a class diagram, in an initial description of the development of the project.
There is also a list of the main requirements and the users' characteristics.
Last the domain assumptions can be found here.

Section 3

The most specific part of the RASD, where there are all the details of the development and all the requirements. It underlines the software and hardware interfaces and design constraint. It also highlights the software system attributes

Section 4

This section contains presentation of what drives the formal modeling activity, a description of the model and it also presents some worlds obtained by the model to prove the correctness.

Section 5

Here it is present the division of hours put in the creation of this document divide by participants.

Section 6

All the references used in the RASD can be found here.

2 OVERALL DESCRIPTION

2.1 Product perspective

In order to explain the different functionalities of the system, here are described different scenarios where in each one is underlined the interaction between the system and the users.

2.1.1 Normal use farmer – login and crop insertion

User initially arrives at his field in the morning and connect with smartphone to the application system. Insert his credentials and wait for validation. After that, he checks the status of the ground (level of humidity) and if he publishes something in the forum, check if someone has answered to his question. There is no suggestion from the agronomist, so he can continue to work with the current cultivation.
After he finished to work, he can upload new information about the cultivation in the system (if there is not any issue, he will simply insert info about the daily crops). The system confirms that the information is successfully uploaded, finally he can close the application and go back home.

2.1.2 Storm case – aftermath, help

In the territory of a farmer there is a bad storm, and the crop is in bad conditions, the farmer needs some help, so he decides to inform an agronomist of the situation.
He opens the web app of DREAM and connects on their account. Once in, he presses the assistance button and immediately an alert is sent to the assistant of their zone.
The assistant receives the alert notification, so he opens the app on his device, logs in, enters the notification panel, and confirm the visualization of the notification and a chat is open with the farmer.

2.1.3 Chart overview and fund reward (suggestions request)

A policy maker decides to see the chart of the farmers of their zone to give to the best farmers some important suggestions.
He opens the DREAM app on his phone, logs in, and on the menu, he chooses the chart button.
The total chart is presented to the policy maker.
He clicks on the first farmer, and the profile of the farmer becomes visible.
The policy maker decides that this farmer is the one that will receive a fund to invest in his crop.
On the profile of the farmer, the policy maker clicks on the GIVE PRIZE.
A notice of a new fund received is sent to the farmer, so he opens the app on his device, logs in and a pop up with the request for new suggestions useful for the other farmers is shown on his screen.
The farmer decides to leave a suggestion, so he clicks okay on the chat, the forum opens in the suggestion topic, he writes his comment and sends it.
The comment is published on the forum.
The farmer closes his app and goes back to work.

2.1.4 Storm case – prevention

The forecast reports an imminent storm on the territory, the app sees the alarm through the frequently visits it automatically does on the forecast websites.
The app sends an alarm notification to all the farmers registered in that zone.
The farmer receives the notification, so he opens the app to see the details, he logs in and click on the notification.
On the notification he sees the forecast and clicks on a link that brings him directly to the suggestion topic of the forum.
He reads some comments to prepare his cultivation properly, to avoid any damage during the storm.
Once he finishes reading, he closes the app.

2.1.5 Checks on data after agronomist intervention

A policy maker wants to see if after the visit of an agronomist, the crop of the farmer has got better.
He logs in in the app and, through a research tab, finds the profile of the farmer.
He clicks on the profile and on the new page he can find all the data and a graph for each parameter of the farmer's crop.
Now he can see and analyze if the intervention has brought any good result to the crop.
Once he has gone through all the data he needs, he closes the app.

2.2 Product functions

The software can be divided in five different abstract components:

1. The farmer web application (FWA): it's a web site that allows farmers to access at his profile and to see all the information about the current crops, forecast, forum and his current position in the chart.
2. The farmer mobile application (FMA): it's an app that should be installed on farmers' smartphone and has the same functionalities of FWA.
3. The policy maker mobile application (PMA): it's an app that should be installed on Policy makers' smartphone and has the same functionalities of PWA.
4. The policy maker web application (PWA): it's a web portal where Policy makers are allowed to login, see local chart, visualize all data about the farmer of interest and visualize the progressions of the farmers.
5. The forecast connection management system (FMS): it's a system based on website which provide updated forecast from TSDPS for a certain zone and send notifications to farmers in order to notify them for possible meteorological adverse events.

2.3 User characteristics

The software is intended for farmers and policy makers, and they do not need any previous knowledge to be able to use the software on their devices.

2.4 Assumptions, Dependencies, Domain

[D1] User has always internet connection available.

[D2] User has always a device (computer or smartphone) working 24/7.

[D3] All the farmers log into DREAM at least once a day to register their crop.

[D4] All the sensors work properly in any situation.

[D5] The website with the forecast is always updated and reachable online.

[D6] All the policy makers, when on their turn, check their DREAM notifications periodically (more than 2 times per day).

[D7] Each zone is always covered by at least one active policy maker.

[D8] The policy makers are always updated on the interventions of the agronomists on the territory and know all the helps and suggestions given to the farmers.

[D9] Daily ranking with history is always accessible by policy makers.

[D10] A farmer inserts always valid data in the daily crop data.

[D11] A farmer, when writing in the forum, posts the comment in the right section. No check on the rightness of the post's location is needed.

3 SPECIFIC REQUIREMENTS

3.1 External Interface Requirements

In this section there is a description of the interaction between DREAM system and users (policy maker and farmers) and external systems (internet browser).

3.1.1. User Interfaces

User interfaces must be designed to clarify the useability of the software. Furthermore, some mock-ups of the main features available for the policy makers and farmers by means web and mobile interface will be illustrated (they should be considered just a draft).

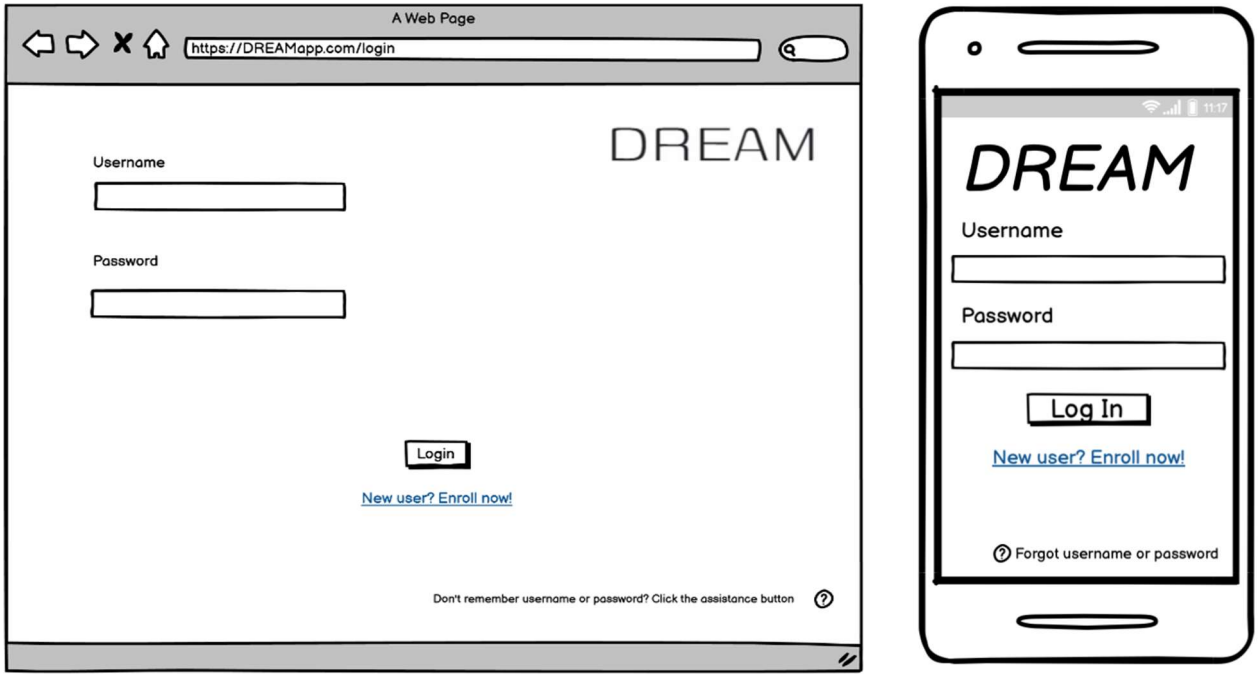


Figure 1:Login Page

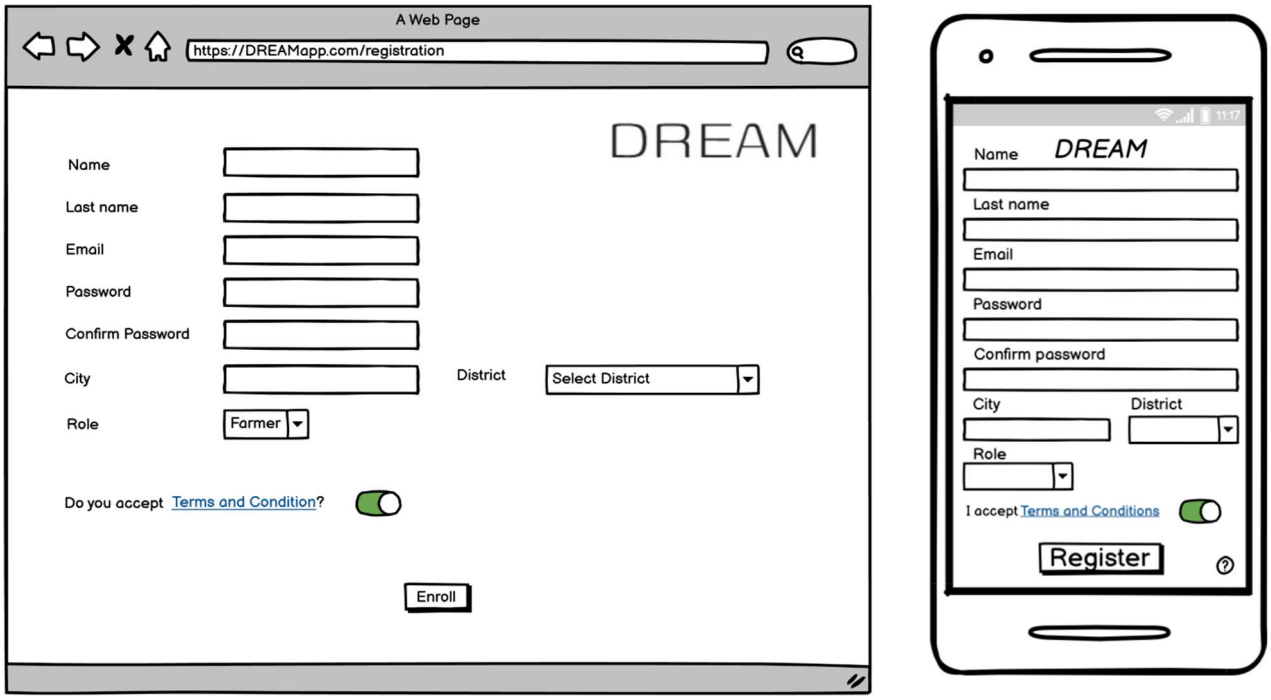


Figure 2:Enroll Page

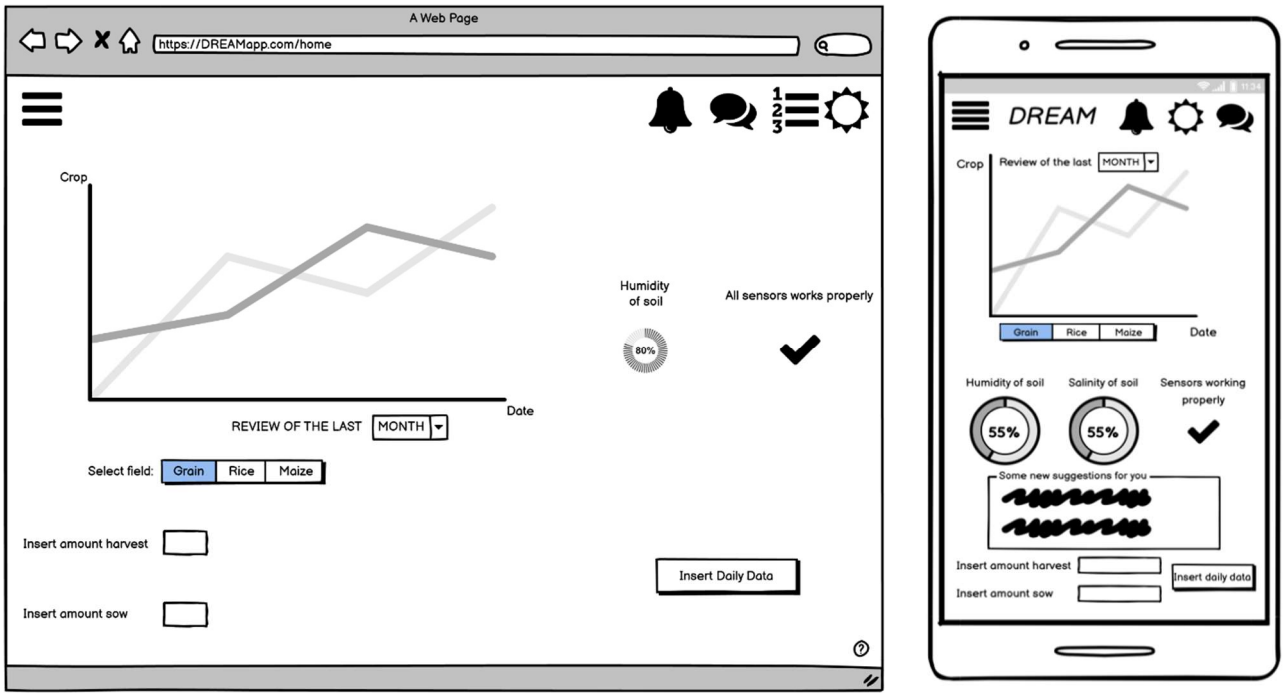


Figure 3:Home Page

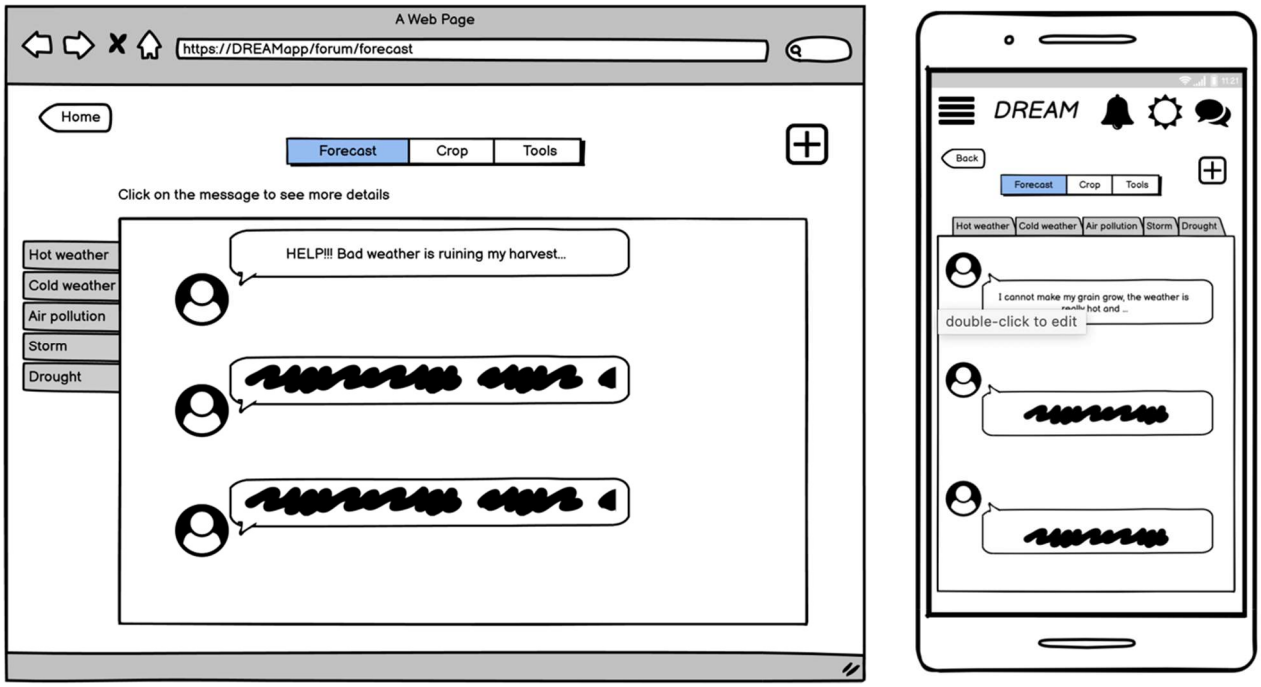


Figure 4:Forum

3.1.2. Hardware Interfaces

DREAM system interfaces with the following hardware systems:

- Smartphone with 3G (or superior) module to be able to interact with the system if there is no presence of Wi-fi available.
- Space available in storage for mobile app.
- PC/Laptop which support minimum requirements of the modern browsers.
- Sensors planted in the soil which supply data to the system.

3.1.3. Software Interfaces

DREAMS needs the following interfaces to work well on the software side:

- To provide the information related to the approach time, the system should interface with an external APIs offering data from sensors.
- The system should interact with forecast provided by TSDPS web service, which supplies notification alert if there is any meteorological adverse event arriving.

In the next page is displayed a general UML with the elements of the software.

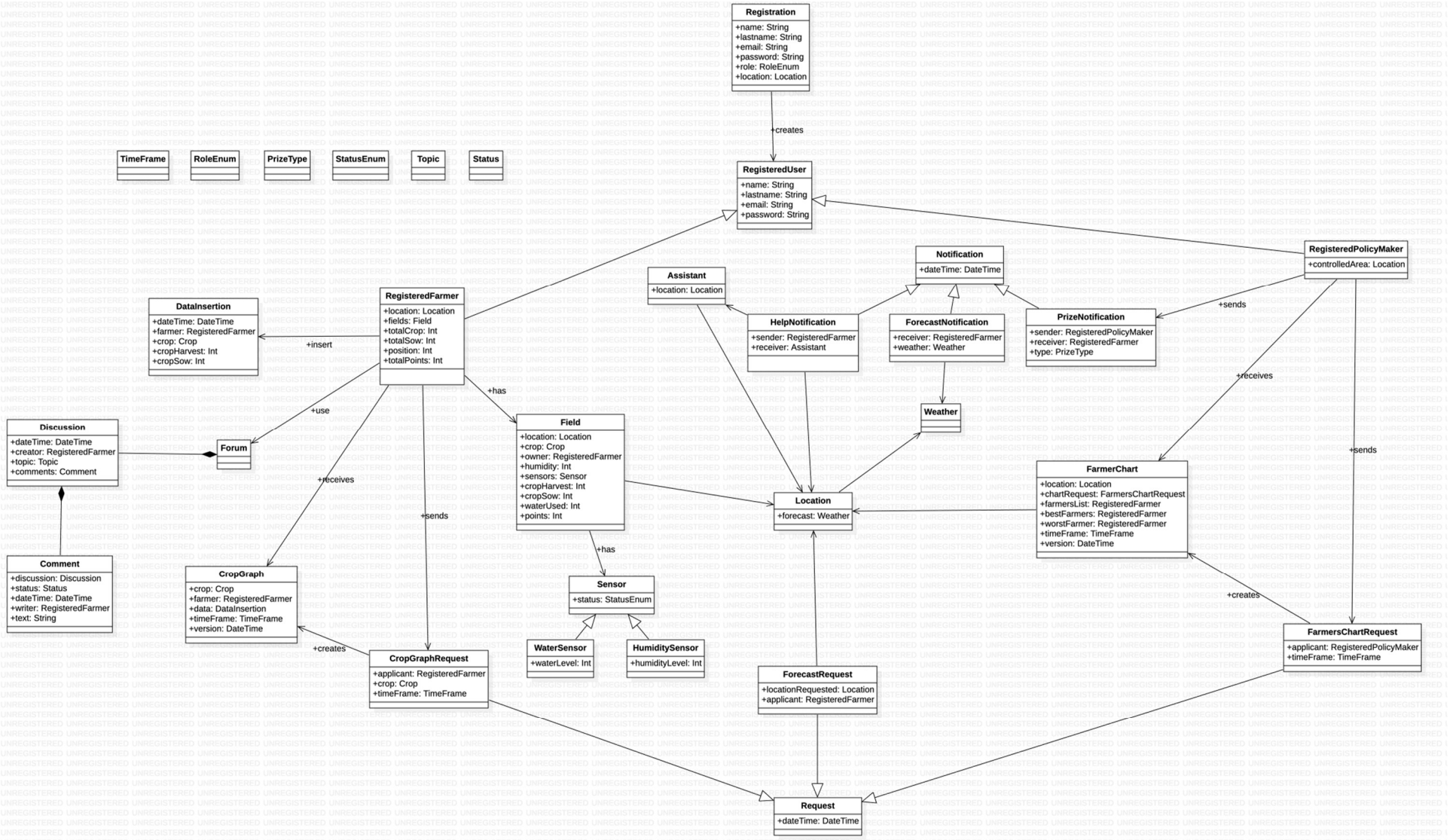


Figure 5: UML Class Diagram

3.1.4. Communication Interfaces

To interact properly, DREAM system must have access to the Internet and all devices involved employ TCP/IP protocol to communicate.

3.2 Functional Requirements

In this section are portrayed the requirements for DREAM system. Each requirement is mapped with a specific goal it is intended to satisfy.

3.2.1 Mapping of Requirement with Goals

[G1] Allow a farmer to have a personal area

[R1.1] DREAM must provide registration form where user inserts his personal data and choose a password.

[R1.2] DREAM must verify if the user is not already registered, if not DREAM will send a confirmation mail, otherwise invite them to login.

[R1.3] DREAM must provide login form in which they have to insert username and password.

[R1.4] DREAM must verify if the username and password are correct, if so DREAM shall show home page, otherwise report the login error.

[R1.5] DREAM provides an assistance procedure to recover password.

[G2] Allow a registered farmer to insert data about daily crops

[R2.1] DREAM must provide insertion data form, where user once a day should provide information about crops.

[R2.2] DREAM must provide one form for each type of crop in the farmer's field.

[G3] Allow a registered farmer to publish and visualize contents on the forum

[R3.1] DREAM must provide a form where farmers are allowed to publish contents such as suggestion, comments and help requests.

[R3.2] DREAM users can create discussion with other farmers making explicit with an '#' the type of argument.

[R3.3] DREAM users can filter contents to visualize in base of their type: forecast, crop and tool.

[G4] Allow a policy maker to visualize chart on farmers' performance

[R4.1] DREAM must identify farmers users who are performing well or badly and create a chart relying on data obtained by sensors planted in the soil.

[R4.2] DREAM must show in farmers' home page their current position in the global chart.

[R4.3] DREAM must provide to policy maker to visualize the overall chart and mark the farmers who have been hit by a meteorological adverse event (storm, strong wind…).

[R4.4] DREAM must allow to policy maker to interact with farmers and chat with those who need help.

[G5] Allow a registered farmer to check the forecast

[R5.1] DREAM must provide an interface where the farmer can check the forecast

[R5.2] DREAM must have a secure and lasting connection with the forecast website

[R5.3] DREAM must send a notification to the farmers in case of imminent meteorological adverse event (storm, strong wind, blizzard, floods…).

[G6] Allow a policy maker to alert a farmer of winning a good production prize

[R6.1] DREAM must provide a button to make a policy maker send a notice of victory to a farmer

[R6.2] DREAM must provide a temporary interface to make the farmer able to see the notification of victory

[G7] Allow a farmer to compare his work in a time frame

[R7.1] DREAM must provide an interface to make a farmer visualize the trend of his crop selecting a time frame

[R7.2] DREAM must show the most updated version of the graphs

[G8] Allow a registered farmer to ask for help

[R8.1] DREAM must show a button to make the farmer able to ask for help

[R8.2] DREAM must send a help notification to an assistant in the zone

3.2.2 Use Cases

In this paragraph are described several uses cases that represent the elementary scenarios of use of the system by the actors who interface with it (human beings or external information systems).

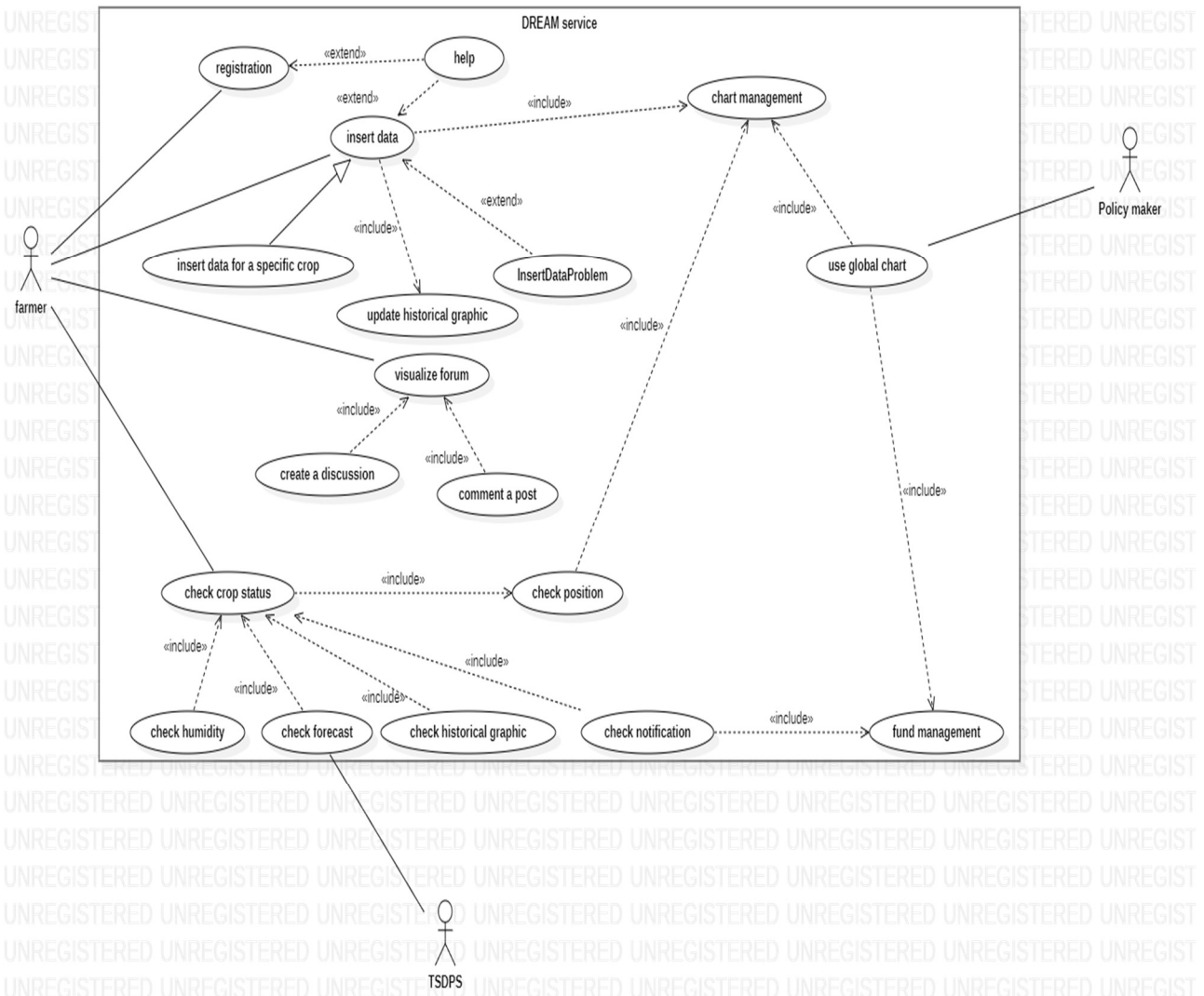


Figure 6: UML Case Diagram Model

Registration

Name	Registration
Related goals	[G1]
Actors	Unregistered farmer or policy maker
Entry Condition	Nothing
Flow Events	1. Unregistered farmer or policy maker opens the registration page in the application. 2. Unregistered farmer or policy maker fills the form with their data (first name, last name, email, username, password, address) and choose the role corresponding their job. 3. Unregistered farmer or policy maker accepts terms and conditions. 4. Unregistered farmer or policy maker submit their data to DREAM application. 5. DREAM check if their data are valid. 6. DREAM send them a confirmation email to the user.
Exit Condition	Unregistered farmer or policy maker info are memorized in the system.
Exceptions	• If at least one of the data inserted by the user is not valid, the application shows a warning error inviting them to insert correct data and then reload the registration page. • If the system observes that there is already a user with the same data, it shows a warning error and load the login page.
Special requirements	Nothing

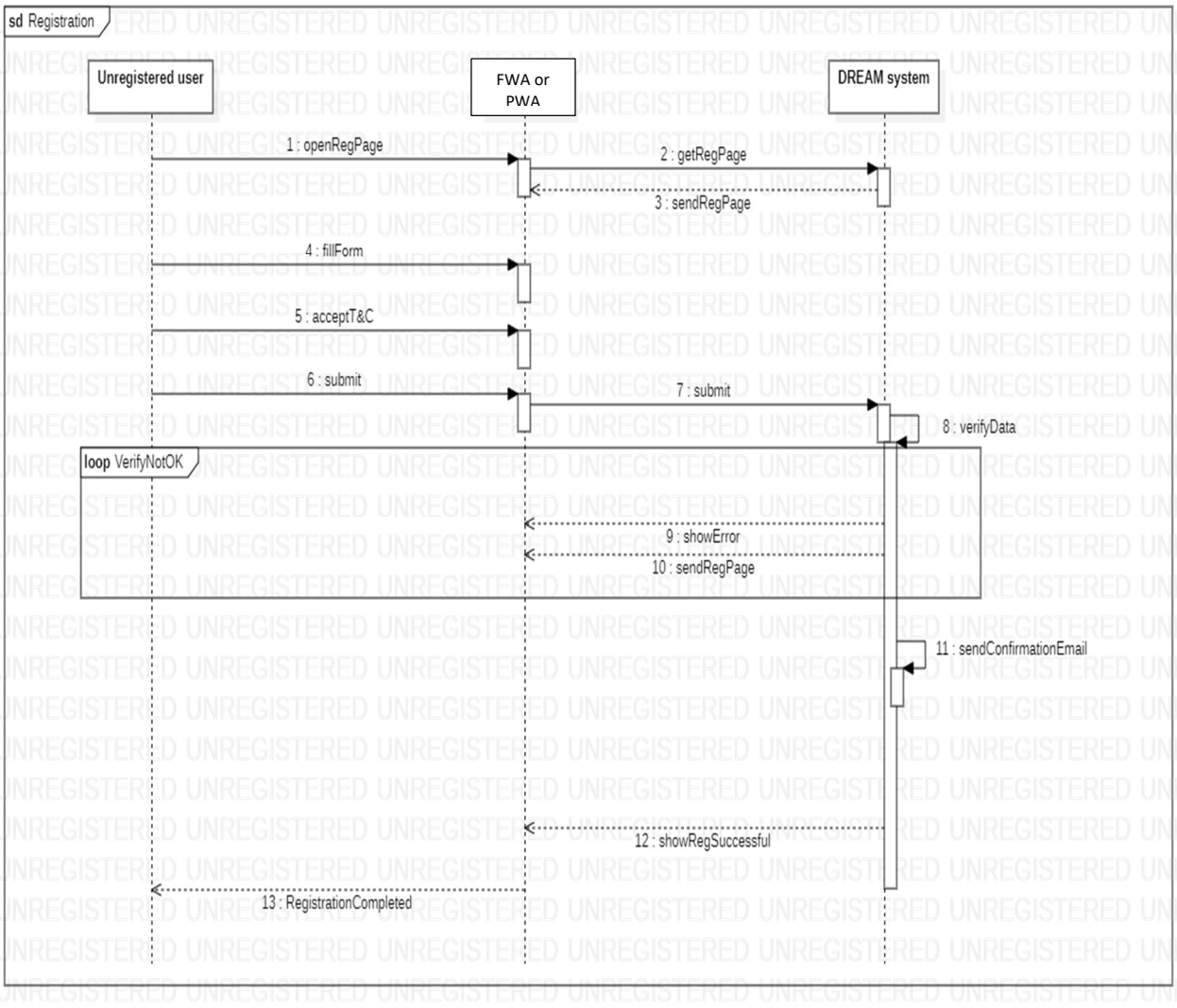


Figure 7: Registration Sequence Diagram

Insert data

Name	Insert data
Related goals	[G2]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer inserts the kg of crop harvested in the appropriate box 3. Registered farmer inserts the kg of crop sowed in the appropriate box 4. Registered farmer clicks the "insert daily data" button and the data enter the system. 5. DREAM checks if the data can be entered 6. DREAM loads the data in the profile of the farmer. 7. DREAM updates the charts on the farmer's profile and the policy maker's charts.
Exit Condition	Registered farmer data are memorized in the system.
Exceptions	• If there are already data for the day, the system does not accept the new data and displays an error message to the farmer.
Special requirements	Nothing

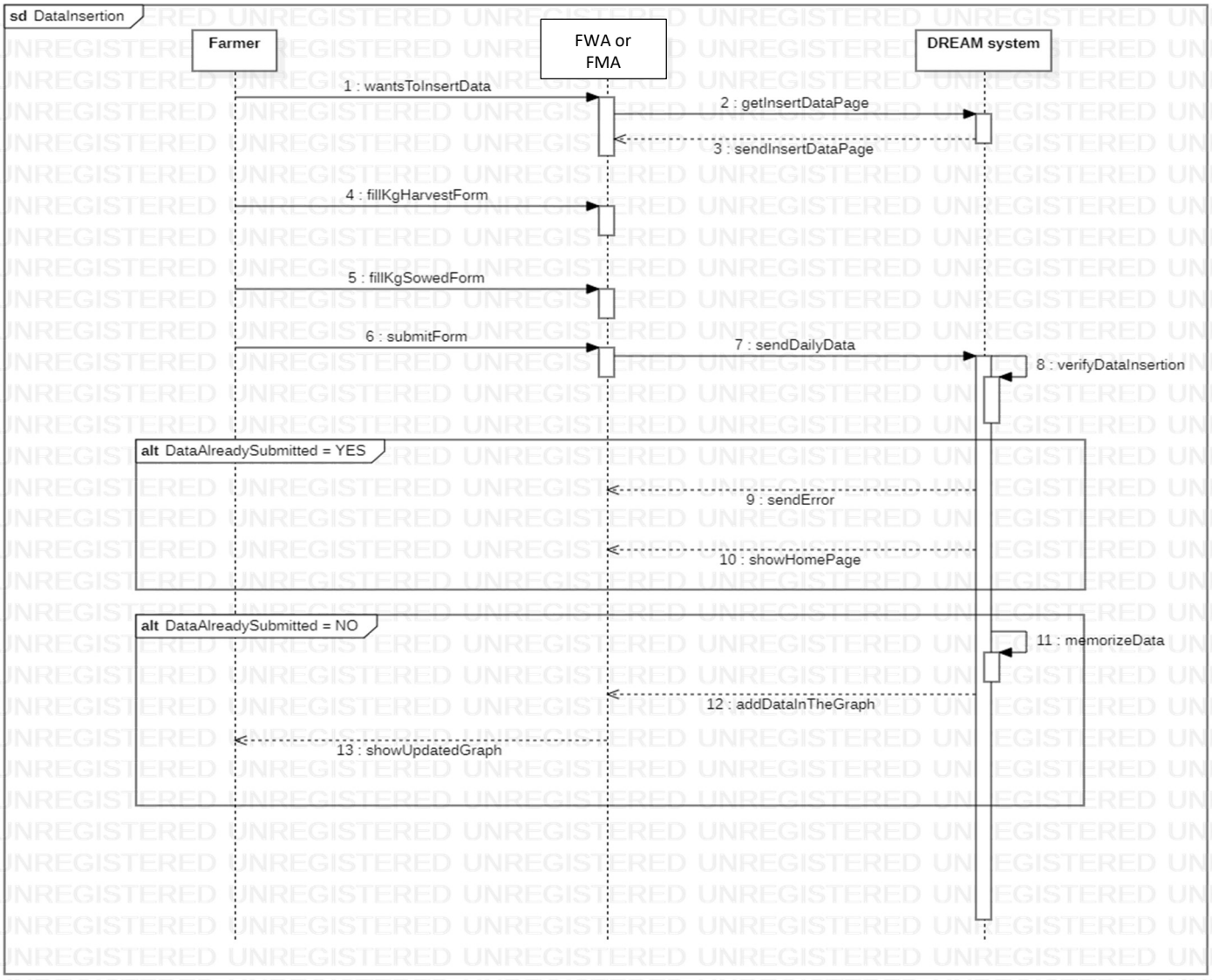


Figure 8: Data Insertion Sequence Diagram

Visualize forum

Name	Visualize forum
Related goals	[G3]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer opens the forum page on the app. 3. Registered farmer chooses from the list of different topics clicking on the corresponding buttons. 4. Registered farmer goes through the different comments by scrolling on the forum page.
Exit Condition	Registered farmer exits from the forum page
Exceptions	• The topic chosen does not have any discussion yet, so the system shows an error message to the farmer
Special requirements	Nothing

Comment on the forum

Name	Comment on the forum
Related goals	[G3]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer opens the forum page on the app. 3. Registered farmer chooses from the list of different topics clicking on the corresponding buttons. 4. Registered farmer clicks on the discussion he wants to comment. 5. Registered farmer clicks on the "+" button to add a comment in the discussion. 6. Farmer writes the comment. 7. Farmer clicks on the "enter" button to publish the comment 8. DREAM publishes the farmer's comment on the discussion in the forum.
Exit Condition	The registered farmer's comment is published on the forum
Exceptions	• If the farmer's comment is empty, the system displays an error message
Special requirements	Nothing

Create a discussion on the forum

Name	Create a discussion on the forum
Related goals	[G3]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer opens the forum page on the app. 3. Registered farmer chooses from the list of different topics clicking on the corresponding buttons. 4. Registered farmer clicks on the discussion he wants to comment. 5. Registered farmer clicks on the "+" button to add a discussion in the forum's section. 6. Farmer writes a comment to create the discussion. 7. Farmer clicks on the "enter" button to publish the comment 8. DREAM publishes the farmer's comment on the forum creating a new discussion.
Exit Condition	The registered farmer's discussion is published on the forum
Exceptions	• If the farmer's comment is empty, the system displays an error message
Special requirements	Nothing

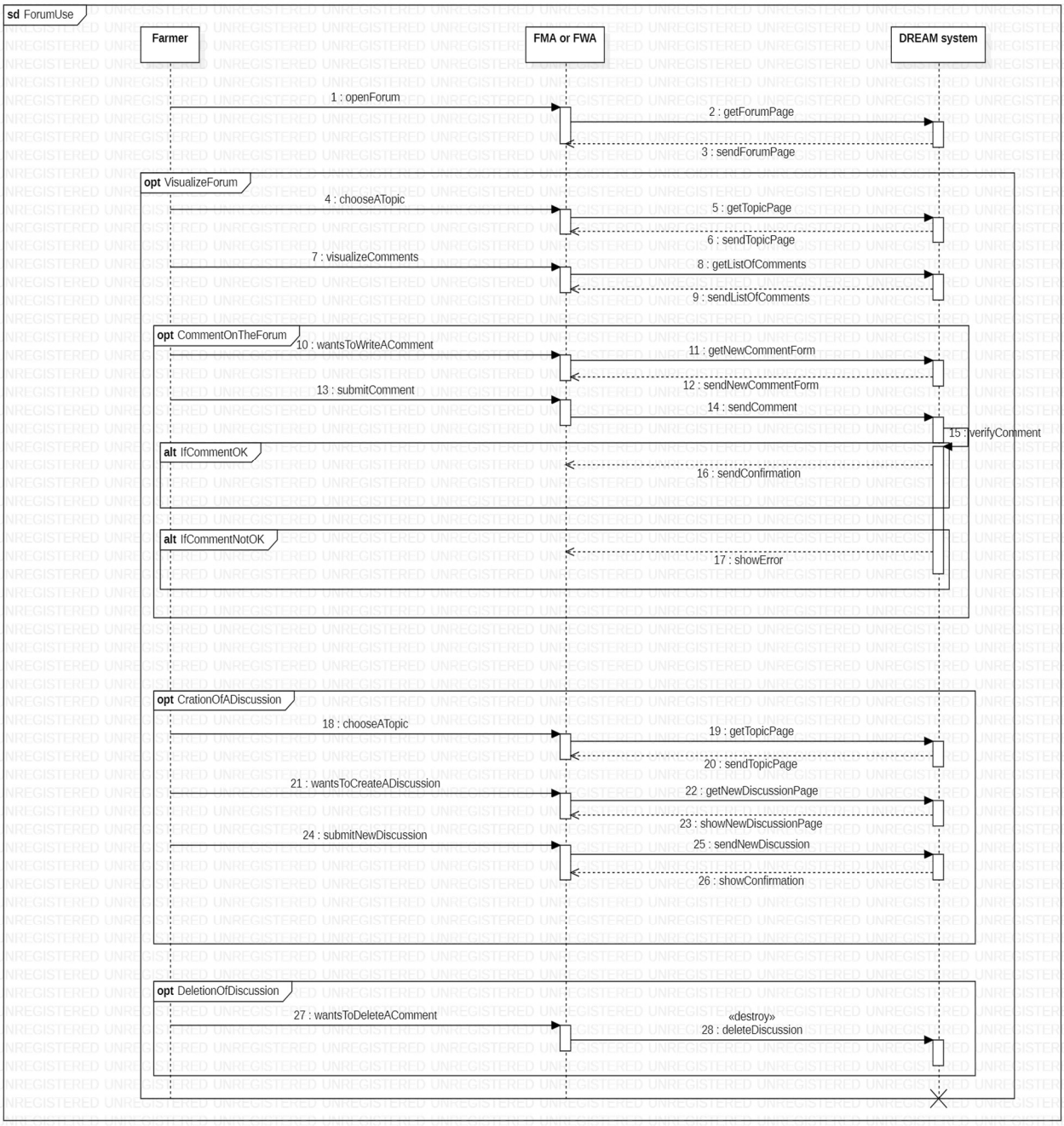


Figure 9: Forum Sequence Diagram

Check crop status

Name	Check crop status
Related goals	[G5], [G6], [G7]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	3. Registered farmer logs in in the app. 4. Registered farmer visualizes the data relative to his crop on the homepage.
Exit Condition	Register farmer closes the app or changes the page visualized
Exceptions	Nothing
Special requirements	Nothing

Check crop charts

Name	Check crop charts
Related goals	[G7]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer visualizes the chart relative to his crop on the homepage. 3. Registered farmer chooses what kind of crop, between the one he has harvested, to look at by clicking on the menu bar. 4. Registered farmer chooses what time frame to look at by clicking on the window menu. 5. DREAM shows the chart selected by the farmer.
Exit Condition	Registered farmer's data are shown on the app
Exceptions	• If the farmer does not have any data yet, an error message is displayed on the app
Special requirements	Nothing

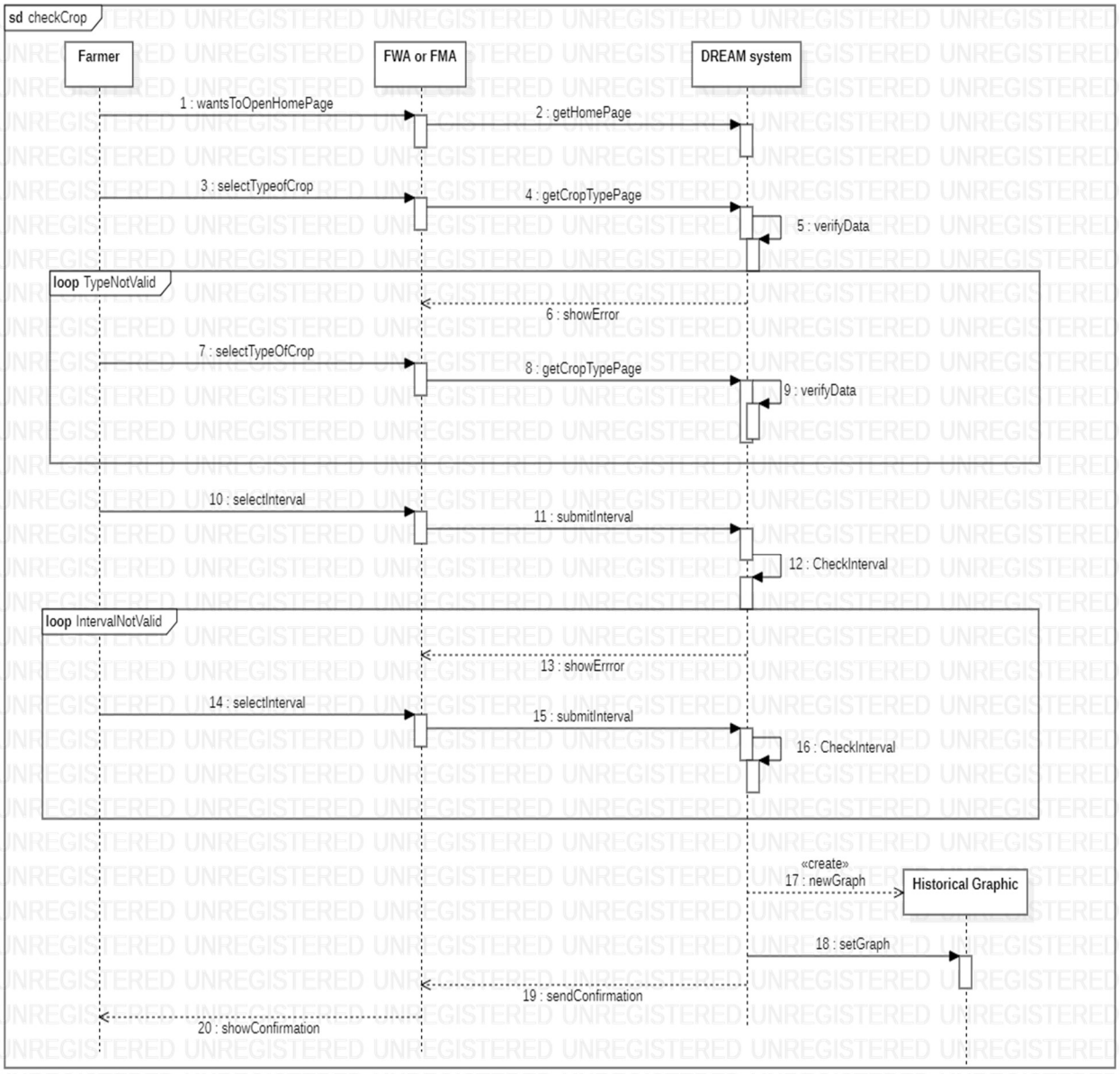


Figure 10: Crop Check Sequence Diagram

Check forecast

Name	Check forecast
Related goals	[G5]
Actors	Registered farmer
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer clicks on the sun button on his homepage. 3. DREAM opens the forecast page. 4. Registered farmer checks the forecast.
Exit Condition	Registered farmer exits the forecast page
Exceptions	• If the connection with the forecast website does not work, the system shows an error message on the app.
Special requirements	Nothing

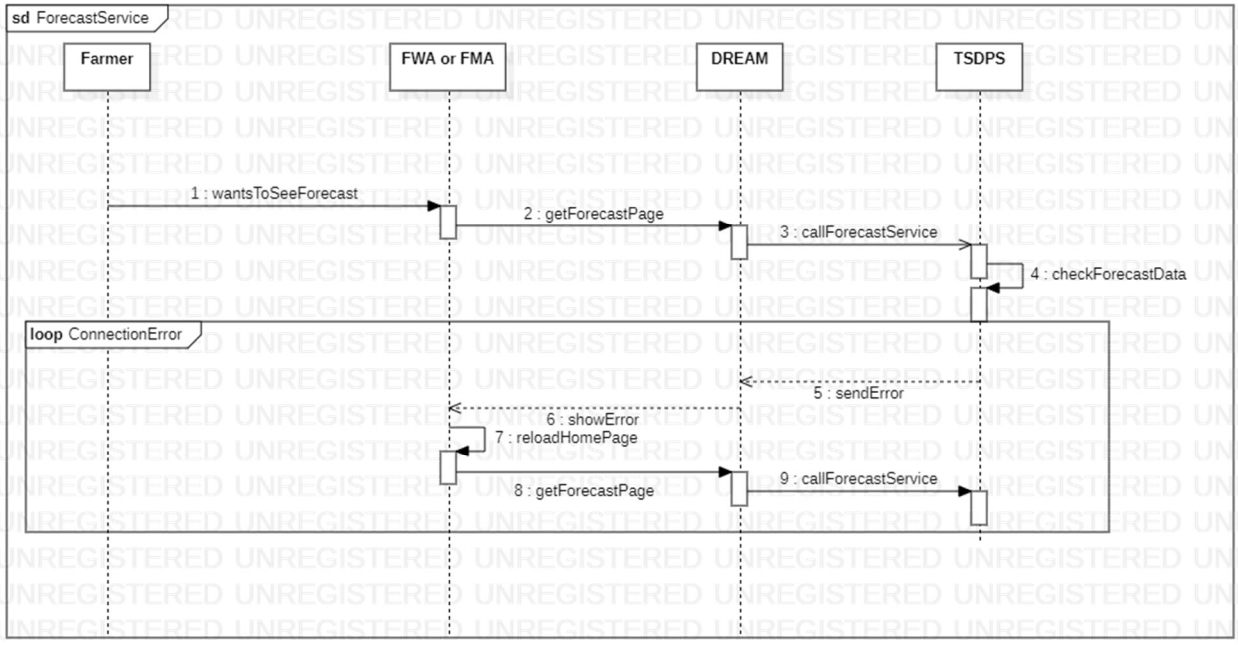


Figure 11: Forecast Service Sequence Diagram

Use global chart

Name	Use global chart
Related goals	[G4]
Actors	Registered policy maker
Entry Condition	Nothing
Flow Events	1. Registered policy maker logs in in the app. 2. Registered policy maker visualizes the chart relative to the farmers in his area on the homepage. 3. Registered policy maker chooses what kind of crop, between the one harvested by the farmers, to look at by clicking on the menu bar. 4. Registered policy maker chooses what time frame to look at by clicking on the window menu. 5. DREAM shows the chart selected by the policy maker.
Exit Condition	The chart is shown on the screen
Exceptions	• If there are not enough data to show a chart, the system displays an error message
Special requirements	Nothing

Help Chat

Name	Help chat
Related goals	[G8]
Actors	Registered farmer, Assistant
Entry Condition	Nothing
Flow Events	1. Registered farmer logs in in the app. 2. Registered farmer clicks on the help button. 3. A help notification is sent to the assistant in the farmer's area.
Exit Condition	The farmer sends a help notification
Exceptions	Nothing
Special requirements	Nothing

In this case we have decided to omit the associated diagram due to the high simplicity of the action sequences.

Prize alert

Name	Prize alert
Related goals	[G6]
Actors	Registered policy maker
Entry Condition	Nothing
Flow Events	4. Registered policy maker logs in in the app. 5. Registered policy maker visualizes the chart relative to the farmers in his area on the homepage. 6. Registered policy maker chooses what kind of crop, between the one harvested by the farmers, to look at by clicking on the menu bar. 7. Registered policy maker chooses what time frame to look at by clicking on the window menu. 8. DREAM shows the chart selected by the policy maker. 9. Registered policy maker chooses a farmer on the chart clicking on it. 10. DREAM shows the page of the farmer to the policy maker. 11. Registered policy maker decides to give this farmer the prize and alert him by clicking on the prize button on the farmer's profile. 12. DREAM sends a notification of prize victory to the farmer
Exit Condition	The farmer receives the prize victory notification
Exceptions	• If there are not enough data to show a chart, the system displays an error message
Special requirements	Nothing

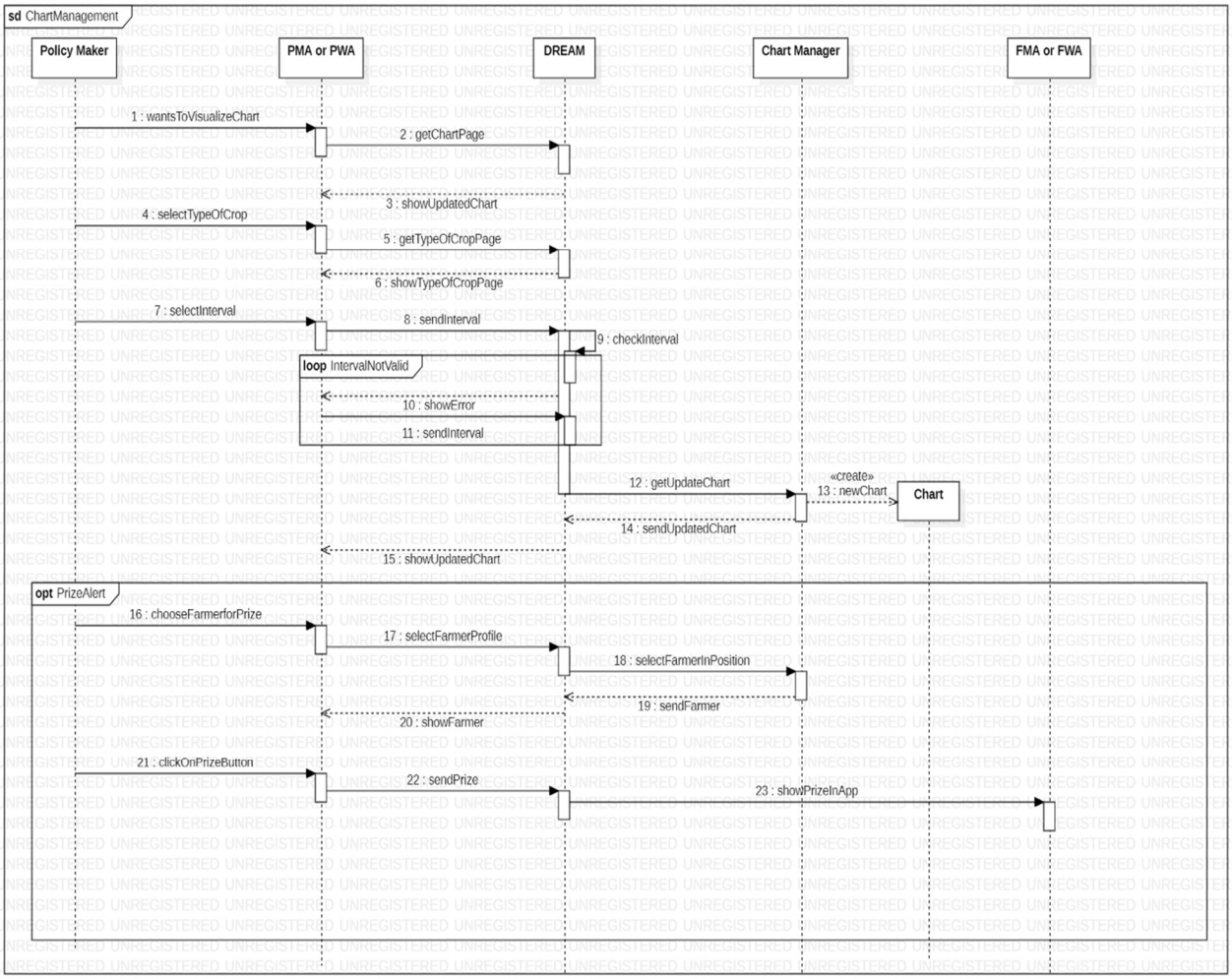


Figure 13: Chart Management Sequence Diagram

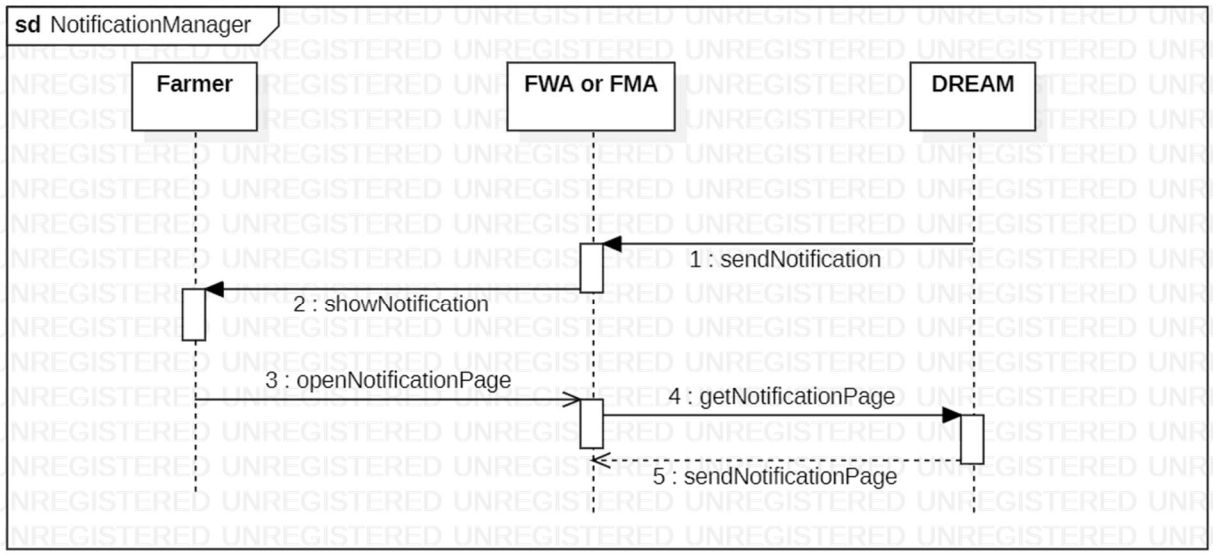


Figure 14: Notification Management Sequence Diagram

3.3 Performance Requirements

In this paragraph is described how every component should perform and user interaction with the system as a set of components.

- Platform: Each data inserted from users in the web application should be computed on the servers of the system, therefore also mobile app should send all data in such a way as to be a thin client software.

- Workload: Since most farmers and policy makers works during daytime, the load of data is expected to decrease considerably during the night.

- Scalability: As specified in the previous point, under light load not all servers must perform at maximum power. Instead, in condition of excessive data traffic, the system must guarantee the availability and efficiency to all users; so, all servers must perform at maximum power.

- Time Response: So long as it is a web-based system, time response must be the lowest possible: at least 90% of the pages must respond in 2 seconds or less, including infrastructure, excluding backends.

3.4 Design Constraints

3.4.1 Standards compliance

These other constraints are mandatory to fulfil the correct functioning of the system:

- Since system does not use GPS service, users have to send all information to DBMS systems which store data in accordance with the more recent standard that ensure the consistency and effectiveness of the database environment.

3.4.2 Hardware limitations

- Mobile farmers and policy maker must download the mobile app available in the most common store application (Google Play Store, AppStore, Microsoft Store). It is assumed that the mobile phones have enough primary memory to run the application.
- The browser used by web users (farmers and policy maker) to access the system must have cookies enabled.
- Each farmer is provided with a smartphone where FMA application must be installed.

3.4.3 Other constraints

- The system will ask for farmers' field information and users have to agree in advance to specific terms and conditions.
- Policy maker must not share collected farmers' information outside of the system.
- System implements stateless protocol in order to improve scalability, reliability and to have better performance in each session.

3.5 Software System Attributes

Since DREAM system aims to increase smallholder farmers crops and decrease food supply shock, according to stakeholders' expectations we identified the following non-functional requirements.

3.5.1 Reliability

To be able to offer a continuous service, the system must be fault tolerant. To do so, mechanisms to prevent error propagation and to handle errors need to be implemented.

3.5.2 Availability

It is essential to have the lowest downtime possible, especially for the assistance and forecast functions (see section 2.2) because they are unpredictable events. These components shall guarantee 99.9% (*three-nines*) of availability, so that only 8.76 hours of downtime per year are allowed. Meanwhile the data insertion and chart checking components shall guarantee 99% (*two-nines*) of availability, so that 3.65 days of downtime per year are allowed.
A backup of data should be executed every 3 months so that prevent data loss due to unexpected events.

3.5.3 Security

All data collected from the users must be stored in safe servers accessible only by authorized staff so that avoid any manipulating, loss or stealing.
The communication between users and DREAM system is based on cryptographic protocol so to provide secure connection through the parts.

3.5.4 Maintainability

To provide an efficient maintainability during the entire life of the system, it should be implemented into different modules, following MVC and Observable design patterns.
DREAM system must supply external APIs to give the possibility of custom to external developers.

3.5.5 Portability

FMA and PMA shall be downloaded in the most common mobile operative systems.
FWA and PWA shall be reachable through the most web browser, having cookies enabled.

4 FORMAL ANALYSIS USING ALLOY

```
sig DateTime{
      date: Date,
      time: Time
}

sig Date{}
sig Time{}

sig Name {}
sig Lastname {}
sig Password{}
sig Email {}

sig Location{
      forecast: one Weather
}

abstract sig RegisteredUser{
      name: Name,
      lastname: Lastname,
      email: Email,
      password: Password
}

sig RegisteredFarmer extends RegisteredUser{
      location: one Location,
      fields: some Field,
      totalCrop: Int,
      totalPoints: Int,
      position: Int
}

sig RegisteredPolicyMaker extends RegisteredUser{
      controlledArea: one Location
}

abstract sig Sensor{}

sig HumiditySensor extends Sensor{
      humidityLevel:Int
}

sig WaterSensor extends Sensor{
      waterLevel: Int
}

sig Crop{}

sig Field{
      location: Location,
      crop: Crop,
```

```
	humidity: Int,
	sensors: some Sensor,
	cropHarvest: Int,
	waterUsed: Int,
	points: Int
}

sig DataInsertion{
	dateTime: DateTime,
	farmer: RegisteredFarmer,
	crop: Crop,
	cropHarvest: Int
}

abstract sig Request{
	dateTime: DateTime
}

sig ForecastRequest extends Request{
	locationRequested: Location,
	applicant: RegisteredFarmer
}

abstract sig TimeFrame{}

sig FarmersChartRequest extends Request{
	applicant: RegisteredPolicyMaker,
	timeFrame: TimeFrame
}

sig CropGraphRequest extends Request{
	applicant: RegisteredFarmer,
	crop: Crop,
	timeFrame: TimeFrame
}

sig FarmerChart{
	location:Location,
	chartRequest: one FarmersChartRequest,
	farmersList: some RegisteredFarmer,
	bestFarmers: some RegisteredFarmer,
	worstFarmers: some RegisteredFarmer,
	timeFrame: TimeFrame,
	version: DateTime
	}

sig CropGraph{
	crop: one Crop,
	request: one CropGraphRequest,
	farmer: one RegisteredFarmer,
	data: some DataInsertion,
	timeFrame: one TimeFrame,
	version: one DateTime}

abstract sig Notification{
	dateTime: DateTime,
	receiver: one RegisteredFarmer
```

```
}

sig PrizeNotification extends Notification{
      sender: RegisteredPolicyMaker
}

sig ForecastNotification extends Notification{
      weather: Weather
}

abstract sig Weather{}
one sig Storm extends Weather{}
one sig Aridity extends Weather{}
one sig Sunny extends Weather{}
one sig Windy extends Weather{}

abstract sig Forum{}
sig Discussion extends Forum{
      dateTime: one DateTime,
      creator: one RegisteredFarmer,
      topic: one Topic,
      comments: set Comment
}

abstract sig Topic{}

sig Comment extends Forum{
      status: one Status,
      dateTime: one DateTime,
      writer: one RegisteredFarmer
}

abstract sig Status{}
one sig online extends Status{}
one sig cancelled extends Status{}



//FACTS


fact { //there are no 2 accounts with the same email associated
      no disj r1, r2: RegisteredUser| r1.email = r2.email
}

fact { //a farmer has at least one field
      all f: RegisteredFarmer | some fi: Field | fi in f.fields
}

Fact { // each farmer, if he has more than 1 field, has fields different
from each others
      all disj f1, f2: Field| f1.farmer =f2.farmer implies f1.crop!=f2.crop
}
```

```
fact { //crop harvest or sow by a farmer is always a quantity major or equal
to zero, so are his points
      all f: RegisteredFarmer | f.totalCrop>=0 and f.totalPoints>=0
}

//each field is owned by  a farmer
fact AllFieldsHaveOnlyAFarmer {
      all f: Field | one fa: RegisteredFarmer | f in fa.fields
}

//a field cannot be owned by two farmers
fact no2FarmersFor1Field{
      all disj f1, f2:RegisteredFarmer| (f1.fields & f2.fields)= none
}

//location of a field of a farmer corresponds to farmer location
fact allFieldsInFarmerLocation{
      all f: Field | all fa: RegisteredFarmer | f in fa.fields
            implies f.location = fa.location
}

fact { //crop harvest or sow in field and its points are always a quantity
major or equal to zero
      all f: Field | f.cropHarvest>=0 and f.points>0
}

fact { //each field has at least an humidity and water sensor connected
all field:Field|one watSens: WaterSensor | one humSens: HumiditySensor |
      watSens in field.sensors and humSens in field.sensors
}

fact { //each sensor is connected to a field only
      all s: Sensor | one f: Field | s in f.sensors
}

fact { // there are not 2 fields with the same sensor
      all disj f1,f2: Field | (f1.sensors & f2.sensors)=none
}

fact{ //the level of humidity and water used is major or equal than 0
all f: Field|  f.humidity>0 and f.waterUsed>=0
}

fact{ //each crop is in a field
      all c:Crop| one f: Field| c=f.crop
}

fact { //each area has one policy maker
      all l: Location | one p: RegisteredPolicyMaker| p.controlledArea=l
}

fact { //if the weather is bad, the farmer is notified
      all w: Weather|some n: ForecastNotification|all rf: RegisteredFarmer|
(w!=Sunny and rf.location.forecast=w)
      implies (n.receiver=rf and (n.weather)=w)
}
```

```
fact { //each notification has only a receiver
      no disj n1, n2: Notification | n1.receiver=n2.receiver
}

fact { //forecast notifications are send only when the weather is bad
      no n: ForecastNotification | n.weather=Sunny
}

fact { //the weather notified is the one of the receiver
      no n: ForecastNotification| n.weather!=n.receiver.location.forecast
}

fact { // the forecast request is made on the location of the sender
      no n: ForecastRequest| n.locationRequested!=n.applicant.location
}


fact { //a farmer has to send data once a day, once he has done it, he
cannot do it again
      all disj d1, d2: DataInsertion| (onTheSameDay[d1,d2] and
      d1.crop=d2.crop)
            implies d1.farmer != d2.farmer
}

fact { //data about the fields are sent everyday
      no date: DateTime| all rf: RegisteredFarmer | all d: DataInsertion |
d.farmer=rf and d.dateTime=date implies
      #rf.fields!=#d
}

fact { //crop harvest or sow in dataInsertion is always a quantity major or
equal to zero
      all di: DataInsertion | di.cropHarvest>=0
}

fact { //each crop has a related graph
      all f: Field| one g: CropGraph |one rf: RegisteredFarmer| g.crop =
f.crop
      and (f in rf.fields) and g.farmer=rf
}

fact { //each cropgraph request has a related graph
      all cr: CropGraphRequest| one g: CropGraph |g.request=cr
}


fact { //data in the cropGraph, taken from the data insertion, are of the
right crop and the same farmer
      all cropG: CropGraph| all di:DataInsertion| di in cropG.data implies
      ((di.crop=cropG.crop) and (di.farmer=cropG.farmer) )
}

fact{ // points of a field are major or equal to the crop harvest in that
field
      all f:Field| f.points>=f.cropHarvest
}
```

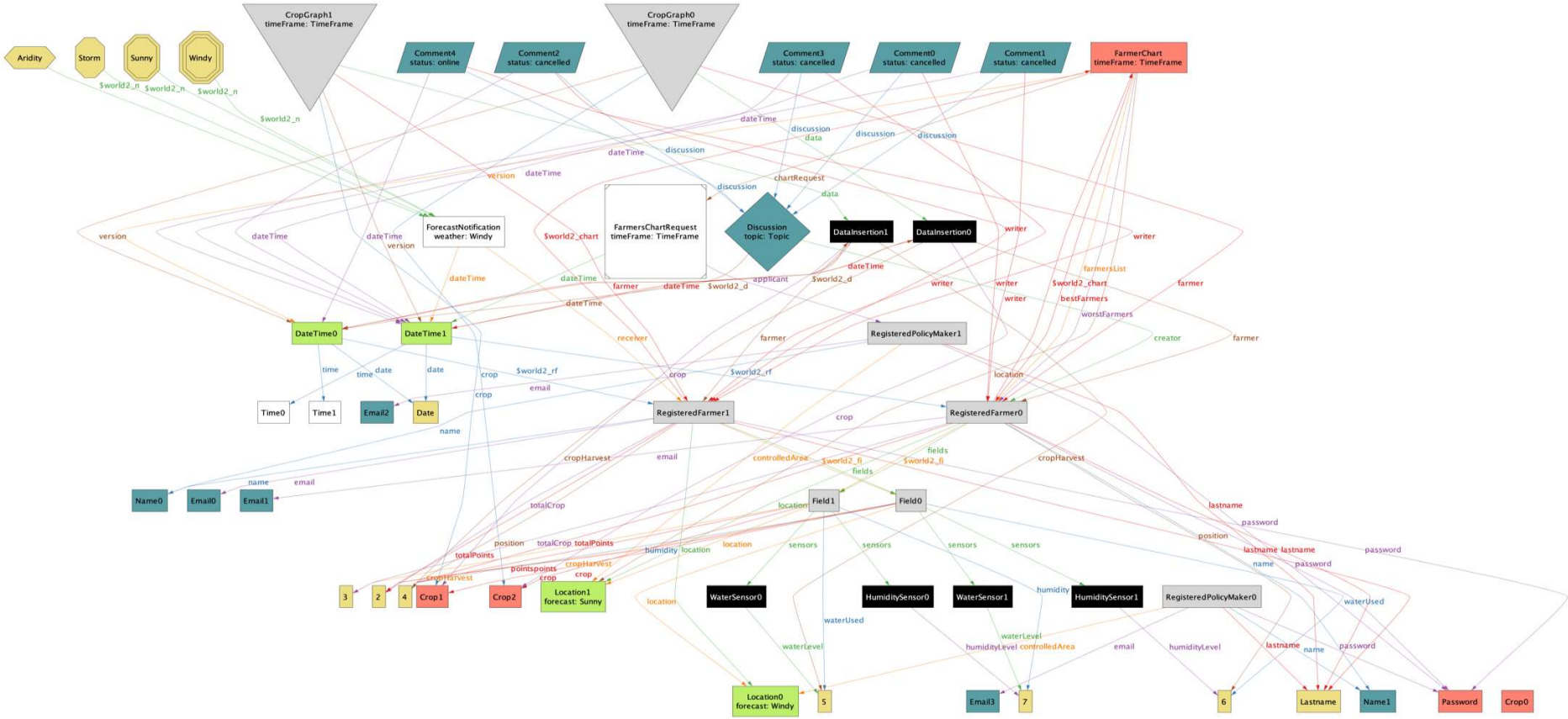
```
fact{ // crop of a field is equal to the crop harvest in that field in the
data insertion
      all f:Field| some d: DataInsertion| d.crop=f.crop and f in
d.farmer.fields implies f.cropHarvest=getSumCropData[d]
}

fact { //farmer's points are equal or more than the sum of crop harvest in
his fields
      all rf: RegisteredFarmer | rf.totalPoints>=getFieldsPoints[rf] }


fact{ //farmer's total crop is the sum of the crop of his fields
      all rf: RegisteredFarmer| rf.totalCrop=getFieldsCrop[rf]
}


fact { //in each farmerChart there are all the farmers of that location
      all chart:FarmerChart|all f: RegisteredFarmer|
f.location=chart.location implies
      f in chart.farmersList
}

fact { //in each farmerChart there are no farmers of a different location
      no rf: RegisteredFarmer| all chart:FarmerChart|
rf.location!=chart.location and
      rf in chart.farmersList
}

fact { //cannot have two different farmers in same position in chart in same
location
      no disj f1,f2: RegisteredFarmer| f1.location=f2.location and
f1.position=f2.position
}


fact { //points of a farmer in a lower position in the chart are less than
one in a better position
      all disj f1,f2: RegisteredFarmer| (f1.position>f2.position and
f1.location=f2.location) implies
            f1.totalPoints>f2.totalPoints}


fact { //two charts do not have farmers in common
      all disj chart1, chart2: FarmerChart| (chart1.farmersList &
chart2.farmersList) =none
}

fact { //Chart timeframe is the same of the related request
      all chart: FarmerChart|all request: FarmersChartRequest|
chart.chartRequest=request implies
      chart.timeFrame=request.timeFrame
}

fact { //Chart location is the same of the related request
      no chart: FarmerChart| chart.location !=
chart.chartRequest.applicant.controlledArea
}
```

```
fact { // policy maker location is the same as farmers in the chart
requested
      all rf: RegisteredFarmer| all p: RegisteredPolicyMaker| all
r:FarmersChartRequest|all c: FarmerChart |
      (one(rf & c.farmersList) and c.chartRequest=r and r.applicant=p)
implies
      rf.location=p.controlledArea
}

fact { //best farmers are in the top 10
      all farmer: RegisteredFarmer |all chart: FarmerChart| farmer in
chart.farmersList and farmer.position<=10
      implies farmer in chart.bestFarmers
}

fact { //if a farmer receives a prize, he is in the top 10
      all prize: PrizeNotification| all c: FarmerChart| all rf:
RegisteredFarmer |
      prize.receiver=rf implies rf in c.bestFarmers}

//the area of the farmer winner is the same of the policy maker sending the
prize
fact PrizeArea{
      all p: PrizeNotification| p.sender.controlledArea =
p.receiver.location
}

fact { //a forum discussion has at least one comment, when created, with
same DateTime
      all d: Discussion| one c: Comment| c in d.comments
            implies atSameTime[d,c]
}

fact { //a forum discussion has at least one comment
      all d: Discussion| #d.comments>=1
}

fact{ //two discussions do not have comments in common
all disj d1,d2:Discussion| d1.comments & d2.comments =none
}

fact { //cancelled comment cannot be in a discussion
      all c:Comment|all d:Discussion| c.status=cancelled implies c not in
d.comments
}

fact { //comments ionline are in a discussion
      all c: Comment | one d: Discussion| c.status=online implies c in
d.comments
}

//FUNCTIONS

//function sum of points of fields of a farmer
fun getFieldsPoints[rf:RegisteredFarmer]: Int{
```

```
      sum p : rf.fields| p.points
}

//function sum of crop harvest from fields of a farmer
fun getFieldsCrop[rf:RegisteredFarmer]: Int{
      sum c : rf.fields| c.cropHarvest
}

fun getSumCropData[d: DataInsertion]: Int{
      sum h: d|h.cropHarvest
}

//PREDICATES

pred onTheSameDay[d1,d2: DataInsertion]{
      d1.dateTime.date = d2.dateTime.date
}

pred atSameTime[d: Discussion, c: Comment]{
      d.dateTime.date =c.dateTime.date and d.dateTime.time=c.dateTime.time
}

//pred if there is no data for the day, they are inserted
pred sendData[setData, newSetData : set DataInsertion, dataInsertion:
DataInsertion, f: some RegisteredFarmer]{
      one field: Field |(no(setData & dataInsertion)) and
dataInsertion.farmer= f and field.crop=dataInsertion.crop and field in
f.fields implies
            newSetData= setData + dataInsertion
      else
            one(setData & dataInsertion) implies
            newSetData = setData
}
run sendData for 10 but 5 Int, 3 RegisteredFarmer

//pred add comment to a discussion
pred insertComment[discussion, newDiscussion: Discussion, comment: Comment]{
      no(discussion.comments & comment) implies
            newDiscussion.comments= discussion.comments + comment and
            comment.status=online
      else
            newDiscussion.comments = discussion.comments
}
run insertComment for 10 but 5 Int, 1 Location, 3 RegisteredUser, 5
Discussion, exactly 4 Comment

//pred delete a comment from a discussion
pred cancelComment[discussion, newDiscussion: Discussion, comment: Comment]{
      (one (discussion.comments & comment) and #discussion.comments>1
)implies
            newDiscussion.comments= discussion.comments - comment and
            comment.status=cancelled
      else
            newDiscussion.comments = discussion.comments
}
run cancelComment for 10 but 5 Int, 1 Location, 3 RegisteredUser, 5
Discussion, exactly 4 Comment
```

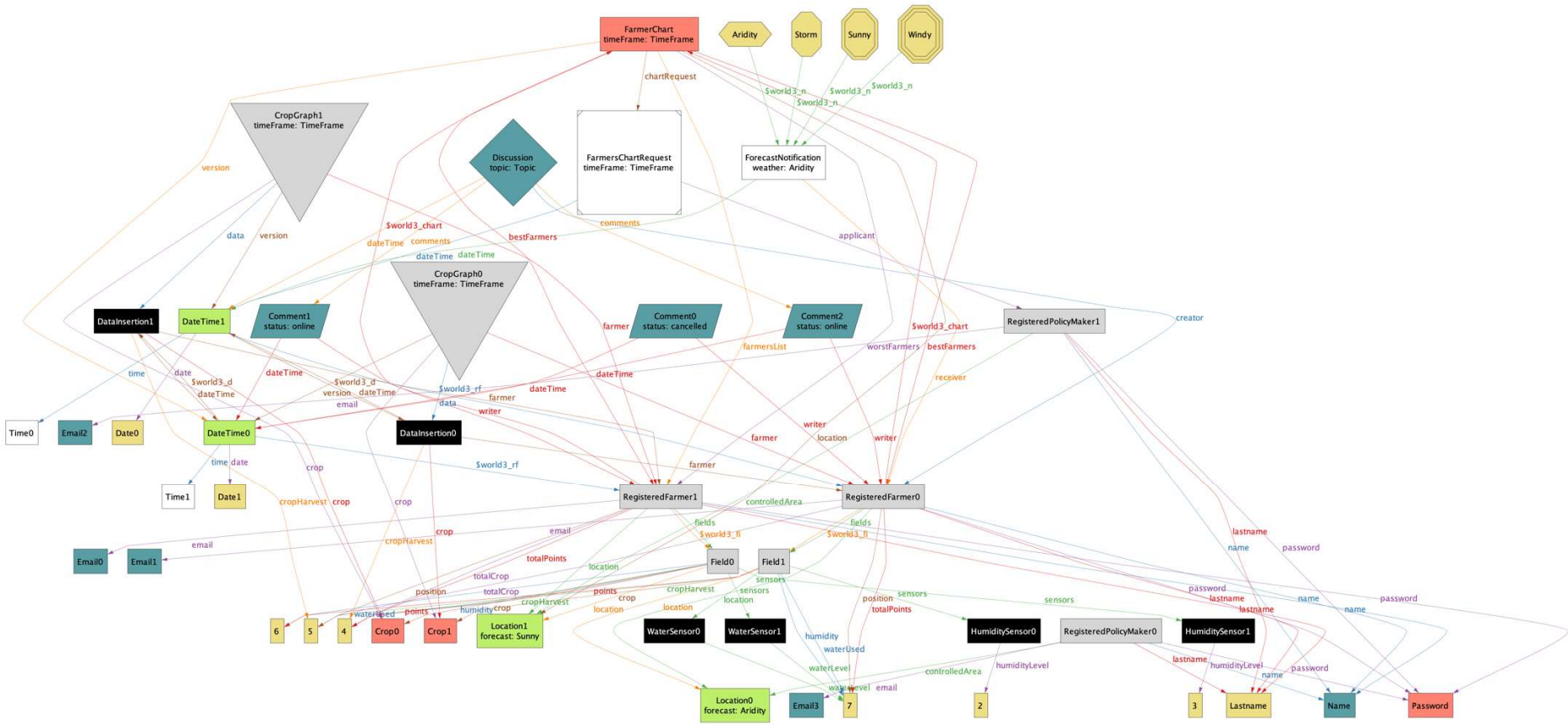
```
// simulation that shows the result some registrations and the basic
relations
pred world1 {
#RegisteredUser>3
#RegisteredFarmer>1
#RegisteredPolicyMaker>1
#Field>2
#Weather>2
#Forum=0
#Discussion=0
#Comment =0
}
run world1 for 6

//simulation on basic relations
pred world2 {
#RegisteredFarmer>1
#RegisteredPolicyMaker>1
#Field=2
#Discussion=1
#Comment >2
#ForecastRequest=0
}
run world2 for 6


//simulation on functional forum
pred world3 {
#RegisteredFarmer>1
#RegisteredPolicyMaker>1
#Field=2
#Comment >2
#ForecastRequest=0
#CropGraphRequest=0
}
run world3 for 4

//simulation on chart and graph request and the connected graphs
pred world4{ //chart and graph request
#RegisteredFarmer>1
#RegisteredPolicyMaker>1
#FarmersChartRequest>1
#CropGraphRequest>1
#FarmerChart>1
#CropGraph>1
}
run world4 for 4

//simulation on forecast and prize notification
pred world5{ //notification forecast and prize
#RegisteredFarmer>1
#RegisteredPolicyMaker>1
#PrizeNotification=1

}
run world5 for 4
```

Executing "Run world1 for 6"
Solver=sat4j Bitwidth=4 MaxSeq=6 SkolemDepth=1 Symmetry=20 Mode=batch
54850 vars. 2964 primary vars. 124344 clauses. 305ms.
Instance found. Predicate is consistent. 807ms.



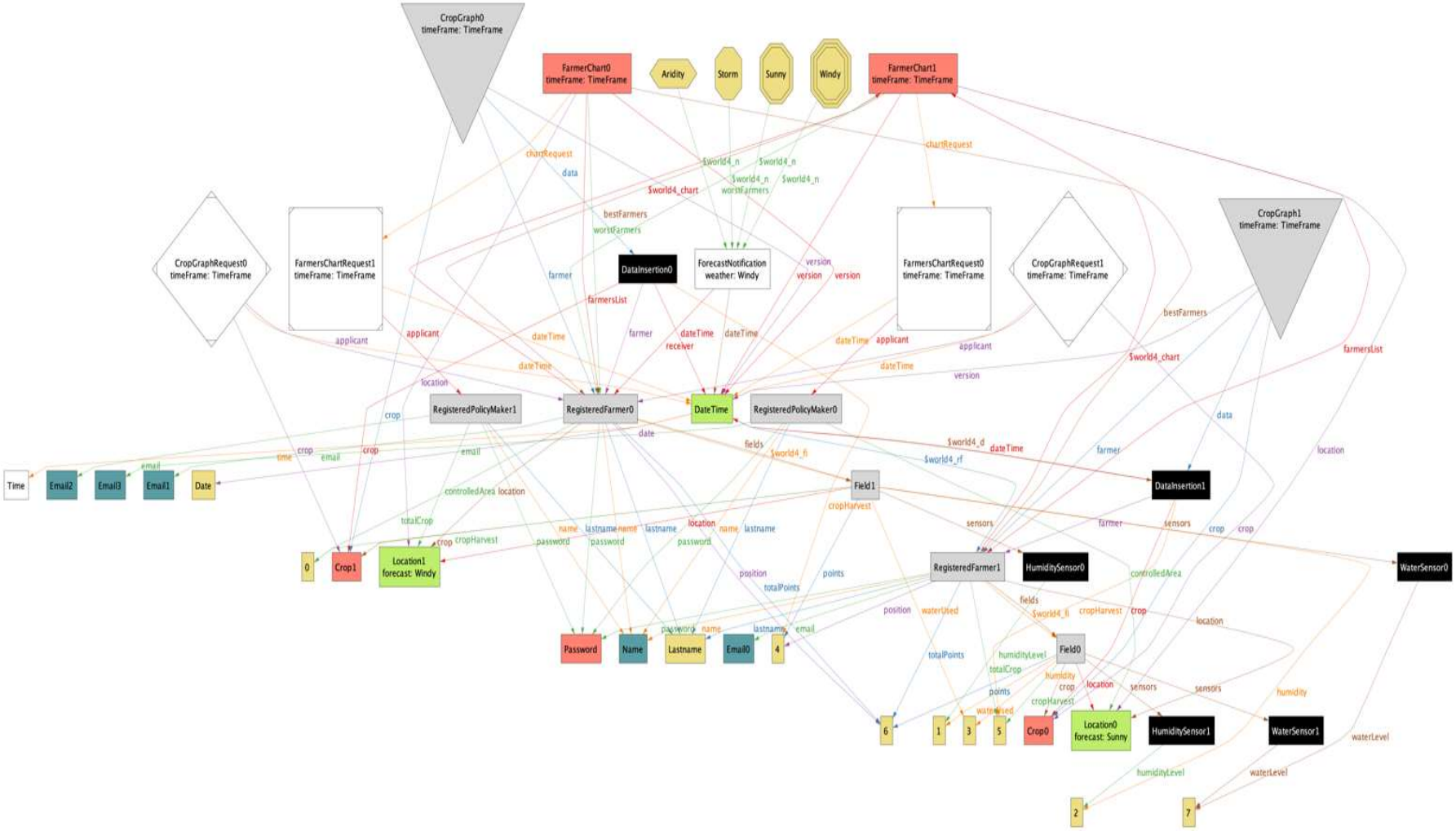
Executing "Run world3 for 4"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
25215 vars. 1568 primary vars. 57814 clauses. 314ms.
Instance found. Predicate is consistent. 156ms.



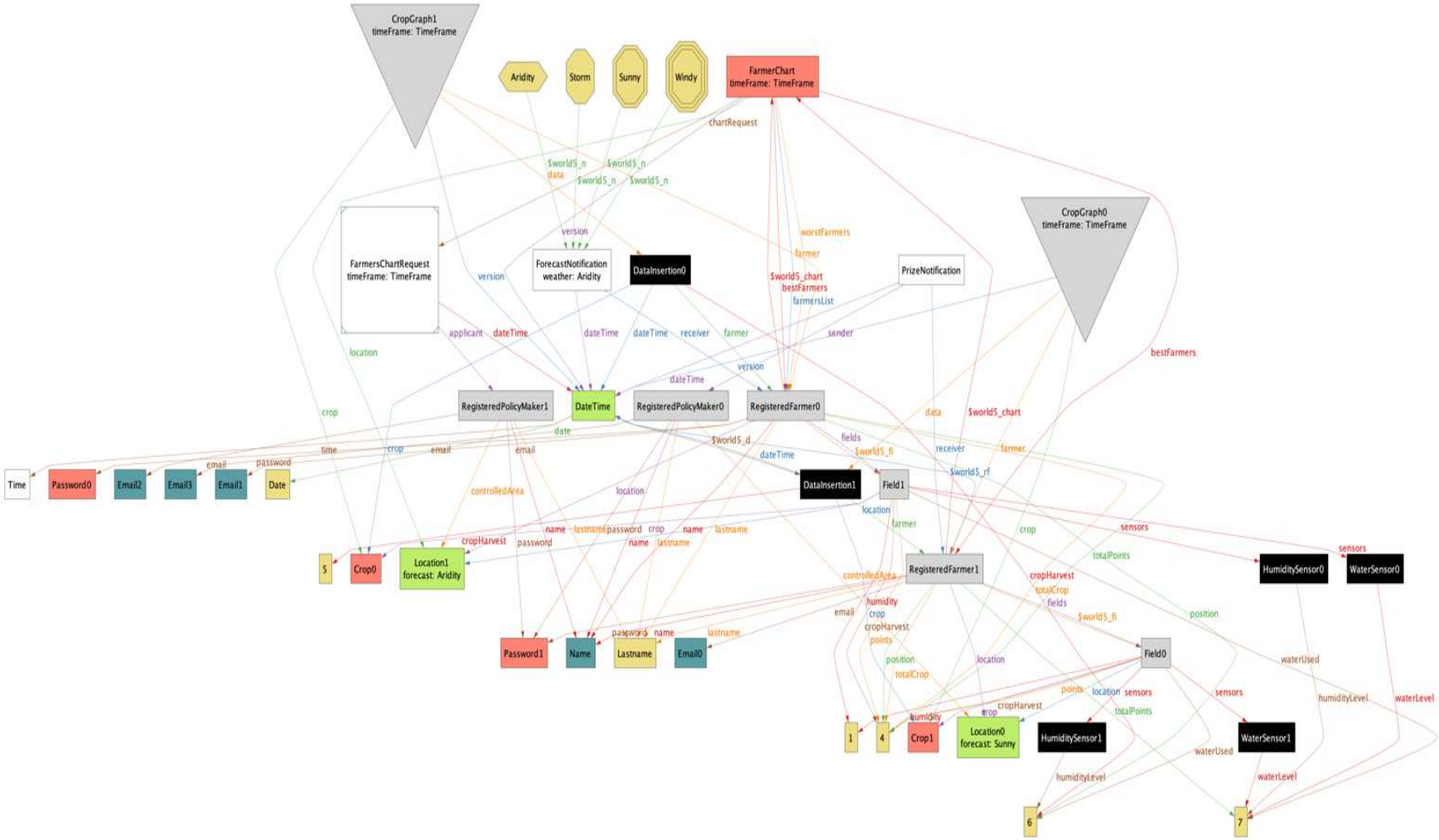
Executing "Run world4 for 4"

Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
25272 vars. 1584 primary vars. 57807 clauses. 129ms.
Instance found. Predicate is consistent. 107ms.

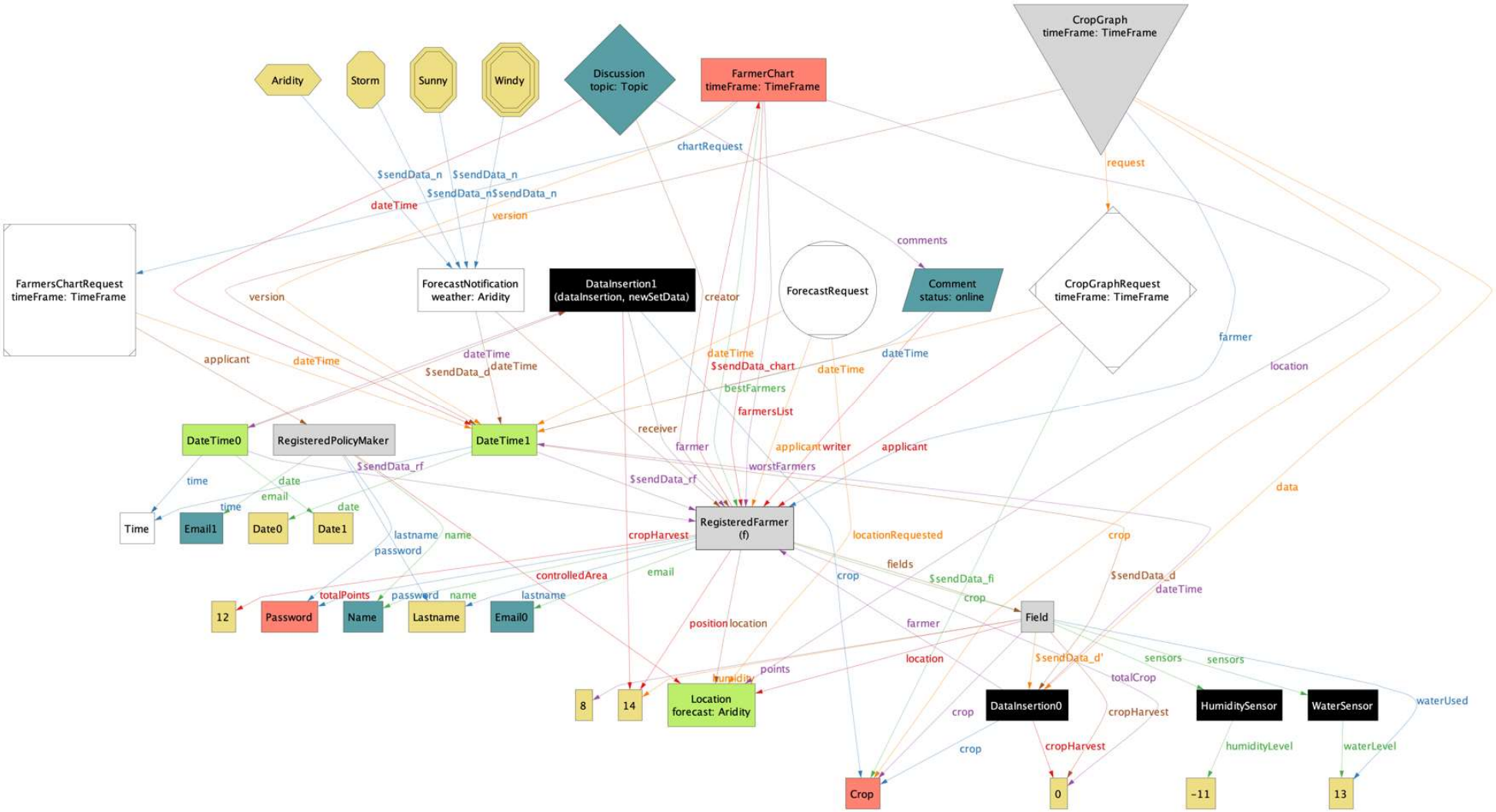


Executing "Run world5 for 4"

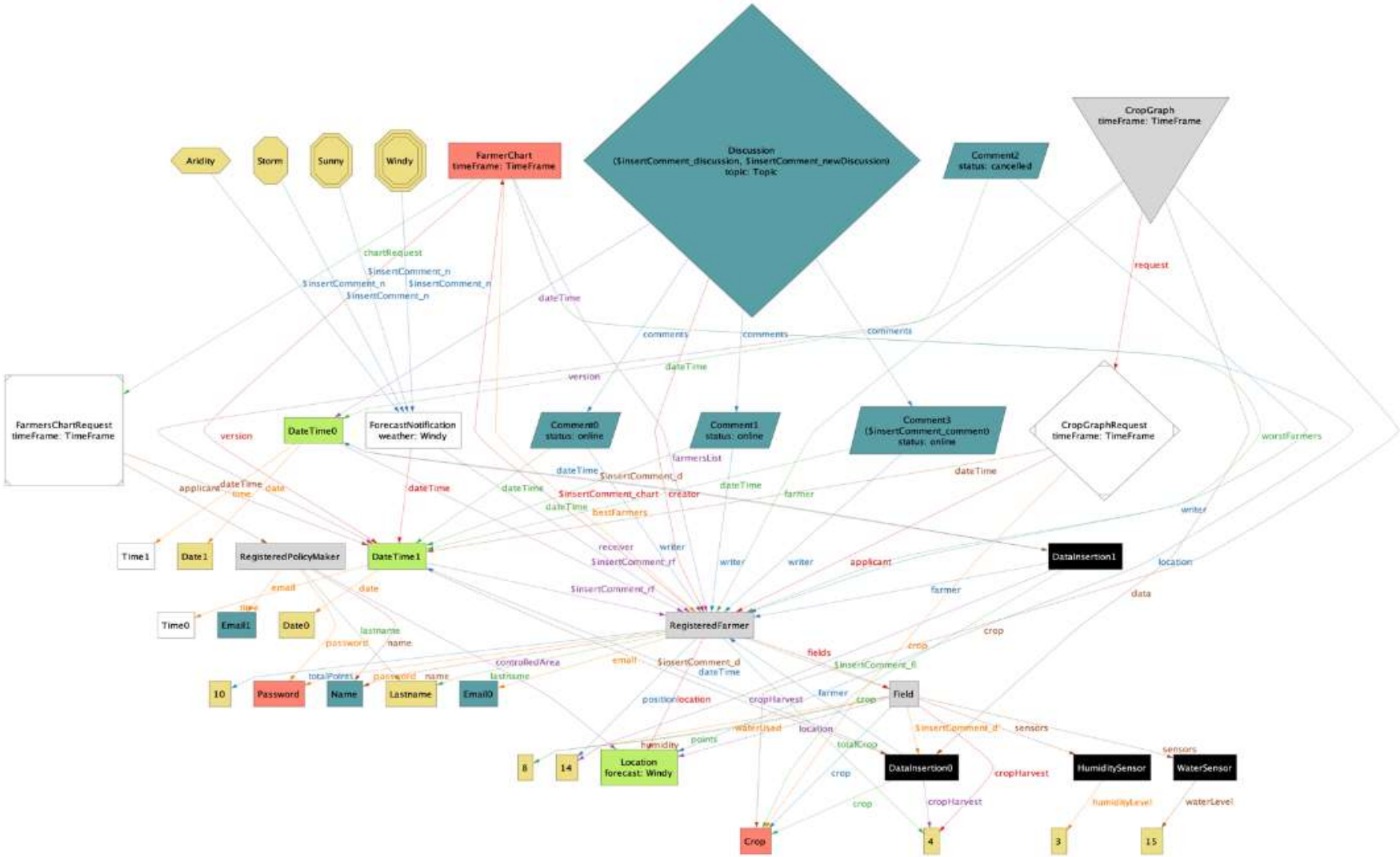
Solver=sat4j Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
25175 vars. 1568 primary vars. 57686 clauses. 197ms.
Instance found. Predicate is consistent. 195ms.



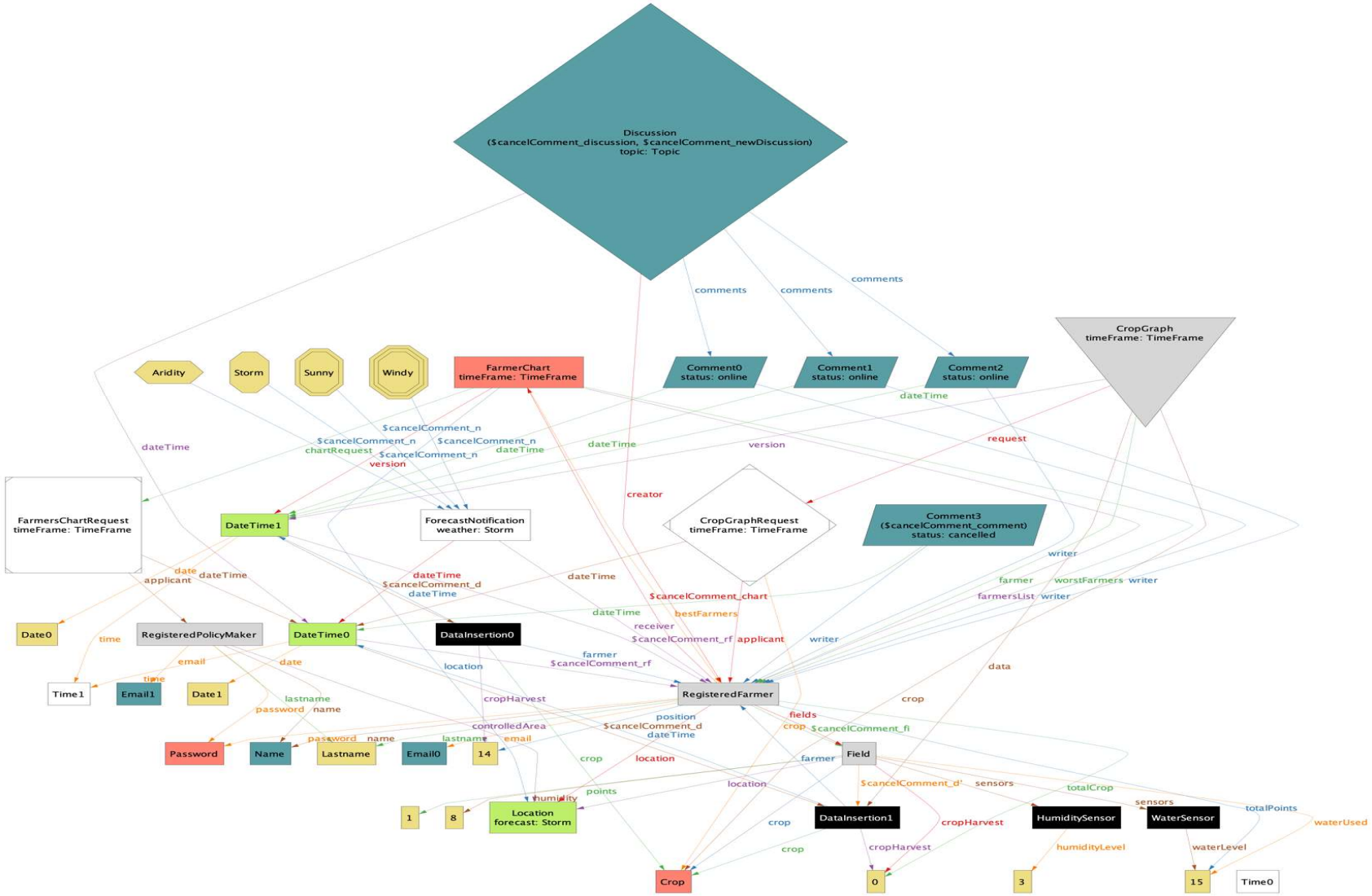
Executing "Run sendData for 10 but 5 int, 3 RegisteredFarmer"
Solver=sat4j Bitwidth=5 MaxSeq=10 SkolemDepth=1 Symmetry=20 Mode=batch
215455 vars. 8640 primary vars. 523865 clauses. 51763ms.
Instance found. Predicate is consistent. 904ms.



Executing "Run insertComment for 10 but 5 int, 1 Location, 3 RegisteredUser, 5 Discussion, exactly 4 Comment"
Solver=sat4j Bitwidth=5 MaxSeq=10 SkolemDepth=1 Symmetry=20 Mode=batch
103628 vars. 5719 primary vars. 269864 clauses. 573ms.
Instance found. Predicate is consistent. 1163ms.



Executing "Run cancelComment for 10 but 5 int, 1 Location, 3 RegisteredUser, 5 Discussion, exactly 4 Comment"
Solver=sat4j Bitwidth=5 MaxSeq=10 SkolemDepth=1 Symmetry=20 Mode=batch
103651 vars. 5719 primary vars. 269946 clauses. 511ms.
Instance found. Predicate is consistent. 2598ms.



5 EFFORT SPENT

Time spent by each group member:

- Camilla Blasucci: 36 h
- Salvatore Buono: 36 h

6 REFERENCES

6.1 Used Tools

1. Microsoft Word (https://www.microsoft.com/it-it/microsoft-365/word) to write this document.
2. Star UML (http://staruml.io/) for use case diagram, class diagram, sequence diagram, activity
3. Alloy Analyser 4.2 (https://alloytools.org/download.html) to build the model and prove its consistency.
4. Balsamiq Mock-up (https://balsamiq.com/wireframes/desktop/) for user interface mock-up generation.

6.2 Reference documents

- Software Engineering 2- year 2021- prof. Di Nitto Elisabetta course slide
- Alloy documentation: https://alloytools.org
- Specification document resources: https://www.tsdps.telangana.gov.in/aws.jsp
- Specification document: "R&DD Assignment A.Y. 2021-2022"