Gonçalo José Esteves Leote

Licenciado em Engenharia Informática

Field Information Web Platform for Agricultural Applications

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Orientador : Carlos Viegas Damásio, Prof. Associado, Universidade Nova de Lisboa

	Júri:
Presidente:	Doutor Pedro Abílio Duarte de Medeiros
Vogais:	Doutor Luís Miguel Mendonça Rato
	Doutor Carlos Augusto Isaac Piló Viegas Damásio

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2013

Field Information Web Platform for Agricultural Applications

To my grandparents

Acknowledgements

Since the beginning of the course, I relied on the trust and support of numerous persons and institutions. Without those contributions, this work would not have been possible.

To Professor Carlos Viegas Damásio, supervisor of the dissertation, I appreciate the support, the sharing of knowledge and valuable contributions to the work. Above all, thank you for being always present and helping me when time began to shorten.

To Professor José Rafael Silva, leader of the research scholarship project PROTOMATE, thank you for helping me whenever I needed and allowing me the opportunity to be part of this project.

I am grateful to all my family for the encouragement received over the years. To my parents, my sister and my grandparents, thank you for the love, joy and attention without reservation ...

My deep and heartfelt thanks to everyone who contributed to the completion of this dissertation, stimulating me intellectually and emotionally.

Abstract

Crops are subject to numerous diseases and pests that may cause significant economic damage. It is essential to forecast the potential occurrence of the major problems in these crops.

The goal of this dissertation is the development of a web platform for registration and issuance of warnings of disease's risk. The generation of these warnings is performed using mathematical models to predict possible occurrence of plant diseases and/or the collection of information in the field by farmers and/or technical experts by means of mobile devices.

This information should be geo-referenced and can be obtained with mobile devices which could be documented with photos, videos, text, data from satellite, weather data, web information or even with information from sensors on the ground.

This thesis describes and shows a fully working platform that is able to collect data and compute alerts from the several sources of information available, in a scalable way.

Keywords: Information system, crop diseases, satellite data, mobile applications.

Resumo

As culturas estão sujeitas a inúmeras doenças e pragas que provocam prejuízos económicos muito significativos. O objetivo desta dissertação é o desenvolvimento de uma plataforma Web de registo e emissão de alertas de risco de doenças.

A geração de alertas de risco é efetuada recorrendo a modelos matemáticos de previsão de doenças das plantas e/ou pela recolha no terreno de informação por parte dos agricultores e/ou de técnicos especialistas utilizando dispositivos móveis.

Esta informação deve ser georreferenciada e pode ser obtida por smartphones podendo ser documentada com fotografias, vídeos, texto, dados satélite, dados metereológicos, informação da Web e ou mesmo com informação de sensores existentes no terreno.

Esta tese descreve e mostra uma plataforma totalmente funcional, capaz de coletar dados e calcular alertas das várias fontes de informações disponíveis, de uma forma escalável.

Palavras-chave: Sistema de informação, doenças de culturas, dados de satélite, aplicações móveis.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and context

Throughout the history of mankind, plant diseases have been responsible numerous times for large losses in our society, such as death of populations due to starvation, extinction of natural resources and direct negative impact on national economies [1].

Nowadays, the majority of plant diseases and pests do not cause such serious danger but still constitute substantial losses to farmers and may reduce value of their growing areas. To reduce losses, there are ways to manage these plant diseases. According to [1], plant disease control is focused on 2 principles:

1. Prevention
2. Curative action

Prevention is about disease management tactics that are applied before plants are infected or ill, while curative action is about methods applied after infection or problems occur.

Prevention products are better than curative products since they prevent the loss of production, so it is reasonable to conclude that the best way to control diseases is by focusing on their prevention.

Thus, this dissertation proposes a computer system that estimates plant disease infection risk and allows farmers and agronomists to verify the possibility of diseases in their crops, and shows its practical feasibility.

While developing this dissertation, I worked for the PROTOMATE project which was made in collaboration with "Universidade de Évora" and "Instituto Português do Mar e da Atmosfera," that showed great interest in this subject, helping to better understanding of some of the specific issues and also providing the necessary data and agronomic expertise for the implementation.

1.2 Objectives

The proposed solution is an information system that helps to determine the potential risk of occurrence of crops' problems and allows users to get updates on this risk. By no means it is foreseen to predict automatically the occurrence of diseases, but instead the aim is to provide information to help the users making better decisions.

These notifications can be generated by using mathematical models to suggest when the farmers should spray their crops in order to keep them healthy, or to determine whether the conditions are appropriate for the occurrence of the disease or pest.

The existing mathematical models require weather information data, such as air temperature, rainfall/precipitation and other obtained sensor data like leaf wetness.

Some of this data is obtained by satellite and via weather stations with the help of the "Instituto Português do Mar e da Atmosfera", but some other (leaf wetness for example) needs to be estimated with mathematical models.

We will restrict to grape wines and some tomato diseases but models for other diseases or pests can be easily integrated. These models have been selected to show the kind of data and computation required.

For our mobile applications we'll allow users to get notifications about their crops possible risk, when they decide to do it. Besides, users will be able to interact with our system adding their own observations made like plant diseases observed, plant treatments made and traps observed.

An additional objective was made due to my work for the PROTOMATE project. The goal was to create a system that would store meteorological information and allow users to analyze this data in order to verify relations between air temperature, land surface temperature and relative humidity in any location of Portugal. This is a particular relevant application of the developed platform.

1.3 Major contributions

1.3.1 Innovation using satellite data

There are many projects similar to this dissertation where systems calculate plants' disease risk based on weather data from weather stations. However, weather stations require appropriate maintenance and requiring significant amounts of time which comes with corresponding costs.

However, satellite information is becoming freely available, and increasingly subject to research in order to increase their functionality, information that can be obtained and corresponding applications. Another advantage is the fact weather stations are specific to a location while satellite data allows to have global perspective of large geographical areas.

This dissertation hopes to contribute to a viewpoint with an innovative system that obtains information via satellite with the possibility of being improved in the future, combined with information available from weather stations.

1.3.2 Managing large volumes of information

Every 15 minutes our system can get information via satellite and every hour via weather stations.

This, it will be necessary to know how to manage this information and ensure that no processing takes longer than the maximum time allowed.

All this information will be maintained, allowing our system to become a large database with the possibility of being used for future analysis, research and maybe help the tuning of disease and pest models.

1.3.3 User interaction

Another aspect of the contribution that our system can provide is that the users (in this case agronomists and farmers) can interact directly with the system, alerting for real cases of infection that may exist. This way it allows the system to interact with other nearby users, warning them, and creating an interactive social network.

1.4 Document organisation

This report is divided into 7 chapters:

1. Introduction

   Summary of what is and what was done, including our goals and contributions.

2. Problem

   Details the problem faced in this dissertation.

3. Approach

   Explains what was implemented and what were our priorities.

4. State of the Art

   Presented technologies, tools and applications considered relevant.

5. Database

   Description of the database, including the database model and some code samples of the implementation.

6. Web Server

   Explanation of our server, comprising its RESTfull interface and the implementation of the most important algorithms.

7. Mobile Application

   Demonstration of the mobile application created by enumerating its activities and characteristics.

8. Performance tests

   Results obtained while testing our system.

9. Conclusions

   Enumerated conclusions, final remarks and future work.

Each chapter has a brief introduction and some also have a brief conclusion with final remarks.

2 Problem

In this chapter it is detailed the problem faced in this dissertation. The chapter starts by briefly presenting the agricultural domain and what were the main requirements to be addressed.

Afterwards are demonstrated several mathematical models used to calculate plants disease risk as well as all the data required for their implementation. This data can either be obtained or calculated, therefore it is discussed how some of the information will be obtained and how the other will be calculated, explaining additionally some empirical models.

2.1 Agricultural domain

In general the main places where agriculture activities take place are denominated farms, and these farms are usually divided into plots for better organisation.

Farms and therefore plots, can have multiple individuals working on them, from the farmers to the agronomists, all of them can be responsible for specific actions.

Usually, in each plot, some type of plant is installed. For example in a farm there can be 2 plots, one with tomato and another with grapes. Also, in some situations farmers might want to plant several types of plants in the same plot.

This dissertation focuses mainly on two types of important crops: grape vine and tomato.

2.1.1 Field notebooks

We follow closely the explanation of field notebooks found in [3].

Figure 2.1: Aerial view of a farm, exhibiting its plots. [2]

The agriculture production involves certain obligations and commitments that must be registered by farmers in a field notebook.

In these field notebooks, several records should be maintained regarding:

- Phenological states of the plant
- Observations of the main plant enemies
- Dates of treatments and plant protection products used
- Additional data, such as pruning, watering, fertilising and harvesting.

These field notebooks are specific for each crop. However they all require the same kind of information and records.

2.1.1.1 Plant phenological states

Phenology is the branch of ecology that studies the periodic phenomena of living beings and their relationships with environmental conditions such as temperature, light and humidity. [4]

In this case, phenological states are states regarding the growth rate of a plant. They are usually important to monitorize treatments and plant infections since some diseases and products allowed might be dependent of the current phenological state of a plant.

In field notebooks, farmers must record when the different phenological states occur and what is the current state when making other observations.

2.1.1.2 Plant enemies [7]

Plant enemies are organisms that can contribute to reducing quantitative or qualitative production of an agricultural crop, with inconvenient consequences for agriculture.

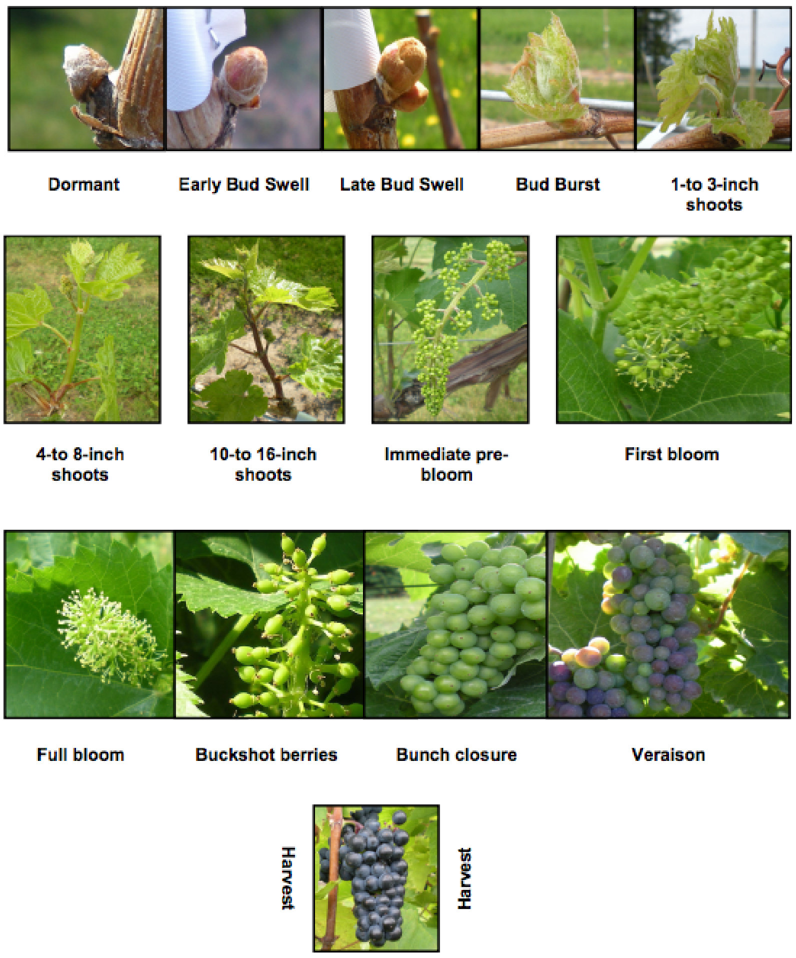


Figure 2.2: Grape vine phenological states. [5]

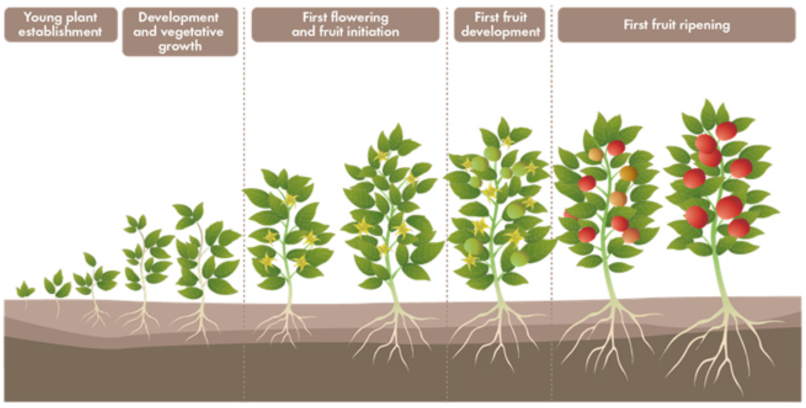

Young plant establishment
Development and vegetative growth
First flowering and fruit initiation
First fruit development
First fruit ripening

Figure 2.3: Tomato phenological states. [6]

They are usually referred to as "organisms that interfere with human activities and desires of human beings" or "organisms that live at the expense of agricultural plants causing more or less important losses."

The concept of plant enemy is influenced by three factors: the plant, the environment, and weather.

The importance of a plant enemy depends on the sensitivity of the crop to that organism and the economic value of the plant.

Environmental factors, including dryness or excessive humidity, wind and ultraviolet radiation have decisive influence on the importance of a plant enemy. Also, weather is essential in order to occur the most favourable environmental conditions and the most appropriate stages of crop growth and its enemies.

Plant enemies can be grouped into:

- Pests: Animal organisms such as dust mites, insects, molluscs and vertebrates.
- Diseases that can be caused by fungi, bacteria, viruses and more.
- Weeds : Plants that grow in places not desired.

In the field notebooks, farmers must record observations of enemies presence and other necessary information regarding them.

2.1.1.3 Plant treatments

There are many possible products that farmers can apply in plants. In the field notebooks, it is required for the users to record in which plot a product was applied, what was the plant enemy targeted, the date and the type of product.

The types of products can vary from:

- Insecticide: a pesticide used against insects.
- Acaricide: a pesticide that kills members of the Acari class, which includes ticks and mites.
- Fungicide: biocidal chemical compound or biological organism used to kill or inhibit fungi or fungal spores.
- Herbicide: a pesticide used to kill unwanted plants.

Plant treatments are essential to assure prevention of plant enemies and to cure them when infection already occured.

2.2 Plant diseases

Regarding plant enemies, this dissertation focused on five specific types of plant diseases. Two regarding grape vine, and three common fungal diseases for tomato.

These diseases have several empirical models showing the kind of required data and the necessary processing algorithms:

- Grape vine:
  - Powdery mildew
  - Downy mildew
- Tomato:
  - Early blight
  - Septoria leaf spot
  - Anthracnose

2.2.1 Grape powdery mildew

We follow closely the explanation of powdery mildew disease found in [8]. Powdery mildew is a disease that attacks all green organs of a vine. The disease is manifested by the appearance of translucent and oily stains on the upper side of the leaves and white areas coinciding with these stains on the underside.

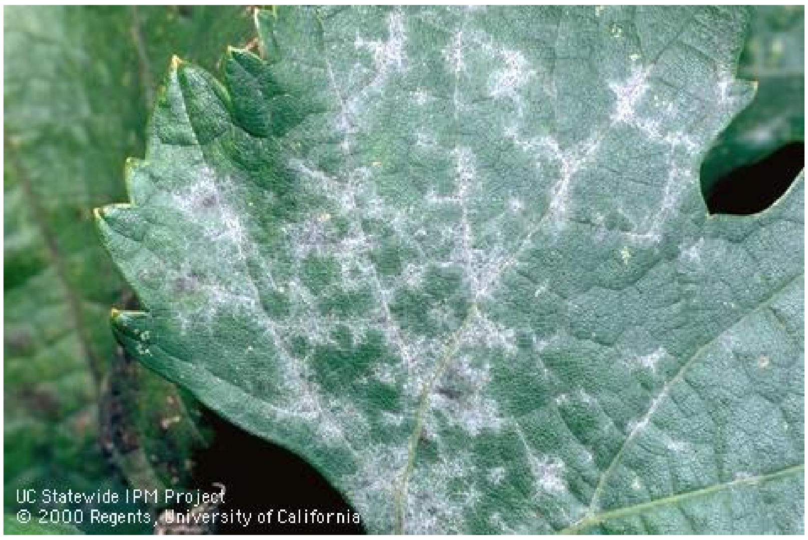


Figure 2.4: Grape plant affected with powdery mildew [9]

The main losses result from the attack of this fungus to inflorescences and berries. The fall of these leaves affect the sugar content of grapes and vitality of the strains.

Powdery mildew has been one of the major diseases in wine culture since over a century.

2.2.1.1 Model 1: UC Davis

This powdery mildew model is due to [10, 11], and a summary of the model can be found in [9] and in [12] from which we adapted the following description.

Environmental input variables:

- Daily average temperature
- Hourly leaf wetness duration
- Hourly average temperature

Stage 1

To determine disease infection risk levels, the model calculates the daily average temperature and measures the hours of leaf wetness required for heavy infection.

Table 2.1: Hours of wetting required for heavy infection in Model 1

Daily Average Temperature (°C)	Hours of Leaf Wetness required
5.8	40
5.9 - 6.3	34
6.4 - 6.9	30
7 - 7.4	27.3
7.5 - 8	25.3
8.1 - 8.5	23.3
8.6 - 9.7	20
9.8 - 10.2	19.3
10.3 - 10.8	18
10.9 - 11.3	17.3
11.4 - 11.9	16.7
12 - 13	16
13.1 - 14.1	14.7
14.2 - 15.2	14
15.3 - 16.3	13.3
16.4 - 16.9	12.7
17 - 24.1	12
24.2 - 24.7	12.7
24.8 - 25.2	14
25.3	17.3

For example, according to the model, at a daily average temperature of 12.5 °C, 16 hours of leaf wetness are required for heavy infection. See Table 2.1.

The temperature ranges in Table 2.1 have been adapted from the original table provided in [9], whose temperatures are specified in the Fahrenheit scale.

Once infection has occurred, the model switches to the risk assessment phase and is based entirely on the effect of temperature on the reproductive rate of the pathogen.

Stage 2

This stage starts when infection has occurred on stage one and afterwards when there are three consecutive days with six consecutive hours of temperatures between 21.1 and 29.4 °C.

It is based on a set of rules which depending on temperature variation, an index is incremented or decremented so that according to its value, one can verify when it is necessary to apply proper products.

1. Index starts at 60.
2. For each subsequent day where at least six continuous hours of temperatures between 21.1 and 29.4 °C occur, index increases by 20.
3. If there are less than six consecutive hours of temperatures between 21.1 and 29.4 °C, index decreases by 10.
4. If the temperature is 35 °C or higher for at least 15 minutes, index decreases by 10.
5. If on the same day with 6 continuous hours between 21.1-29.4 °C the temperature exceeds 35 °C for 15 minutes or more, index increases by 10.
6. Index is never lower than 0 or higher than 100.
7. In one day the index can't decrease by more than 10 or increase by more than 20.

Conclusions:

- Index of 30 or less indicates that a spray interval can be stretched to the label maximum of the product.
- Index of 40 to 50 indicates that a spray interval can be of intermediate length.
- Index of 60 to 100 indicates that there is high pressure for powdery mildew and spray intervals should be shortened to the label minimum.
- After treatment, the index is reset to zero.

2.2.1.2 Model 2: PMI

This powdery mildew model is due to [13], and like the other model, a summary of the model can be found in [9] from which we adapted the following description.

Environmental input variables:

- Daily high and low temperatures
- Precipitation

First dusting should occur twelve days after initial leaf appearance or 15 cm shoot growth.

Subsequent dustings should occur when the difference between the current PMI and the PMI on the last dusting date equals or exceeds 1.0.

When precipitation exceeds 0.25 cm, the vineyard should be re-dusted.

Like in previous models, the temperature ranges in Table 2.3 have also been adapted from the original table provided in [9] and converted to Celsius degrees.

Table 2.2: Daily indexes for high and low daily temperatures in Model 3: PMI

Low	**Daily high temperature — °C**										
	13-16	16-18	18-21	21-24	24-27	27-29	29-32	32-35	35-38	38-41	41-43
4-7	0.083	0.083	0.083	0.083	0.083	0.077	0.067	0.067	0.056	0.05 0	0.043
7-10	0.083	0.083	0.083	0.083	0.091	0.083	0.077	0.067	0.059	0.05 3	0.048
10-13	0.083	0.083	0.083	0.083	0.100	0.091	0.083	0.077	0.063	0.05 9	0.053
13-16	0.083	0.083	0.083	0.091	0.111	0.100	0.091	0.083	0.077	0.06 3	0.059
16-18	—	0.083	0.083	0.111	0 .111	0.111	0.100	0.091	0.077	0.071	0.067
18-21	—	—	0.100	0.143	0. 143	0.125	0.125	0.100	0.091	0.077	0.071
21-24	—	—	—	0.143	0.1 67	0.143	0.125	0.111	0.091	0.077	0.067
24-27	—	—	—	—	0.14 3	0.125	0.091	0.083	0.067	0.056	*
27-29	—	—	—	—	—	0.091	0.077	0.059	*	*	*
29-32	—	—	—	—	—	—	0.059	0.059	*	*	*
* The amount of product applied should be reduced to avoid excess leaf burn.											

2.2.2 Grape downy mildew

We follow closely the explanation of downy mildew disease found in [8] and [14].

In short, downy mildew is a highly destructive disease of grapevines across all wine regions of the world where it rains in the spring and summer with temperatures above 10°C.

Harvest losses in single years can be up to 100% if the disease is not controlled during favorable weather conditions.

Currently, there are no adequate sources of resistance in commercially acceptable varieties, causing fungicides to be the primary means of controlling the disease.

2.2.2.1 Model 1: 3-10 rule

This downy mildew model is due to [15], from which we adapted the following description.

Input variables:

- Air temperature
- Current phenological state
- Rainfall of last 48 hours

This empirical model is very simple. It verifies disease occurrence if there are simultaneous occurrence of the 3 following conditions:

1. Air temperature equal or greater than 10°C
2. Vine shoots higher than 10 cm. (Possible to verify on the current phenological state of plant)
3. There is at least 10 mm of rainfall in the past 48h.

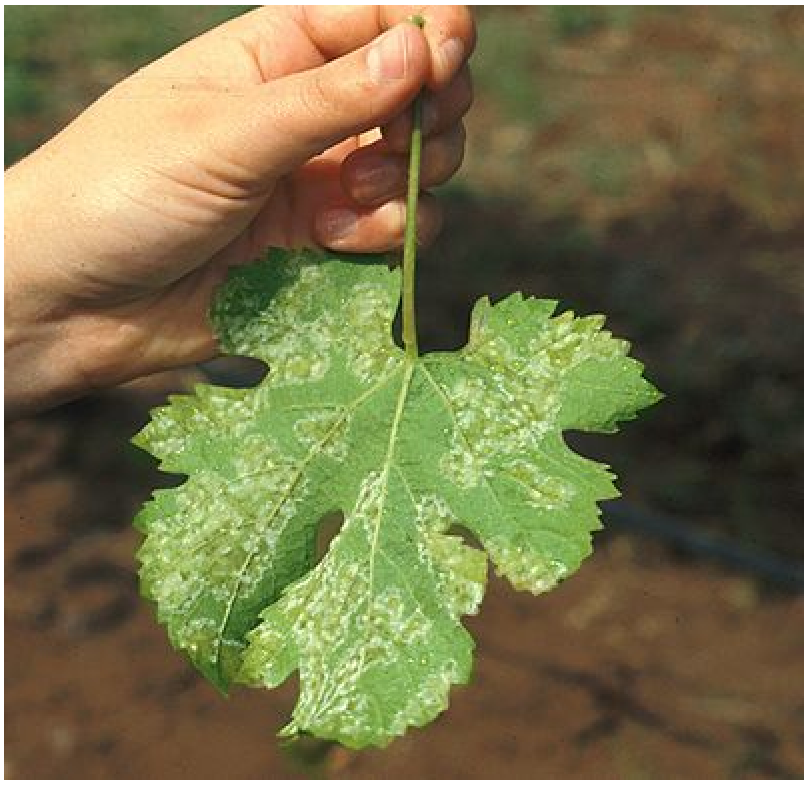


Figure 2.5: Grape plant affected with downy mildew [9]

2.2.2.2 Model 2: EPI

Like the other model, a summary of the model can be found in [15] from which we adapted the following description.

Input variables:

- Climatic monthly values of: rainfall, temperature, number of rainy days, nocturnal average of relative humidity
- Montly values of: rainfall and average temperature
- Decade (10 days) values of: rainfall and number of rainy days.
- Average diurnal relative humidity between 10:00 hours and 18:00 hours
- Average daily temperature

Climatic values are average values calculated for a relatively long and uniform period, being of at least thirty consecutive years.

EPI is based on two different equations: one expresses potential energy and the other kinetic energy.

Stage 1: Potential energy

Potential energy is calculated between 1 October and 31 March, with a time step of 10 days, based on the differences in air temperature and rainfall of the current year compared to the climatic average calculated over a 30-year period, according to the following equation:

$$\begin{aligned} Pe = & \left[2 * ct \left(\sqrt{Rm} - \sqrt{\frac{\underline{Rm} * 95}{100}}\right)\right] \\ & + \left\{0.2 * \left[\sqrt{Rm} * \sqrt{Tm} - \left(\sqrt{\frac{\underline{Rm} * 95}{100}} * \sqrt{Tm}\right)\right]\right\} \\ & - \left[\left(\frac{\underline{RDm} * 1.5}{18}\right) * \log \frac{Rd}{RDd}\right] \end{aligned}$$

Where,

- $Pe =$ potential energy
- $ct = 1.2$ in October - November; 1 in December; 0.8 in January, February and March;
- $\underline{Rm} =$ climatic monthly rainfall;
- $\underline{Tm} =$ climatic monthly temperature;
- $\underline{RDm} =$ climatic number of rainy days per month;
- $Tm =$ average monthly temperature;
- $Rm =$ monthly rainfall;
- $Rd =$ rainfall in the decade (10 days);
- $RDd =$ number of rainy days in the decade (10 days).

Stage 2: Kinetic energy

Kinetic energy is calculated every day between 1 April and 31 August, according to the following equation:

$$Ke = 0.012 * \left[\frac{\left(\frac{5 * \underline{RHm} + 3RHi}{8}\right)^2 * \sqrt{Ti} - RHi^2 * \sqrt{\underline{Tm}}}{100}\right]$$

Where,

- Ke = kinetic energy;
- $\underline{RHm}$ =climatic monthly nocturnal average of relative humidity;
- $\underline{Tm}$ = climatic monthly temperature;
- RHi = average diurnal RH between 10.00 hours and 18.00 hours;
- Ti = average daily temperature.

Summation of equations gives the EPI index as follows:

$$EPI = \sum_{October}^{March} Pe + \sum_{April}^{September} Ke$$

This model considers that the first seasonal infection occurs when $EPI > -10$.

2.2.3 Tomato diseases

Early blight is perhaps the most common foliar disease of tomatoes. This disease causes direct losses by the infection of fruits and indirect losses by reducing plant vigor [16].

Septoria leaf spot is one of the most common foliar diseases of tomato. It can be highly destructive given the proper conditions and has been known to cause complete crop failure. Although the causal fungus will not directly infect the fruit, losses are the result of defoliation which can lead to the failure of fruit maturation and sunscald of exposed fruit [17].

Anthracnose is a tomato disease that produces black, sunken lesions on the ripening fruit. Although symptoms do not appear until the fruit is ripening, the infection actually occurs when fruits are small and green [18].

2.2.3.1 Model 1: TOMCAST

This model is due to [19] from where we adapted the following description.

TOMCAST (TOMato disease foreCASTing) is a computer model model that uses local weather conditions to predict fungal disease development, on tomatoes, specifically Early Blight, Septoria Leaf Spot and Anthracnose.

Input variables:

- Hourly average temperature per day
- Hours of leaf wetness per day

The TOMCAST model index is determined by two factors, leaf wetness and temperature during the leaf wet hours. As the number of leaf wet hours and temperature increases, the index accumulates at a faster rate, i.e., increased disease pressure. Conversely, when there are fewer leaf wet hours and the temperature is lower, the index accumulate slowly if at all, i.e., decreased disease pressure. The table below shows the interaction between those two factors:

Table 2.3: Daily indexes for TOMCAST model

Average Temperature During Leaf Wet Hours	**Hours of Leaf Wetness per Day**				
13-17 degree C	0-6	7-15	16-20	21 +	
18-20 degree C	0-3	4-8	9-15	16-22	23+
21-25 degree C	0-2	3-5	6-12	13-20	21+
26-29 degree C	0-3	4-8	9-15	16-22	23+
Daily Index =	0	1	2	3	4

When the accumulated value of the index exceeds a pre-determined limit, the spray threshold, a fungicide spray is recommended to protect the foliage and fruit from disease development.

The spray threshold can range between 15-20. By following a 15 index spray threshold, a more conservative use of the TOMCAST system, a grower will apply fungicides more frequently than a grower who uses a 20 index spray threshold.

2.2.4 Models validation

Below, we discuss the validations made of the models previously described. However, it is not the purpose of this dissertation to validate these models, it is only important to know whether they can be efficiently used.

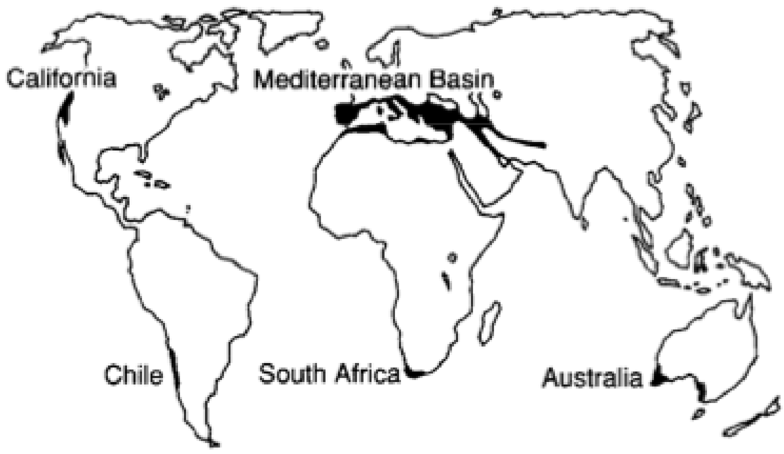


Figure 2.6: Areas with mediterranean climate [20]

2.2.4.1 Grape powdery mildew

Both the powdery mildew models were validated in multiple growing regions of California and the first model is actually being implemented in several grape-growing counties of the state [9].

Both Portugal and California have Mediterranean climate making their weather very similar, the difference is the fact Portugal summer heat is tempered by the Atlantic influence instead of the Pacific [20].

In Mediterranean climates, wine regions have long growing seasons of moderate to warm temperature and winters are usually warmer than those of maritime and continental climates. This leads to very small amounts of rain fall, requiring that farmers water grapes more often due to the increase of drought risk [21].

The first model is also currently being validated in New York, Washington, Oregon, Germany, Austria and Australia.

2.2.4.2 Grape downy mildew

The grape downy mildew models are already being used by a Portuguese institution called COHTN[1] (Centro operativo e tecnológico hortofrutícula Português). They also provide results of these models in their website.

2.2.4.3 Tomate lateblight

Both TOMCAST and BLITECAST were validated in the Region of Ribatejo from 2002-2005 [22], and shown appropriate with small changes. BLITECAST is a model adapted from TOMCAST for the potato late blight disease.

Currently, and according to our partnership in the PROTOMATE project, it was told to us that the TOMCAST model is currently being validated in Portugal by a major multinational company.

2.3 Data required

In order to be able to calculate diseases infection risk with the previous models, it is required to obtain weather values for various locations.

There are two types of data being used:

- Environmental
- Calculated

In our approach, environmental values are provided by the "Instituto Português do Mar e da Atmosfera", and these values can be obtained via:

- Satellite.

[1]http://www.cothn.pt/

- Weather stations spread throughout the country.

We also have online access to historical data from SNIHR [2] (Sistema Nacional de Informação de Recursos Hídricos), which is the portuguese national information system for water resources. They have meteorological information from their own weather stations of the previous 20 years. However these stations are no longer being maintained.

Additionally there might also be environmental values from farmers which have their own weather stations and want to use their values as input.

Calculated values are values that can't be obtained via satellite or from a weather station so it is necessary to estimate them using mathematical models. Currently, it is only required to calculate one value: leaf wetness.

2.3.1 Environmental

2.3.1.1 Satellite

The satellite that collects meteorological data is the EUMETSAT[3] Satellite, and this information is provided by the LAND SAF[4] , which is a system that provides analysis of land surface temperature. LAND SAF is also maintained by the IPMA.

Satellite information is obtained by satellite images where for each pixel of the image its possible to obtain the latitude, longitude and the corresponding value of the land surface temperature of an area of roughly a square of 4 by 4 kilometres, at the portuguese latitudes.

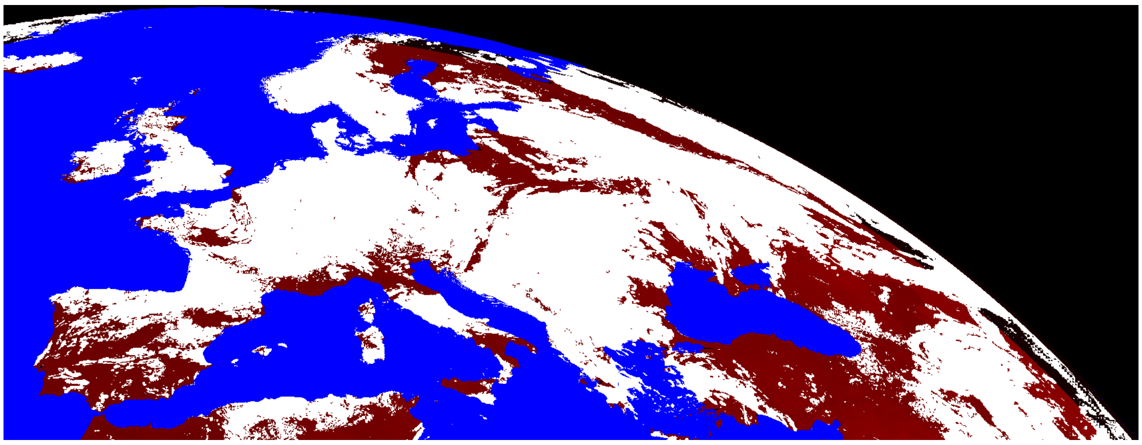


Figure 2.7: Example of a LST satellite image

In the figure 2.10, the colours represent:

- Black: Unavailable data

- Blue: Water pixels

[2] http://snirh.pt/
[3] http://www.eumetsat.int/
[4] http://landsaf.meteo.pt/

- White: Pixel covered with clouds, therefore value might be not available or inaccurate
- Red: Pixel with available land surface temperature value

The information is provided in files of HDF5 format, which can be easily read by software. Is is available in a FTP server hosted by our collaborators in Universidade de Évora.

Since we're working with satellite data, sometimes due to cloud coverage it might be impossible to obtain some values for a specific region. These lacks of information will be obviated by calculating some estimative of the temperature using spatial interpolation algorithms.

The satellite provides this information every 15 minutes, occurring 96 times per day. As of this moment its possible to obtain via satellite the following data:

- Land surface temperature at periods of 15 minutes

In the future, if other satellites or other types of environmental data are available, they can be easily included in our platform (e.g. precipitation).

2.3.1.2 Weather stations

There are a total of 146 weather stations from Instituto Português do Mar e da Atmosfera located in Portugal and all of them provide hourly data with values at its current location (See Table 2.4).

This information can be less accurate than information from satellite due to lack of maintenance and there are few less number of weather stations than locations where satellite data is available.

IPMA has some meteorological information from their weather stations in their website, providing data from the previous 5 days for any station.

2.3.1.3 Mobile phone sensors

One of the most expected features in mobile devices in the future is the integration with environmental sensors to detect values like temperature and relative humidity at the exact place where the device is.

Since it is planned the development of a mobile application, it would allow our system to have another information provider.

Android has recently introduced the new sensor API[5] for humidity and temperature, but after investigating and searching it was not possible to find any mobile devices on the market with those built-in sensors, and therefore implementation was not performed. It will however be considered in possible future contributions.

[5] `http://developer.android.com/guide/topics/sensors/sensors_environment.html`

Table 2.4: Information provided by each weather station.

Description	Unit
Year to which the information relates	-
Digit of the month to which the information relates	-
Day of the month to which the information relates	-
Time at which the information relates	-
Average air temperature at 1.5 m	°C
Maximum air temperature at 1.5 m	°C
Minimum air temperature at 1.5 m	°C
Average relative humidity	%
Maximum relative humidity	%
Average temperature of the wet thermometer	°C
Maximum temperature of the wet thermometer	°C
Minimum temperature of the wet thermometer	°C
Average dew point temperature	°C
Maximum dew point temperature	°C
Minimum dew point temperature	°C
Medium vapor pressure	hPa
Maximum vapor pressure	hPa
Minimum vapor pressure	hPa
Average wind direction	
Maximum wind direction	
Average wind intensity	m/s
Maximum wind instant intensity	m/s
Average air temperature at +0.05 m	°C
Maximum air temperature at +0.05 m	°C
Minimum air temperature at +0.05 m	°C
Average soil temperature at -0.05 m	°C
Maximum soil temperature at -0.05 m	°C
Minimum soil temperature at -0.05 m	°C
Precipitation	mm
Total global radiation	KJ/m2

2.3.2 Calculated

2.3.2.1 Leaf wetness

One of the most important input variables for the more complex mathematical models is leaf wetness.

Leaf wetness duration is the period of time during which free water – from dew, rainfall, fog, or irrigation - is present on the aerial surfaces of crop plants [23]. It is used for monitoring leaf moisture for agricultural purposes, such as fungus and disease control, for control of irrigation systems, and for detection of fog and dew conditions, and early detection of rainfall [23].

In order to obtain the best previsions possible for our mathematical models it is important to obtain a prediction if leaf wetness is present at each location.

Figure 2.8: Example of leaf wetness present on a leaf. [24]

To predict the presence of leaf wetness there are many mathematical models, however in this case we're limited to the weather data provided, so three usable models were found:

- Constant threshold
- Extended threshold
- CART/SLD/Wind model

A potential advantage of these empirical models is that they may be used to estimate leaf wetness from either on-site weather measurements, offsite or remotely estimated data, or both [25].

Constant threshold [26]

Constant threshold model is a simple model that only has one input variable: relative humidity (RH).

This method assumes that leaf is considered wet if RH is greater than or equal to a constant threshold. It was developed from observations that condensation on grass cover began before saturation in the air was reached, when relative humidity ranged from 91% to 99%.

Different values of the threshold have subsequently been tried being the most used RH = 90%.

Extended threshold [26]

Like the constant threshold model, this model only requires air relative humidity as an input.

The extended threshold approach considers hours in which RH is higher than 87% as wet hours. For values of RH between 70% and 87%, an hour is considered wet if RH is at least 3% higher than the RH of the hour before and dry if RH is at least 2% lower than the RH of the hour before. The hours in which RH is lower than 70% are considered dry.

If these conditions are not satisfied, the hour is considered the same as the previous one.

CART/SLD/Wind model [25]

CART/SLD/Wind is a nonparametric empirical model for estimating leaf wetness duration using classification and regression tree (CART) analysis with stepwise linear discriminant analysis (SLD). In order to calculate leaf wetness status it is required to obtain initial input data:

- Dew point depression (air temperature - dew point temperature)
- Wind speed
- Relative humidity

All this data is available via weather stations located nearby.

This model uses a CART Tree and in 2 situations it has to verify an inequality to calculate leaf wetness.

The inequalities are:

$$(1.6064\sqrt{T_{air}}+0.0036T_{air}^2+0.1531RH-0.4599Wind\times DPD-0.0035T_{air}\times RH) > 14.4674$$

(Inequality 1)

$$(0.7921\sqrt{T_{air}}+0.0046RH-2.3889Wind-0.0390T_{air}\times Wind+1.0613Wind\times DPD) > 37.0$$

(Inequality 2)

Where,

- T_{air} = Air temperature °C.
- RH = Relative humidity %.
- $Wind$ = Wind speed m/s.
- DPD = Dew point depression °C.

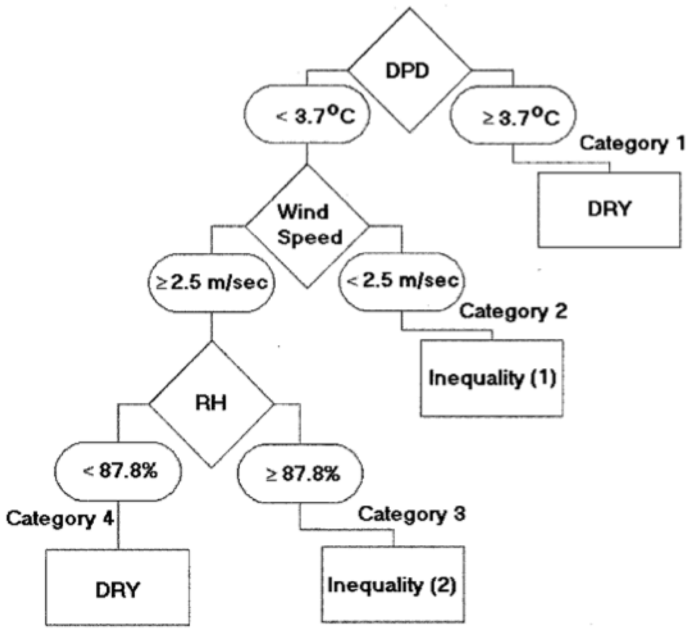


Figure 2.9: Classification tree for prediction of leaf wetness [25]

2.3.3 Spatial interpolation

Due to the fact information obtained is dependent on the location of weather stations and satellite accessible points, it becomes necessary to calculate approximate values for other locations.

Spatial interpolation is the process of using points with known values to estimate values of other points. It concerns a set of techniques designed to create continuous surfaces from sample points.

The techniques researched include deterministic methods of interpolation such as Nearest Neighbour, Inverse Distance Weighting, and the Kriging stochastic method.

Deterministic models assign values to geostatistic locations based on the surrounding measured values while stochastic methods are based on statistical models that include autocorrelation.

2.3.3.1 IDW (Inverse distance weighting) [27]

The inverse distance weighting method is a deterministic interpolation procedure (as opposed to a stochastic process) that uses a separate data set, typically in space.

The locations of unknown value are calculated using a weighted average of the inverse of the distance from that location to the location of known values.

It is commonly used in geographic information systems and geostatistics and therefore compared with other interpolation methods used in this area such as kriging interpolation or nearest neighbour.

2.3.3.2 Nearest neighbour [28]

Nearest neighbour interpolation is a deterministic method of interpolation where the estimated value is always equal to its nearest sample.

Due to its simplicity is regularly used for quick interpolations, and in areas well sampled.

2.3.3.3 Kriging [27]

Kriging is a stochastic regression method used in geostatistics to interpolate data.

It is similar to IDW in that it assumes that nearby points in space tend to have more similar values than points farther apart.

To do so it uses a linear combination of weights at known points to estimate values at unknown points. These weights change according to the spatial arrangement of the samples.

2.4 PROTOMATE project and data problems

"PROTOMATE - uma ferramenta de apoio à gestão da cultura do tomate para indústria" (PROTOMATE - a tool to support the management of tomato crop for industry) is a project that aims the development of a software tool to deal with a tomato pest called *Tuta absoluta*. It has the contributions of several different organizations:

- "COTHN - Centro Operativo e Tecnológico Hortofrutícola Nacional", a portuguese center for technological operations of fruits and vegetables.
- "ESAS - Escola Superior Agrária de Santarém"
- "ISA - Instituto Superior de Agronomia"
- "UE – Universidade de Évora"
- "FNOP- Federação Nacional de Organizações de Produtores", the national federation of farmer groups.
- LUSOSEM, an agriculture products company.
- And AGROMAIS,15 other farmer organizations.

Due to the similarities between my dissertation and the project, I had a research scholarship for the University of Évora, under guidance of Prof. José Rafael Silva to develop some functionalities on the platform so they could make data analysis with the data obtained via satellite and from weather stations.

Besides, there are also specific problems in our approach that need to be addressed. These will be specified in the following sections and the solution proposed for them will be presented.

2.4.1 Current issues

2.4.1.1 Temperature

Air temperature and land surface temperature are significantly different. Air temperature is usually the temperature measured at 1.5 meters of the ground, and land surface temperature is the temperature measured exactly at ground level.

These 1,5 meters can make a big difference when using temperature values in our empirical models since all models require air temperature and not land surface temperature.

Air temperature can be obtained via weather stations and land surface via satellite. However weather stations have weak maintenance and its values can't be entirely trusted.

For this, it is necessary to make an application in order to help create a correlation between these two values.

2.4.1.2 Humidity

Relative humidity is essential for disease models since it is one of the most important input variables and is required to obtain leaf wetness values.

At this moment there are several studies made which provide a way of obtaining relative humidity from dew point temperature and air temperature. However, like in the previous section, all these data types are only available from weather stations and so, a solution is required.

Based on current investigations, Prof. José Rafael Silva is researching a way of determining relative humidity based only on temperature. He conjectures that creating a relative humidity index can be important in estimating relative humidity values from the land surface temperature.

The formula is as follows:

$$RHIndex = \frac{MaxValue - MinValue}{\Delta t}$$

Where $MaxValue$ is the maximum value of land surface temperature verified in a day, and $MinValue$ is the minimum value respectively. Also, Δt is the time between occurrence of $MaxValue$ and the occurrence of $MinValue$ in hours.

2.4.2 Time-series segmentation

Time-series segmentation is the proposed solution by Prof. José Rafael Silva to try to solve our data problems and allow further analysis on the data.

It is a method of time series analysis, in which time the input series are divided into a sequence of segments, and produce a piecewise linear representation.

There are 3 possible segmentation algorithms[29]:

- Sliding Windows: A segment is grown until it exceeds some error bound. The process repeats with the next data point.

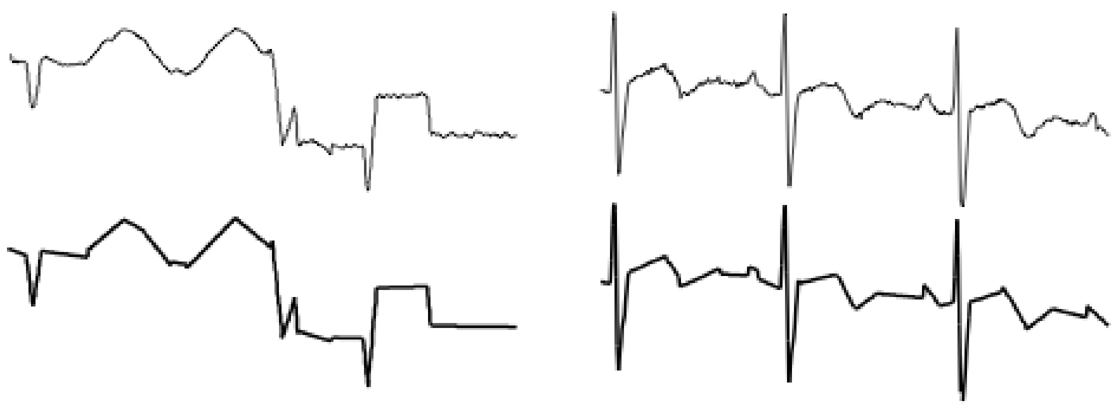


Figure 2.10: Two examples of time series segmentation. The figures above display the time series, and below its segmentation.[29]

- Top-Down: The time series is recursively partitioned until some stopping criteria is met.
- Bottom-Up: Starting from the finest possible approximation, segments are merged until some stopping criteria is met.

2.4.2.1 Solar calculations

Since we are working with time series and using temperature values, some other important information could be useful while analyzing our data.

For that we chose to calculate the time at which sunrise, sunset and solar noon occur at a specific day in a specific location.

We calculate these times using solar calculation formulas provided by NOAA/ESRL's Global Monitoring Division[6]. These formulas are found in [30].

Fractional year in radians:

$$y = \frac{2\pi}{365} * (day\_of\_year - 1 + \frac{hour - 12}{24})$$

Equation of time in minutes:

$$eqtime = 229.18 * (0.000075 + 0.001868 \cos y - 0.032077 \sin y - 0.014615 \cos 2y - 0.040849 \sin 2y)$$

Solar declination angle in radians:

$$decl = 0.006918 - 0.399912 \cos y + 0.070257 \sin y - 0.006758 \cos 2y + 0.000907 \sin 2y - 0.002697 \cos 3y + 0.00148 \sin 3y$$

For sunrise and sunset, the zenith is set to 90.833 degrees (the approximate correction for atmospheric refraction at sunrise and sunset).

[6]`http://www.esrl.noaa.gov/gmd/`

The hour angle in degrees:

$$ha = \arccos\left(\frac{\cos(90.833)}{\cos(lat)\cos(decl)} - \tan(lat)\tan(decl)\right)$$

Then, UTC for sunrise in minutes is:

$$sunrise = 720 + 4(longitude - ha) - eqtime$$

And sunset:

$$sunset = 720 + 4(longitude + ha) - eqtime$$

Also, it is possible to obtain the solar-noon in minutes:

$$solar\_noon = 720 + 4 * longitude - eqtime$$

2.5 Conclusions

In conclusion, this chapter viewed the main subjects to be addressed.

In the models of disease prevention, it is observed that the most important input values are temperature and duration of hours when leaf wetness occurred. To calculate leaf wetness, the main input value is relative humidity, being also possible to increase the accuracy of the results if wind speed and dew point temperature are known.

This data can be obtained from satellite and weather stations. However, at present, it is only possible to obtain the value of surface temperature from the satellite, being required to also obtain values from weather stations for the most complex models.

Finally it was also addressed the different types of interactions that must be recorded for users to fill their field notebooks. These interactions can be observations made of the current phonological stage of a plant, presence of enemies, treatments made and trap observations. These interactions can be also important for some enemy models, since some enemies can move from one crop to another, like *Tuta absoluta*.

3

Approach

This chapter describes every step of the development of the constructed system. It focuses in explaining what was implemented and what were the priorities.

3.1 Solution presented

Our solution consists of a system that obtains meteorological data, calculates the possibility of infection in cultures and at the same time allows interaction to users via mobile devices, recording treatments and possible culture enemies observed.

For this, it was developed:

1. A database to store all information.
2. Algorithms to obtain data from satellite and weather stations and store them in the database.
3. Implementation of leaf wetness and disease models, storing results in database.
4. Scripts that allow analysis of the temporal data in the system.
5. Web-services to provide results and ability to make queries to the database.
6. Mobile application for users to interact with the system.

Besides the goal of having a system that interacts with users and notifies them of disease risks, another major objective was to develop a platform that could supply all the requirements made and, at the same time, allow future integrations of other possible applications. An example of such applications are the data analysis functions that were developed.

In the Figure 3.1 the arrows represent the direction in which information flows. For example, satellite data is only sent to the system, while in mobile applications information can be received from the system (notifications) and can be sent from the mobile (user interactions).

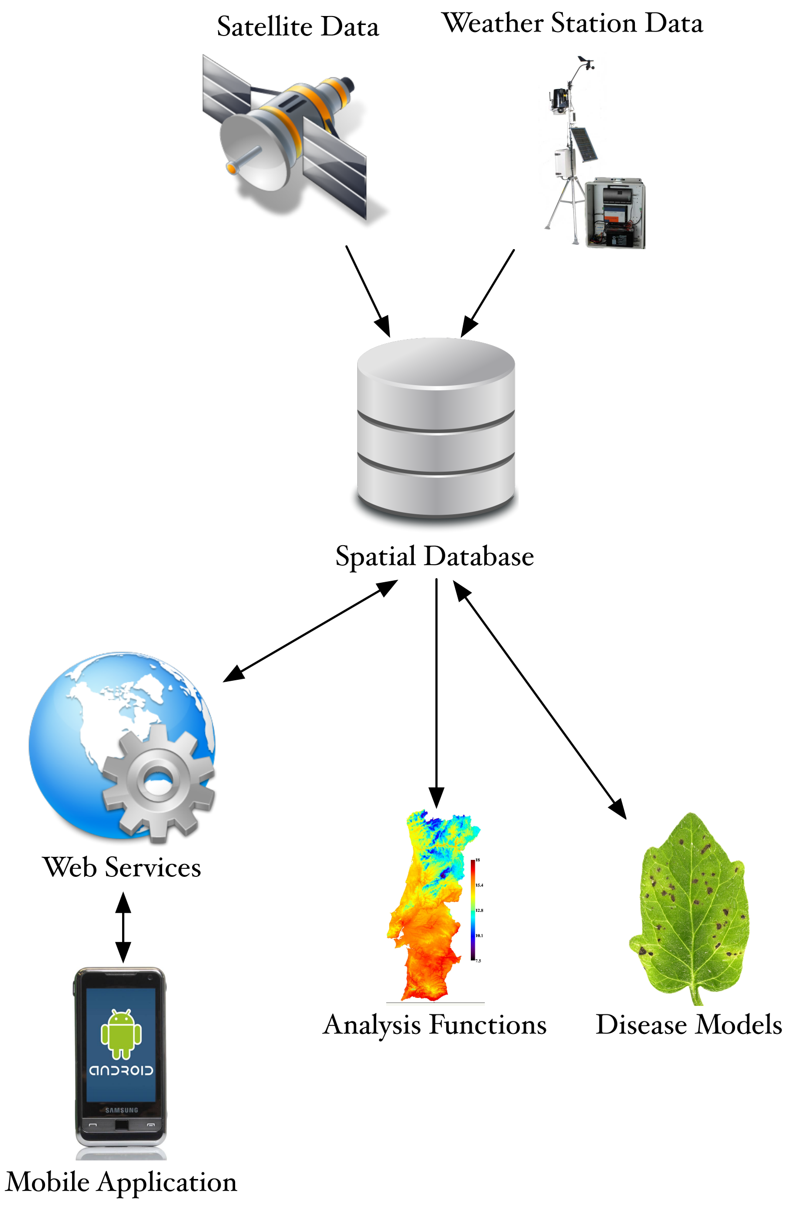


Figure 3.1: Architecture of the system

3.2 Database

A database was necessary to store all the data gathered by the system. This includes weather data, results from disease models, and observations made by the user.

Besides temporal data, our database will also store information regarding the agricultural domain. It includes all necessary information by the field notebooks, such as plants, plant enemies, users and their plots, traps, products, etc.

3.2.1 Database type

Since we are working with geographical information it was required to use a spatial database. A spatial database is a database that is optimized to store and query data that is related to objects in space, including points, lines and polygons.

While typical databases can understand various numeric and character types of data, additional functionality needs to be added for databases to process spatial data types. These are typically called geometry.

3.3 Web server

After having a database, it was required to develop an information platform that would interact with it, collecting weather information, calculating status of possible diseases and providing this information to the user.

For that, a server was built, which is always running and ready for incoming calls from mobile applications. Its main requirements are:

- Weather data acquisition - It should be able to obtain data from sources and store it in the database.
- Run plant enemy models - Should run daily models with information previously obtained.
- Web services - In order to connect with mobile application
- Other important applications - Extra scripts with access to the database

3.3.1 Weather data acquisition

Four scripts were created to download and insert meteorological data:

- A script to download recent HDF5 files from the FTP server and store the satellite data in the database.
- A script to download weather station data from previous 5 days in www.ipma.pt and insert it in database if not inserted already.
- A script to read from local text files containing satellite data and store it.

- A script to read from local Excel files containing weather station data and store it.

Whenever possible, it is required for the web server to run these scripts since there is new data every 15min for satellite and 1h for weather stations. However these could usually be done daily before executing plant enemy models since that is the only time the data is truly necessary.

Due to cloud coverage, sometimes the satellite can't provide the temperature of certain locations. To overcome this problem we use interpolation algorithms to predict these values in order to maintain accuracy of mathematical models.

The interpolation algorithm chosen to be implemented is the Inverse Distance Weight since it is algorithm used by the IPMA (Instituto Português do Mar e da Atmosfera).

3.3.2 Run plant enemy models

Usually on a daily basis our server executes the enemy models in order to calculate the risk of plant infection in all user plots.

To do so the models for disease prediction were implemented for all diseases addressed.

A generic architecture was created to allow plugging in easily other models for other diseases and pests in future contributions. This way the system provides scalability for every new model that is available to be inserted.

3.3.2.1 Calculating leaf wetness

While executing each model, if leaf wetness is necessary and not available, then it will be calculated.

The three models were chosen to be implemented. Depending on the available data, the system tries to use CART/SLD/Wind since it is the most accurate of the three.

All these calculated values are also stored in the database to help in future usages and/or statistical analysis.

3.3.3 Web services

In our server, we provide a web service for mobile applications to interact, allowing them to obtain information about a specific location and to manually enter data, such as specific stage of a plant according to the opinion of the technician and updated images of a plant with a possible disease.

The basic web methods available are:

- User login

  Users are able to log in the system and have an account associated.

- Get user farms and plots information

  Delivers all data regarding the user farms, plots and plants.

- Get model results

  Present model results from the current day for a specific plot.

- Receive information about user interactions such as:

  - Treatments made
  - Culture enemy observed
  - Insect traps observation

  These reflect the actions required in the field notebooks. They are always stored in the database.

  When submitting an interaction, the user must always supply the specific plot, the plant being analyzed, its enemy and the current phenological stage of the plant.

  The user might also add some additional text to enhance the observation made.

3.3.4 Analysis functions

Since the system will have access to the database there are inumerous possible functions that can be made.

Also, as previously stated, our collaborators in the PROTOMATE project asked us to implement some analysis functions using our database, so they could make posterior analysis on the results.

3.3.4.1 Time-series segmentation

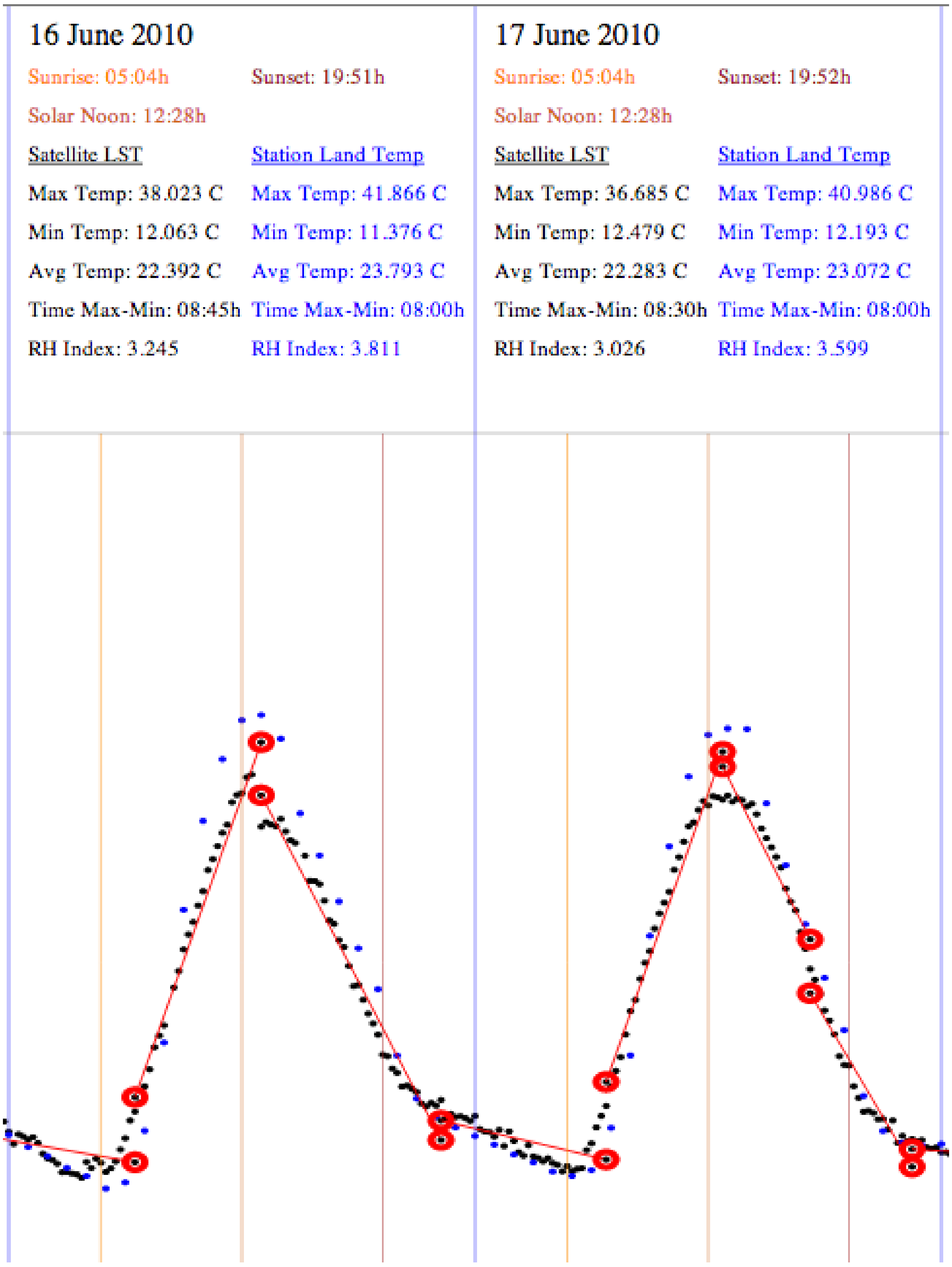
In order to solve these 2 problems, two scripts were created that based on data in our database, create an image with a linear representation of time series segments for two different data types from satellite and weather stations. One script is for the comparison between land surface temperature from satellite and air temperature at 1.5 meters from weather stations, and the other between land surface temperature from satellite and air temperature at 0.05 meters from weather station.

In the image, besides the segments, we also present some additional information of each particular day. First we display the time at which sunrise, sunset and solar noon occurred, displaying also vertical bars in our time series at each of these times.

Then, we present the maximum, minimum and average value of each time series. In addition, we also show the relative humidity index calculated and the time between the occurrence of the maximum value and the occurrence of the minimum value in hours.

The scripts also create 2 files each with values from the segments presented. These files are in the CSV format.

The future analysis of the results will be made by our colleagues at Universidade de Évora and based on their results, one future expected contribution will be to implement a way to obtain approximating air temperature from land surface temperature for a specific



16 June 2010

Sunrise: 05:04h	Sunset: 19:51h
Solar Noon: 12:28h	
Satellite LST	Station Land Temp
Max Temp: 38.023 C	Max Temp: 41.866 C
Min Temp: 12.063 C	Min Temp: 11.376 C
Avg Temp: 22.392 C	Avg Temp: 23.793 C
Time Max-Min: 08:45h	Time Max-Min: 08:00h
RH Index: 3.245	RH Index: 3.811

17 June 2010

Sunrise: 05:04h	Sunset: 19:52h
Solar Noon: 12:28h	
Satellite LST	Station Land Temp
Max Temp: 36.685 C	Max Temp: 40.986 C
Min Temp: 12.479 C	Min Temp: 12.193 C
Avg Temp: 22.283 C	Avg Temp: 23.072 C
Time Max-Min: 08:30h	Time Max-Min: 08:00h
RH Index: 3.026	RH Index: 3.599

Figure 3.2: Example of the linear representation of time series with Sliding Windows algorithm.

point at a specific time of the day. The same will be attempted to be made for relative humidity.

Preliminary findings show a strong correlation between land surface temperature and the air temperature at 0.05 meters, obtained via weather stations. These were possible to obtain due to our system.

3.4 Mobile application

Our primarily user interface is trough a mobile application for Android.

The development of a mobile application was important for farmers and agronomists to use in order for them to request the results from our disease models and allow them to manually submit interactions such as plant treatments, disease observations and plant phenological stages.

However, due to lack of time, the application created is only a prototype and requires future modifications in order to be fully functional. We focused on allowing the user to see their farms and plots, and for each plot check the model results and submit some interactions he wishes to make.

The requirements were:

- Native application for a mobile device.
- Show user information regarding its farms and plots.
  - He should be able to see its farms, plots, plants, traps and interactions made.
- Show user the models results for its plots.
- Allow the user to submit user interactions to the server. These interactions could be:
  - Add disease observation
  - Add product treatment made
  - Add trap observation
  - Add an observation of a plant current phenological stage

Internet access is required to run this application since several connections are made constantly in every operation made by the user.

4 State of the Art

In this section are presented technologies, tools and applications that were considered to be used in the development of the proposed solution.

First, possible database management systems are reviewed, focusing on the important features for the implementation of this work, in particular their ability to store and process geographic information.

Then, various options for the implementation of our information systems are discussed, briefly detailing their interaction capabilities with our data.

Finally, a summary is made on possible technologies for the implementation of our mobile application.

4.1 Database

There are several database management systems that support geographical data. However for this project is necessary to focus on open source software, noticeably:

- PostgreSQL
- MySQL
- SQLite

Another requirement is the support for geography types (lat, long) since all our information will be related to specific coordinates of the territory.

4.1.1 Possible options

4.1.1.1 PostgreSQL

PostgreSQL[1] is an Open Source database management system able of handling large sized databases.

Today PostgreSQL DMBS is one of the most advanced Open Source DBMS, with features such as: complex queries, foreign keys, transactional integrity, multi-version concurrency control and an extension to store geographical data called PostGIS.

PostGIS

PostGIS[2] is an open source software library that allows PostgreSQL to be used as a backend spatial database for geographic information systems, by adding support for geographic objects in its object-relational database.

Some of its main features are the addition of geometry types for points, multi-points, line strings, multi-line strings, polygons and multi-polygons.

PostGIS supports both geometry (x, y) and geography (lat, long) types and functions.

It also has very usefull functions for calculating distances between locations using different coordinate systems.

4.1.1.2 MySQL

MySQL[3] is a database management system (DBMS), which uses the SQL language (Structured Query Language) as an interface.

It is currently one of the most popular database systems, with over 10 million installations worldwide.

Some of the great advantages of using MySQL are its portability, compatibility, excellent performance and stability, little demand for hardware capabilities and ease of use.

MySQL supports a multitude of geometry types, but only on a 2D plane. The geography type systems (lat, long) are not implemented, making it an option not viable for our system.

4.1.1.3 SQLite

SQLite[4] is a C language library that implements an SQL database embedded. Programs that use the SQLite library can have access to SQL database without running a separate RDBMS process.

SQLite is not a client library used to connect to a big database server, but the server itself. The SQLite library reads and writes directly to and from the database file on disk.

[1] `http://www.postgresql.org/`
[2] `http://postgis.net/`
[3] `http://www.mysql.com/`
[4] `http://www.sqlite.org/`

Spatialite

Spatialite[5] is a variant of the SQLite database that provides GIS features. It uses the GEOS, PROJ.4, etc... libraries that are also used in PostGIS, and hence the resulting feature set is similar.

Only geometry types (2D x,y operations) are supported, making the lack of support for geography projections (lat, long) one major issue like MySQL.

However, since mobile operating system Android uses SQLite databases on the device, Spatialite might be useful for implementation of other functions.

4.1.2 Database chosen

The database system chosen is PostgreSQL with PostGIS.

PostGIS allow us to associate satellite information to specific points and relate the areas of plots to polygons, or single points, making it possible to take advantage of these point coordinates for possible data interpolations required to obtain missing values or required values in specific locations.

4.2 Web server

In the server-side there are inumerous options. In order to ease this decision, it was opted to develop using Java programming language and its SDK technologies since it is the language the candidate had most experienced with and it has plugins/libraries for implementation of all the requirements.

4.2.1 Apache Tomcat

Apache Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF).

It will be used to start the system being always available for any incoming connection.

4.2.2 JAX-RS and Jersey

The web-services in our server were made in a RESTful API, using JAX-RS for it's development along with Jersey.

JAX-RS: Java API for RESTful Web Services is a Java programming language API that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.

Jersey is a reference implementation of JAX-RS, which implements support for the annotations defined in JSR 311, making it easy for developers to build RESTful web services by using the Java programming language.

[5] `http://www.gaia-gis.it/gaia-sins/`

4.2.2.1 Jackson

Also a library called Jackson was used to allow parsing of JSON objects received from the mobile application and being sent to it.

Jackson is a multi-purpose Java library for processing JSON data format. With the extension Jackson-JAXRS-JSON it is possible to handle JSON input/output for JAX-RS implementations (like Jersey and RESTeasy) using standard Jackson data binding.

4.2.3 PostgreSQL JDBC

PostgreSQL JDBC allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

It provides a reasonably complete implementation of the JDBC 3 specification in addition to some PostgreSQL specific extensions.

4.2.4 HDF Object Package

The HDF Object Package is a Java package that provides an object-oriented interface to HDF data objects. The package offers a common API to access both HDF4 and HDF5 files.

It allows the development of easy routines to read the HDF5 files with temperature values from the satellite and store them in the database.

4.3 Mobile application

Our primarily user interface will be trough a mobile application.

4.3.1 Operating system

There are 3 main operating systems to develop mobile applications:

- Android
- iOS
- Windows Phone 8

4.3.1.1 Android

Android[6] is an operating system based on Linux for mobile devices. It is developed by the Open Handset Alliance led by Google and other companies.

According to Google, more than 1 million 300 thousand devices with this operating system are activated every day.

[6] `http://www.android.com/`

4.3.1.2 iOS

iOS[7] (formerly iPhone OS) is a mobile operating system from Apple Inc. originally developed for the iPhone.

Interaction with the OS includes gestures such as just tapping the screen, slide your finger, and the movement of "tweezers" used to enlarge or reduce the image.

4.3.1.3 Windows Phone 8

Windows Phone 8[8] is the second generation of Windows Phone mobile operating system from Microsoft.

Nokia announced a partnership with Microsoft in February 11, 2011 becoming its main operating system.

4.3.1.4 Operating system chosen

As far as operative systems, Android was chosen due to being the most popular operative system for mobile devices and for being the simplest operative system to develop applications.

Android phones are cheaper than iPhones and a previous poll to an “Associação de Produtores” led us to conclude that there is a preference for recent Android phones.

Windows Phone 8 is still new in the market so it will be dropped as third and last option.

In addition, the candidate also had experience in developing applications for Android, making it the most viable option.

4.4 Conclusions

In conclusion, several technologies were chosen to be used for the implementation of our project.

For the database, we opted for PostgreSQL with PostGIS. In the server we chose Apache Tomcat server and the creating of a RESTfull API with Jersey. Lastly, the mobile application is made for the Android operating system.

[7]http://www.apple.com/ios/
[8]http://www.windowsphone.com/

5 Database

In this chapter a detailed explanation of the database implementation is described.

First, the database diagrams are showed and briefly explained, then we focus on specific code for database creation and interaction.

5.1 Database model

The database can be divided in 2 sections for better understanding:

- Geographical and temporal data
- User interactions

5.1.1 Geographical and temporal data

In this section, it is presented how the database was structured to store temporal data from any kind of sources. The following is a summary of each table and its relationships:

5.1.1.1 Organizations

These are simple organizations that provide the data to be stored in the database. For example, in our case the organization that provides both satellite data and meteorological data is IPMA.

5.1.1.2 Source Types

Like organizations, our sources will also have source types. These can be for example "Satellite" or "Weather Station".

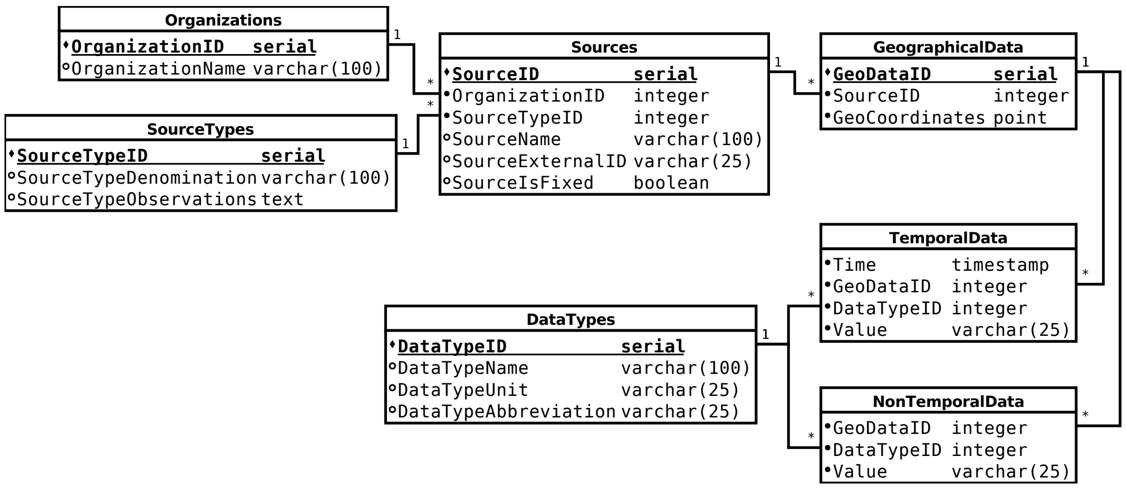


Figure 5.1: Database diagram of geographical and temporal data.

5.1.1.3 Sources

Sources are where the data comes from. For example a source can be one weather station like "Évora", "Santarém", etc. and can be a satellite like "LSA SAF" which is the satellite that provides land surface temperature.

Sources can have an organization, a source type, a flag to check if the source is fixed and one external ID to be identified externally. Weather stations from IPMA have their own external codes and those are used to retrieve its data.

5.1.1.4 Geographical data

Geographical data are the locations in our system. Each location will have coordinates represented by the POINT type in Postgis. The POINT type is a geometry type which includes the latitude and longitude of the location, as well as the coordinate system. In this project all points use the WGS84 coordinate system, which is the reference coordinate system used by the Global Positioning System (GPS).

Each geographical data must belong to one source, and one source can have multiple geographical datas. This is important since a weather station is a fixed source and therefore will have only one location, while a satellite can provide data from multiple locations.

5.1.1.5 Data types

Data types are the types of information that can be inserted in the database, provided by the sources.

These can have any value like "Air Temperature" or "Wind Speed", a type unit such as "C (Celsius)" or "m/s (Meters per second)" and an abbreviation like "Air Temp".

One important factor is in case more sources are integrated in the future with new data types, all that is necessary is to add those data types into the table so the system can store its data.

5.1.1.6 Temporal data

This is the table where all the temporal, and in particular meteorological data is stored. These are georeferenced time series [31].

Each record has a timestamp representing the time when the source verified its value, a geographical data so it is possible to know what is the location where the value belongs and therefore its Source, a DataType so we know what type is that value from, and the value itself.

This value is not necessarily a number since all kinds of data types can be stored in the database. For instance weather stations provide data from the data type "Wind Direction" which is represented by 1 to 2 letters such as : "N", "SW", "S", "W", "NW", etc.

5.1.1.7 Non temporal data

This table is exactly like temporal data but without a time value. This means it stores static information regarding the geographical data.

An example can be if one wants to store the altitude of a geographical data. A data type for "Altitude" can be created and this information can be stored in the non temporal data table regarding the respective geographical data.

5.1.2 User interactions

This section elaborates the information regarding users, their farms and plots and the interactions they make with the system trough the web application.

5.1.2.1 Users and farms

The users table stores the users that interact with our system. Each user has a username, a password, an email, a phone number and an address.

Also farms will be stored in a table, containing a farm address, parish, county, district and its regional directorate of agriculture (DRA).

The relationship between users and farms is N-to-N, since a user can work in/own multiple farms (for example if he is the owner of multiple farms) and a farm can have multiple users working at it (owners, agronomists, etc). This relationship is stored in the UserFarms table.

5.1.2.2 Plots

Farm's plots are stored in a table where each plot has its plot name and plot area size. A plot can belong to only one farm. Also, a plot has a SourceID which means it will also be related to the sources table previously mentioned in the geographical data section.

This means that a plot is a source which will also have multiple geographical data and therefore locations. These locations represent the plot and will be used to store the results from the disease models and other data related to each plot.

5.1.2.3 Plants and phenological stages

Plants are stored in their own table which will have the common plant name. Each plant also has several phenological stages and these are stored in a side table.

Finally, plants and plots will be related in a N-to-N relationship, where one plot can have multiple plants in it, and a plant may be in several plots.

5.1.2.4 Enemies and plant enemy models

The enemies table will store all kinds of enemies that can harm a plot and its plants. Each enemy will have its name and its type which can be either a pest, a disease or a weed.

Enemies also have observations which are specific observations made by the users. For example an enemy can have different stages of observation, and when the user interacts with the server from the mobile application, he identifies the enemy and what is the observation being made. These are stored in the table EnemyObservations.

In case an enemy is a disease it has a relationship with the plants in the PlantEnemies table, since a plant can have several enemies and one particular enemy can be harmful to many plants.

These plant enemies can also have models. These are the models that will be executed to verify the infection risk of an enemy in all plots of the system.

A table was created for models, relating them to the plant enemies, storing their name and a code. This code is used by the system to identify which model algorithms to run and store the model results in the temporal data table.

5.1.2.5 Insect traps

There is a table which stores all kinds of traps available, which are identified by their name.

These traps will have an N-to-N relationship with plots in the table PlotTraps. This table will store the trap, its plot, and the trap location inside the plot. This location is of the point type just like the locations stored in the geographical data database.

5.1.2.6 Products and active substances

Products that can be applied to plants are stored in a table on our database. These products are identified by their name and can have many substances active.

The product substances are also be stored in their own table. The relationship between products and substances is N-to-N in the table ProductSubstances.

5.1.2.7 User interactions

User interactions are the interactions made from the mobile application with the server. Every time a user wants to make an observation of an enemy on a crop, the number of pest occurrences on a trap or a plant treatment that has been made, an user interaction is created.

Each record of the UserInteractions table has the time of the interaction specified by the user, the user, the enemy, the plot, the plant and its current phenological stage and optionally a photo which can be uploaded from the mobile device and an extra field for possible observations.

This table is in a ISA relationship with other three tables:

- UserEnemyObservations
- UserTreatments
- UserTrapObservations

This means that every time a record is created in one of these 3 tables the respective record must be created in the UserInteractions table.

These three tables represent more specific interactions, and require other particular attributes.

User enemy observations

This table stores the enemy observations made and besides the data stored in the record on user interactions table, it also stores the enemy observation, the zone where the enemy is being observed (usually is either "central zone" or "border") and the number of plants observed.

The field observations on user interactions table is used to specify the status of the enemy observed.

User treatments

This table stores the product treatments made and besides the data stored in the record on user interactions table, it stores the product used, the dosage and its security interval.

The field observations on user interactions table is used to make additional comments on the user treatment.

User trap observations

User trap observations stores only one field besides the required in user interactions table: the trap observed.

The field observations on user interactions table is used to specify the status of the trap observed.

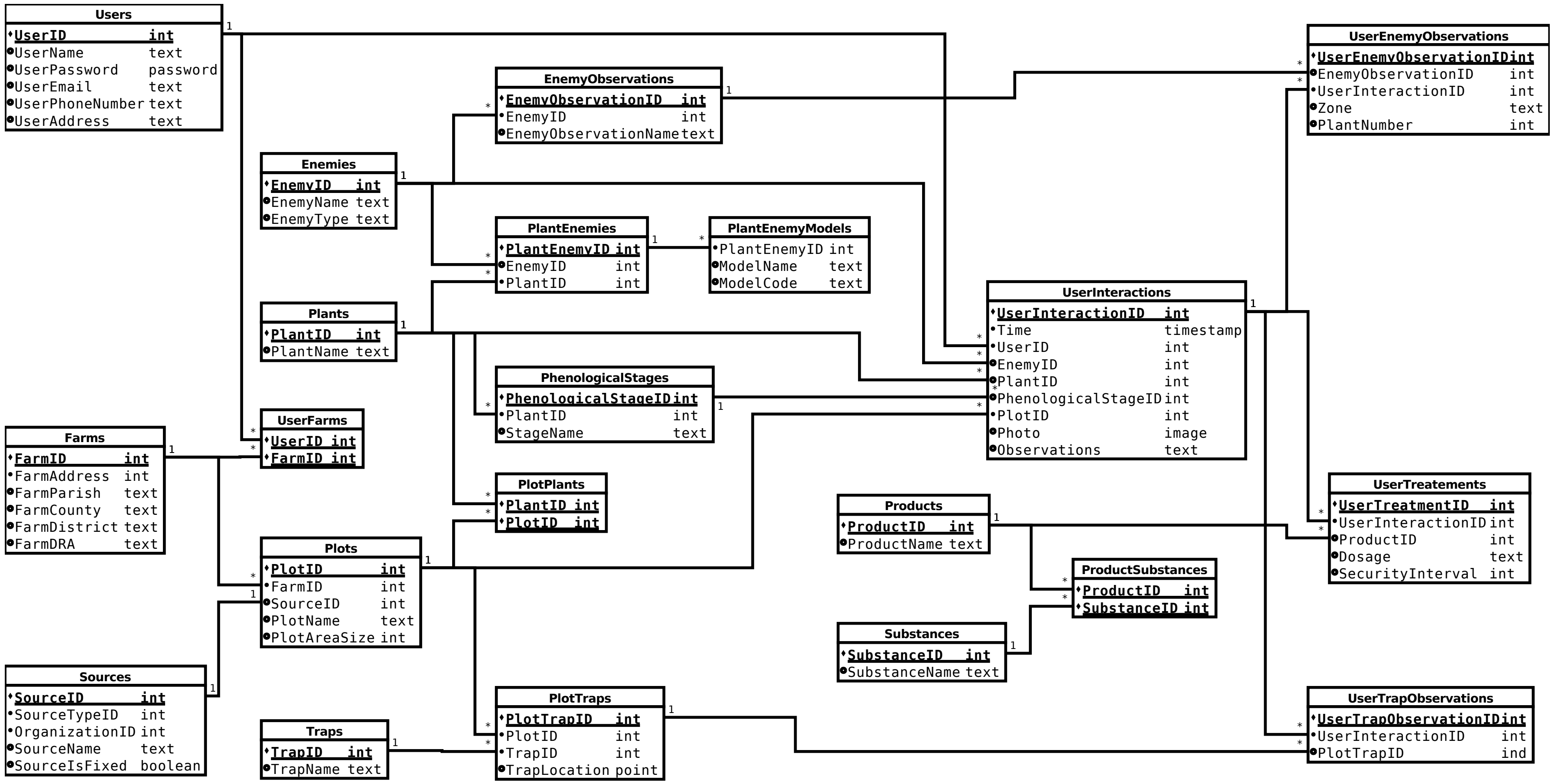
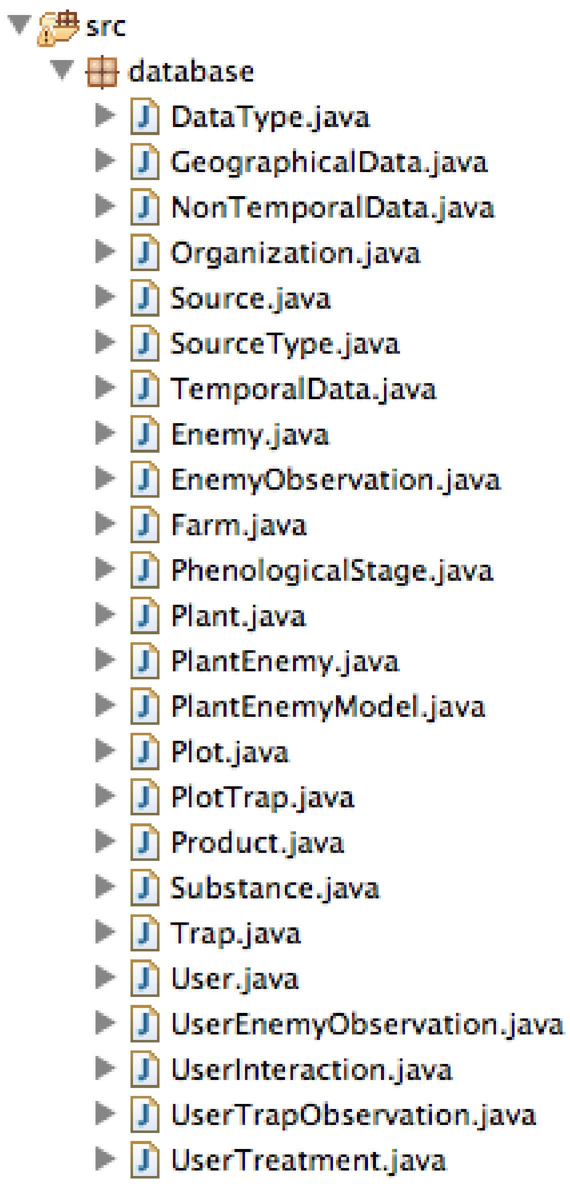


Figure 5.2: Database diagram of user interactions.

5.1.3 Object-relational mapping

In order to allow the web server and analysis functions to interact with our database, an object-relational mapping was created in Java programming language.

Each table represented earlier has its own Java class with its own private attributes and specific methods that create SQL statements to be executed in our database system.



```
src
  database
    DataType.java
    GeographicalData.java
    NonTemporalData.java
    Organization.java
    Source.java
    SourceType.java
    TemporalData.java
    Enemy.java
    EnemyObservation.java
    Farm.java
    PhenologicalStage.java
    Plant.java
    PlantEnemy.java
    PlantEnemyModel.java
    Plot.java
    PlotTrap.java
    Product.java
    Substance.java
    Trap.java
    User.java
    UserEnemyObservation.java
    UserInteraction.java
    UserTrapObservation.java
    UserTreatment.java
```

Figure 5.3: Java classes created for object-relational mapping in our Eclipse project.

All classes have methods for inserting, updating and selecting from the database. Besides these generic methods, some specific ones were also created. These are some examples:

5.1.3.1 GeographicalData

- getDistance(double, double)

  Returns the distance from the the current geographical data to another location.

- getDistances(List<Integer>, int)

  Returns the distances from the current geographical data to a list of locations.

- getDistancesToLocation(double, double, Object[], int, double)

  Returns the minimum distance from one location to a list of locations in the database.

- getNearestGeoDataBySourceType(double, double, int)

  Returns the nearest geographical data of the same source type.

- getNearestGeoDataBySource(double, double, int)

  Returns the nearest geographical data of the same source.

5.1.3.2 TemporalData

- insertAll(List<TemporalData>)

  Inserts a list of TemporalData in the database.

- getLatestDailyTemporalData(int, int, Calendar)

  Returns the latest TemporalData in a specific day of a specific geographical data and datatype.

- getLatestTemporalData(int, int)

  Returns the latest TemporalData of a specific geographical data and datatype.

- getTemporalDataInInterval(Calendar, Calendar, int, int)

  Returns all TemporalData in a time interval of a specific geographical data and datatype.

- getTemporalDataInInterval(Calendar, Calendar, int)

  Returns all TemporalData in a time interval of a specific datatype.

- getTemporalDataExactTime(Calendar, int)

  Returns all TemporalData in a specific time of a specific datatype.

- getTemporalDataExactTime(Calendar, int, int)

  Returns all TemporalData in a specific time of a specific geographical data and datatype.

5.2 Implementation

The following sections include specific parts of our database creation. Some Java code will be displayed as well as some SQL of some statements used.

5.2.1 Connecting to the DB

As mentioned in State of Art, we use the PostgreSQL JDBC (Java Database Connectivity) which allows us to connect to the database and execute statements in our Java classes.

For this a main class called DB was created to ensure connectivity to the database. It reads the database properties such as url, username and password from a file and opens a connection.

The connection is opened with the Java code:

```
Properties props = new Properties();
FileInputStream in = new FileInputStream("database.properties");
props.load(in);

String url = props.getProperty("db.url");
String user = props.getProperty("db.user");
String passwd = props.getProperty("db.passwd");
Connection con = DriverManager.getConnection(url, user, passwd);
```

Every time we need to execute a statement, first we get the connection from the method getConnection() in the DB class. When issued, if the connection isn't open it opens automatically.

In order to execute it, first the statement must be created, allocated its parameters and in the end executed. This is an example of how such can be done:

```
Connection con = DB.getConnection();

PreparedStatement statement = con.prepareStatement("SELECT␣datatypeid,typename,
    typeunit␣FROM␣datatypes␣WHERE␣datatypeid␣=␣?");
statement.setInt(1, datatypeidvalue);

ResultSet result = statement.executeQuery();
if (result.next()) {
  DataType type = new DataType();
  type.dataTypeID = result.getInt(1);
  type.typeName = result.getString(2);
  type.typeUnit = result.getString(3);
  }

DB.disconnect();
```

In the end it is always necessary to disconnect from the database so the JDBC releases its resources.

5.2.2 Geographical data

Like specified in the previous chapter, geographical data are created using the geometry type POINT in Postgis.

5.2.2.1 Table creation

To create this column in a table, it is required to execute a command when the table already exists:

```
SELECT AddGeometryColumn('geographicaldata','geocoordinates',4326,'POINT',2);
```

It adds a geometry column to an existing table of attributes. In this case, 'geographicaldata' is the name of the table, and 'geocoordinates' the name of the new column. The value 4326 is the value reference of the WGS84 coordinate system, the 'POINT' uppercase string corresponds to the POINT geometry type and the last value 2 means it is in 2 dimensions (lat + long).

5.2.2.2 Insert

To insert a value we used the ST_GeomFromText command:

```
INSERT INTO geographicaldata(sourceid,geocoordinates) VALUES(5,ST_GeomFromText(
    'POINT(-8.5␣40.8)' , 4326));
```

This command constructs a PostGIS ST_Geometry object from the OGC Well-Known text representation. The text representation for points expects first the longitude (-8.5) and then the latitude (40.8). It is also required to specify the coordinate system reference.

5.2.2.3 Read

Just like while inserting we had to transform the text representation to the geometry object, when reading it is necessary to do the opposite.

For this, in order to select values and read their latitude and longitudes we used 2 commands: ST_X() and ST_Y:

```
SELECT geodataid,sourceid,ST_X(geocoordinates),ST_Y(geocoordinates) FROM
    geographicaldata
```

ST_X returns the X coordinate of the point and ST_Y returns the Y coordinate. Both commands require the input to be a point, and they return NULL if it isn't available. The X coordinate represents the point's longitude and Y the latitude.

5.2.2.4 Distances

Another very useful command in PostGis is ST_Distance which returns the distance in meters between two points.

This is particularly used for spatial interpolation in order to obtain all the distances from known points to the point we wish to interpolate.

```
SELECT ST_Distance(ST_GeomFromText('POINT(-8␣40)',4326),␣ST_GeomFromText('POINT
    (-7 50)',4326),true)
```

It returns the distance between two geographies in meters. In this case both points are created with the ST_GeomFromText command and the true value represents that the command will return the spheroidal minimum distance instead of a 2-dimensional cartesian minimum distance.

5.2.3 Temporal Data

Each .HDF5 file has around 6000 locations for Portugal which means it also has 6000 values to insert in the database. In order to maximise the efficiency while inserting these values it was preferable to insert all values at once instead of one by one. For this we used the Postgres' COPY statement.

COPY FROM copies data from a file to a table (appending the data to whatever is in the table already), so in order to use COPY it is required to create a CSV file with the values we plan to insert.

Technically, it is created a buffer that contains the data to be inserted and, instead of saving the file to disk, we add the parameter STDIN to the statement which means the input comes from the client application.

Then to insert it is only required to execute the statement:

```
COPY temporaldata FROM STDIN WITH CSV
```

Where `temporaldata` is the table where the data will be inserted, and WITH CSV means the input is of the file type CSV.

5.2.4 Indexes

Since we will constantly be inserting meteorological data in our TemporalData table, eventually we will have millions of rows. For example, every 15 minutes we will insert around 6000 rows of data just from satellite.

In order to allow quick data access when necessary, we created indexes on several columns.

5.2.4.1 Time and data types

The most common queries to the system will be specially related to time and data types, such as selecting all air temperature values in one particular hour or in an interval.

So, an index was created with both columns to make data access faster:

```
CREATE INDEX timedatatype
  ON temporaldata
  USING btree
  ("time" DESC NULLS LAST, datatypeid);
```

We sort time descending since more often we will want to obtain the results from the previous day.

5.2.4.2 Time and geographical locations

However, in some particular cases for our analysis functions, we also make queries related to the geographical locations. Often we want to get all data from a specific location in an interval of time.

So, another index was created, this time with the geodataid column:

```
CREATE INDEX timegeodata
  ON temporaldata
  USING btree
  ("time" DESC NULLS LAST, geodataid);
```

Both these indexes allow faster queries to the database and minimise the time necessary to run our daily enemy models. They are used automatically by the database management system whenever it founds it is worth using them.

If these indexes weren't created, the database management system would have to search sequentially the entire table, often taking hours just to get all data for a specific given time.

5.3 Conclusions

In short, we have created a database that allows all the required information to be stored and to be easily organised. It also enables future insertion of new types of data in our system, as long as they are geo referenced.

6 Web server

In this chapter we provide analysis of the web server implemented.

First, we talk about the web server, including it's RESTfull API and the hierarchy of the enemy models. Then we describe some specific implementation components including the meteorological data insertion and models execution.

6.1 Modelling

In this section it is presented our REST API that was developed to connect with the mobile application, and the structure used for the enemy models that were implemented in our system.

6.1.1 REST API

A RESTful web API was created to receive requests from the mobile application. It is divided in several entities, each with its own specific actions available. These are some of the main web methods implemented:

Url	Type	Description
/rest/users/register	POST	Registers a user in the system.
/rest/users/login	POST	Logs in a user in the system.
/rest/users/getUserFarms	GET	Returns a list of the farms that belong to a user.
/rest/users/addFarm	POST	Adds a new farm to the list of user farms.
/rest/users/removeFarm	GET	Removes a farm from the list of user farms.

Table 6.1: Users REST API

Url	Type	Description
/rest/farms/getFarm	GET	Returns the details of a farm.
/rest/farms/getFarmPlots	GET	Returns the list of plots that belong to a farm.
/rest/farms/addPlot	POST	Adds a new plot to the list of farm plots.
/rest/farms/removePlot	GET	Removes a plot from the list of farm plots.

Table 6.2: Farms REST API

Url	Type	Description
/rest/plots/getPlot	GET	Returns the details of a plot.
/rest/plots/getPlotPlants	GET	Returns the list of plants in a plot.
/rest/plots/addPlant	POST	Adds a new plant to the list of plot plants.
/rest/plots/removePlant	GET	Removes a plant from the list of plot plants.
/rest/plots/getPlotTraps	GET	Returns the list of traps in a plot.
/rest/plots/addTrap	POST	Adds a new trap to the list of plot traps.
/rest/plots/removeTrap	GET	Removes a trap from the list of plot traps.

Table 6.3: Plots REST API

Url	Type	Description
/rest/plants/getPlant	GET	Returns the details of a plant.
/rest/plants/getPlantEnemies	GET	Returns the list of enemies of a plant.
/rest/plants/ getPlantPhenologicalStages	GET	Returns the list of phenological stages of a plant.
/rest/plants/getModelResults	GET	Returns the results of the plant enemy models of a plant in a specific plot.
/rest/plants/addDiseaseObservation	POST	Adds a disease observation of a plant.
/rest/plants/addProductTreatment	POST	Adds a product treatment to a plant.

Table 6.4: Plants REST API

Url	Type	Description
/rest/plants/getTrap	GET	Returns the details of a trap.
/rest/plants/addTrapObservation	POST	Adds a trap observation of a trap in a plot.

Table 6.5: Traps REST API

Url	Type	Description
/rest/plants/getSubstances	GET	Returns all the substances available.
/rest/plants/getProduct	GET	Returns the details of a product.
/rest/plants/getSubstanceProducts	GET	Returns a list of products from a substance.

Table 6.6: Products REST API

6.1.2 JSON response

All the requests to the REST web methods return JSON files so our mobile application can parse them. An example of the result from a request made is: (in this case, the url used was "/rest/plots/getPlot?plotid=19")

```
{
   "plotID":19,
   "farmID":3,
   "sourceID":548,
   "plotName":"Parcela␣Uvas␣e␣Tomate",
   "plotAreaSize":25,
   "plants":[
      {
         "plantID":8,
         "plantName":"Tomate"
      },
      {
         "plantID":7,
         "plantName":"Uva"
      }
   ],
   "plotTraps":[
      {
         "plotTrapID":0,
         "plotID":19,
         "trapID":4,
         "trapLatitude":39.243355,
         "trapLongitude":8.669807
      }
   ]
}
```

6.1.3 Enemy models

As previously mentioned in the database section, our system stores the enemy models in a table, and each model has a code. This code is used to recognise which Java method is supposed to be executed when running our models. This is done using a process called Reflection.

Models are organised in a way that allows easy insertion of a new model into the system. A model is developed in a Java class and it must implement an interface created called EnemyModel. This interface only has one method called run() which returns a list of Object type values. Each model receives its parameters in the constructor and is executed using the run() method. The content of the list that it returns depends on the model specific results. For example it could contain an index value of type Integer or a result value of type Boolean. The reason it returns a list of Object type values is to be able to implement any enemy model and allow it to return its values to be subsequently interpreted.

Then, we have a main class called ExecModels which has one method for each model. Each method runs the respective model class and interprets its results. For example if a model returns an index, the method in ExecModels will see if that index means there is or not risk of infection. It will then store the result on the database. Each method should have its own model code as name (example: GrapeDownyMildewEPI) so we execute it using Reflection.

In order to add a new model to the system it is only required to:

1. Insert model in database, so the system knows for which plot it should run it (each plot has a set of plants, each plant has its enemies and enemies may have several models).

2. Create Java class that executes the model and returns its value.

3. Create a method in the class ExecModels that runs the previously created Java class and interprets its results.

Whenever we want to execute the models in our system, we iterate through all the users, their farms, their plots, their plants and finally their plant enemy models. For each model we want to run, we execute their respective method in the ExecModels class. It will then automatically execute the model, interpret the results and store them in the database in the corresponding plot source.

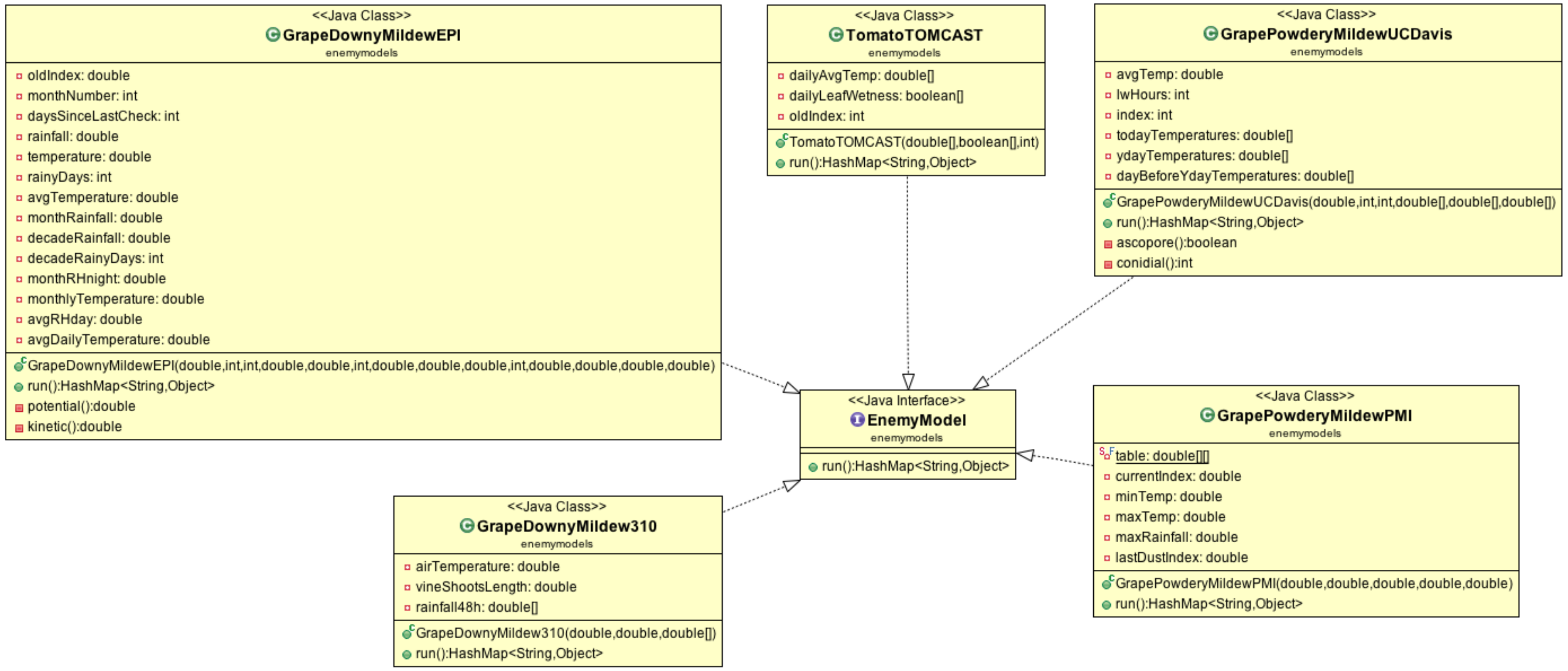


Figure 6.1: Java class diagram of enemy models implemented.

6.1.4 Time-series segmentation

As stated previously we created two scripts that generate image files in SVG format with segmentation of time series with 2 different types of data from our database.

The scripts were developed in Java, and therefore it is possible to execute them from the command line. The notation for their execution is:

```
java -cp "bin:lib/*" ExecSegmentSeries [-interpolate numberNearest maxDistance]
     lat lon day month year numberOfDays maxError
java -cp "bin:lib/*" ExecSegmentSeriesTsp [-interpolate numberNearest
    maxDistance] lat lon day month year numberOfDays maxError
```

ExecSegmentSeriesTsp is the same as ExecSegmentSeries but instead of using air temperature at 1.5 meters, uses air temperature at 0.05 meters.

And their parameters are:

- Optional:
  - -interpolate, if user wants to interpolate data. If not, script will use closest source of information.
  - numberNearest - maximum number of nearest sources to interpolate
  - maxDistance - maximum distance to the nearest sources to interpolate (in meters)
- Required:
  - lat - latitude
  - lon - longitude
  - day - day of the first day to be presented
  - month - month of the first day to be presented
  - year - year of the first day to be presented
  - numberOfDays - number of days following the first day
  - maxError - maximum value of error to be used on the segmentations. Usually is 1

Example of a execution:

```
java -cp "bin:lib/*" ExecSegmentSeries -interpolate 10 50000 35 -8 24 8 2013 5
    1
```

When executed, the script will follow a set of operations in order to generate its outputs.

1. First, it starts by reading the parameters.

2. Then the process of obtaining values start. For both satellite data and weather station data there are 2 different ways of obtaining values depending if the user requested interpolation or not.
    - If user requested interpolation, for each day, for each 15 minute interval, do:
        (a) Obtain all the values in the database for that specific time.
        (b) Calculate the interpolated value for the location provided.
        (c) Add the value into the time series if exists.
    - If user didn't request interpolation
        (a) Obtain nearest source to the location provided
        (b) For each day, for each 15 minute interval, obtain the value from that source and insert into the time series if exists.
3. After, the script starts creating the SVG.
    (a) First, it creates the string which will have the content and inserts the calculated values of sunrise, sunset and solar noon.
    (b) Then, add every known point in our time series to the string.
    (c) After, execute the segmentation algorithm and add the segments into the string. Each segment will be represented by 2 points highlighted, and a line between them.
    (d) Finally the script creates the SVG and prints the string into the file.
4. At the end, the script creates 2 CSV files, one for each time series (each data type) and prints the respective segments, exporting the starting point, end point and the segment slope.

6.2 Code samples

The web server implementation is mainly divided in three sections:

- Scripts for downloading data, inserting it and running disease models.
- Web server with a RESTfull API that receives requests from the mobile application.
- Scripts created to generate image frills with time-series segmentation.

6.2.1 Download satellite data

Satellite data is downloaded from a FTP server hosted by Universidade de Évora. This server receives automatically data from the IPMA and allows us to download them. The files are in Bzip2 format containing a compressed HDF5 file.

We created a script in Java that essentially executes the following steps sequentially:

1. Connect to the FTP server and get the list of files available.
2. For each file:
   (a) Check if data from the file was already inserted previously.
   (b) If it wasn't:
      i. Download it, decompress and obtain the HDF5 file.
      ii. Read data from HDF5 file and insert in our database.

This script can be executed anytime and it will insert the newest data, but usually it is executed daily just before running the enemy models.

6.2.1.1 Connecting to the FTP server

Using the JSch java library we connect to the FTP server and get the list of files:

```java
JSch jsch = new JSch();

// Create session
Session session = jsch.getSession(SFTPUSER, SFTPHOST, SFTPPORT);
session.setPassword(SFTPPASS);
session.connect();

// Open channel
Channel channel = session.openChannel("sftp");
channel.connect();
ChannelSftp channelSftp = (ChannelSftp) channel;

// Get contents of target directory
channelSftp.cd(SFTPWORKINGDIR);
Vector<LsEntry> list = channelSftp.ls("*.bz2");
...
```

6.2.1.2 Downloading .bz2 files and decompressing them

When downloading the .bz2 files from the FTP server we need to decompress and store them in a temporary .h5 file. For the decompression we used the Apache Commons Compress library.

```java
for (LsEntry entry : list) {

  // If file size = 0 ignore file.
  if (entry.getAttrs().getSize() == 0)
    continue;

  // Create temporary HDF5 file to read data from.
  BufferedInputStream in = new BufferedInputStream(channelSftp.get(entry.
      getFilename()));
  FileOutputStream out = new FileOutputStream("temp.h5");
  BZip2CompressorInputStream bzIn = new BZip2CompressorInputStream(in);
  final byte[] buffer = new byte[1024];
  int n = 0;
  while (-1 != (n = bzIn.read(buffer))) {
    out.write(buffer, 0, n);
  }
  ...
}
```

6.2.1.3 Reading HDF5 file

After the HDF5 file is available we open it and read its values.

An HDF5 file is a file containing multiple data sets in the form of spreadsheets. In this case the only dataset that is important is the one called LST which is the first one. To do this we have used the Java HDF5 Object Package:

```
// Retrieve an instance of the temporary H5File
FileFormat fileFormat = FileFormat.getFileFormat(FileFormat.FILE_TYPE_HDF5);

// Open the file with read-only access
FileFormat testFile = fileFormat.open("temp.h5", FileFormat.READ);

// Open the file and retrieve the file structure
testFile.open();
Group root = (Group) ((javax.swing.tree.DefaultMutableTreeNode) testFile.
    getRootNode()).getUserObject();

// Retrieve the dataset "LST"
Dataset lst = (Dataset) root.getMemberList().get(0);
short[] lstRead = (short[]) lst.read();
...
```

After reading the file and obtaining the data set we have to:

1. Cycle through the sheet cells
2. For each cell:
   (a) Check if cell value is valid (-80 means value is not available)
   (b) Get cell coordinates and verify it they belong in Portugal.
   Cell coordinates are obtained using the formulas found in [32].
   (c) If they belong and cell value is valid, add value to be inserted in the end
3. Insert all values previously selected

6.2.2 Download weather station data

Weather station data is obtained from the IPMA web site, so it is only necessary to make an HTTP GET call and receive the CSV files.

The sequence for the download of weather station data and insertion in the database is as follows:

1. For each weather station:
   (a) Download CSV file
   (b) Cycle through its lines and columns
   (c) For each value:
      i. Check if value was already inserted in the database.
      ii. If it wasn't add value to be inserted in the end.
   (d) Insert all values previously selected

To read CSV files we used the OpenCSV parser library for Java.

This is a sample code explaining how it is done:

```
// Get weather stations
SourceType type = SourceType.getSourceTypeByDenomination("Weather␣Station");
Source[] sources = type.getSources();
for (Source source : sources) {

  // Download CSV
  URL website = new URL("http://www.ipma.pt/resources.www/data/observacao/emas/
      hora.csv/ema" + source.sourceExternalID + ".csv");
  BufferedReader in = new BufferedReader(new InputStreamReader(website.
      openStream()));

  // Read CSV
  CSVReader csvReader = new CSVReader(in, ';');
  List<String[]> csv = csvReader.readAll();
  for (String[] line : csv)
    insertValues(line);
}
```

6.2.2.1 Read weather data from files

In addition to inserting data that has been downloaded, it is also possible to insert data that is on disk. However, these data were provided in different formats and therefore it was necessary to create different scripts for each of them.

- LST on text files

  One single text file contains all LST values for all locations in portugal in every 15min interval of a day. The file contains one line for each location (aprox. 6000) and one column for each time interval (96).

- Weather station on Excel files

  One single Excel file contains all meteorological values for one weather station in all available years. The file contains one row for each time interval of 1 hour and one column for each data type.

6.2.3 Enemy models executions

Enemy models are to be executed on a daily basis and for that they require the meteorological data from the previous day.

The overall order of steps required to be executed are:

1. Retrieve all values from the previous day of all necessary data types and all locations. We do this first since this data will be commonly used for all following interpolations.
2. Get all plots in the system.
3. For each plot:
   (a) Interpolate required values for the plot location.
   (b) Check which models will need to be executed in the current plot.
   (c) For each model of each plot:
      i. Execute it and store the results in the database.

6.2.4 Spatial Interpolation

A Java class called IDW was created to make spatial interpolation of values. It contains the following methods:

- IDW(HashMap<Integer, Double>, double, int)

  Constructor that receives an HashMap which is the set of known points, where the key is an Integer with the geodataid, and the Double is its known value. It also receives the maximum distance between points to be used and the maximum number of points.

- setMaxDistance(double)

  Sets the maximum distance to be used.

- setNumNearest(int)

  Sets the maximum number of points to use, ordered by distance to the point to interpolate.

- getSortedDistances(double, double)

  Private method which queries the database for the distance between one point to the whole list of known points, and returns them ordered by the distance ascending.

- interpolate(double, double)

  Returns the value interpolated for the specific location. The parameters are the latitude and longitude.

Whenever we require to interpolate a value for a specific plot we create a new instance of IDW with the known points, and execute the interpolate method.

Example of how the interpolation of daily values is done for one plot:

```java
private static double[] interpolate24hValues(List<HashMap<Integer, Double>>
    positions, GeographicalData geo, Double maxDistance, int numNearest) {
  double[] interpolatedValues = new double[24];

  // For each hour
  for (int i = 0; i < 24; i++) {
    IDW idw = new IDW(positions.get(i), maxDistance, numNearest);
    interpolatedValues[i] = idw.interpolate(geo);
  }
  return interpolatedValues;
}
```

Then, in the IDW class the method interpolate will execute the Inverse Distance Weight algorithm for the location received as a parameter. The following is the code of the method:

```java
  public double interpolate(GeographicalData point) {

    // Get distances ordered ascending
    HashMap<Integer, Double> nearest = getSortedDistances(point.Latitude, point
        .Longitude);

    double sumValues = 0;
    double sumWeight = 0;

    // Iterate trough all known values
    for (int geoID : nearest.keySet()) {

      // Get weight for current known point.
      // Power parameter = 2 as default.
      double weight = 1 / Math.pow(nearest.get(geoid), 2);

      // Add value to sum of values
      sumValues = sumValues + ((positions.get(geoID)) * weight);

      // Add weight to sum of weights
      sumWeight = sumWeight + weight;
    }

    // Return value interpolated
    return sumValues / sumWeight;
  }
```

6.2.5 Web methods implementation

As stated in the State of the Art chapter, the implementation of our web services is done using Apache Tomcat togheter with JAX-RS (Jersey) and a library called Jackson.

The following is an example of how the implementation was made. A simple web method was chosen to demonstrate how the Jersey annotations are specified and how the Jackson library is used to map objects into JSON strings.

```
@Path("/plots/")
public class Plots {

  /**
   * Web Method that receives a plot identifier and returns the plot in JSON
   * format
   *
   * @param plotID
   *            - the ID of the specified plot
   * @return String with the JSON content of the plot
   */
  @GET
  @Path("/getPlot")
  @Produces(MediaType.APPLICATION_JSON)
  public String getPlot(@QueryParam(value = "plotid") int plotID) {

    // Get plot from our database.
    Plot plot = Plot.get(plotID);

    // Instantiate a Jackson mapper.
    ObjectMapper mapper = new ObjectMapper();

    String json = "";
    try {
      // Map the object as a JSON string using Jackson mapper.
      json = mapper.writeValueAsString(plot);
    } catch (JsonProcessingException e) {
      // Do something with the exception. Log error for example.
      ...
    }

    // Return the mapped string or empty if exception occurred.
    return json;
  }

  ...
}
```

6.2.6 Time-series segmentation implementation

The implementation of the time series scripts consisted on taking advantage of a object oriented programming language like Java, creating Java classes to build a hierarchy for our segmentation.

The following are some code samples of the implementation made:

6.2.6.1 Obtain data with interpolation

```java
    // Calendar aux starts from the first time and will be incremented every 15
        min.
    Calendar aux = (Calendar) start.clone();

    // If is to do interpolation
    if (interpolation) {

      // For each day
      for (int i = 0; i < numOfDays; i++) {

        // Create daily values for later calculations
        TimeSeries<TimeDate> daily = new TimeSeries<TimeDate>(24);
        Calendar day = (Calendar) aux.clone();

        // For each 15 minute interval
        for (int j = 0; j < 24 * 4; j++) {

          // Get all values in this specific time
          List<TemporalData> datas = TemporalData
              .getTemporalDataExactTime(aux, type.dataTypeID);

          // Insert values into a HashMap to use for interpolation
          HashMap<Integer, Double> positions = new HashMap<Integer, Double>();
          for (TemporalData data : datas) {
            positions.put(data.geoDataID,
                Double.parseDouble(data.value));
          }

          // Create IDW instance
          IDW idw = new IDW(positions, maxDistance, numNearest);

          // Obtain interpolated value for position provided
          double value = idw.interpolate(lat, lon);

          // If value is valid, insert into TimeSeries
          if (value != 0 && !Double.isNaN(value)) {
            // Create TimeDate value to be inserted in TimeSeries
            TimeDate time = new TimeDate(origin, new Date(aux.getTimeInMillis()
                ));

          ts.add(time, value);
          daily.add(time, value);
        }

        // Increase calendar by 15min for next iteration
        aux.add(Calendar.MINUTE, 15);

      }
      // At the end of each day, addStatsToText will calculate daily max, min
           and avg values and add them to SVG.
      addStatsToText(i, daily, true, day);
    }
  }
```

6.2.6.2 Obtain data without interpolation

```
  // Calendar aux starts from the first time and will be incremented every
  // 15min.
  Calendar aux = (Calendar) start.clone();

  // If is to do interpolation
  if (interpolation) {
    ...
  } else {

    // Get nearest source. In this case is for satellite.
    GeographicalData geo = GeographicalData
        .getGeoDataByLatLon(lat, lon);
    if (geo == null) {
      geo = getNearbyLST(lat, lon);
    } else {
      Source src = Source.getSource(geo.SourceID);
      if (src.sourceTypeID != Source.getSourceByName("LSA␣SAF").sourceTypeID)
        geo = getNearbyLST(lat, lon);
    }

    // newStart will be used as the first time of each interval. Starts
    // as the previous day since it will be immediatly incremented.
    Calendar newStart = (Calendar) start.clone();
    newStart.add(Calendar.DAY_OF_YEAR, -1);

    // For each day
    for (int i = 0; i < numOfDays; i++) {

      // Create daily time series for later calculations
      TimeSeries<TimeDate> daily = new TimeSeries<TimeDate>(24);

      // Add a day to both calendars to adjust interval
      newStart.add(Calendar.DAY_OF_YEAR, 1);
    aux.add(Calendar.DAY_OF_YEAR, 1);

    // Obtain all values from a day from one source
    List<TemporalData> allData = TemporalData
      .getTemporalDataInInterval(newStart, aux,
        type.dataTypeID, geo.GeoDataID);

    // Iterate through all values
    Iterator<TemporalData> ite = allData.iterator();
    while (ite.hasNext()) {
     TemporalData data = ite.next();

     // Create TimeDate value to be inserted in TimeSeries
     Date d = new Date(data.time.getTime());
     TimeDate time = new TimeDate(origin, d);

     // Insert value into TimeSeries
     double value = Double.parseDouble(data.value);
     ts.add(time, value);
     daily.add(time, value);
    }
    // At the end of each day, addStatsToText will calculate daily
    // max, min and avg values and add them to SVG.
    addStatsToText(i, daily, true, newStart);
   }
  }
```

6.2.6.3 SVG creation

The SVG is created initially by filling a string with XML. Then when the string is finished, we create the SVG file and add the string content into it.

Creating the string and file header, graphHeader, footer and the graph itself with the points:

```
String header = "<svg␣xmlns='http://www.w3.org/2000/svg'␣width='"
      + (numOfDays * 240) + "'␣height='860'␣>";
String graphHeader = "<g␣transform='translate(0,700)␣scale(10,-8)'>";
String footer = "</g></svg>";
String graph = ts.graph("black") + tsStation.graph("blue");
```

The method graph from the TimeSeries class, simply returns a string with all the points in SVG notation:

```
public String graph(String color) {

  String graph = "";
  for (int i = 0; i < time.size(); i++) {
    String coordX = Double.toString(this.time.get(i).valueOf());
    String coordY = Double.toString(this.data.get(i));
    graph += "<circle␣r='0.2'␣fill='" + color + "'␣cx='" + coordX
        + "'␣cy='" + coordY + "'/>";
  }

  return graph;
}
```

For the segments, we must first execute the segmentation algorithms, and then insert each statement into the SVG string:

```
for (Segment<TimeDate> seg : slidingWindow(ts, error)) {

      double x1 = seg.getTimeSeries().getTimeValue(0);
      double x2 = seg.getTimeSeries().getTimeValue(
          seg.getTimeSeries().getLength() - 1);
      graph += seg.getLine().graph(x1, x2, "red");
    }
```

Where, the graph method from the Line class has a similar return string to the previous graph method:

```
public String graph(double x1, double x2, String color) {
  double y1 = getY(x1);
  double y2 = getY(x2);

  String line = "<line␣stroke='" + color + "'␣stroke-width='0.1'␣x1='"
      + x1 + "'␣y1='" + y1 + "'␣x2='" + x2 + "'␣y2='" + y2 + "'␣/>";
  String p1 = "<circle␣r='0.2'␣stroke='" + color + "'␣cx='" + x1
      + "'␣cy='" + y1 + "'/>";
  String p2 = "<circle␣r='0.2'␣stroke='" + color + "'␣cx='" + x2
      + "'␣cy='" + y2 + "'/>";

  return p1 + line + p2;
}
```

Finally, when the string is finished, all that is required is to create the file and write the string into it:

```
      OutputStream os = new FileOutputStream("SegmentSeries_" + lat + "_" + lon
           + "_"
          + day + "_" + month + "_" + year + "_" + numOfDays + "_"
          + interpolation + ".svg");
      PrintStream printStream = new PrintStream(os);
      printStream.print(header + text + graphHeader + graph + axis
          + footer);
      printStream.close();
```

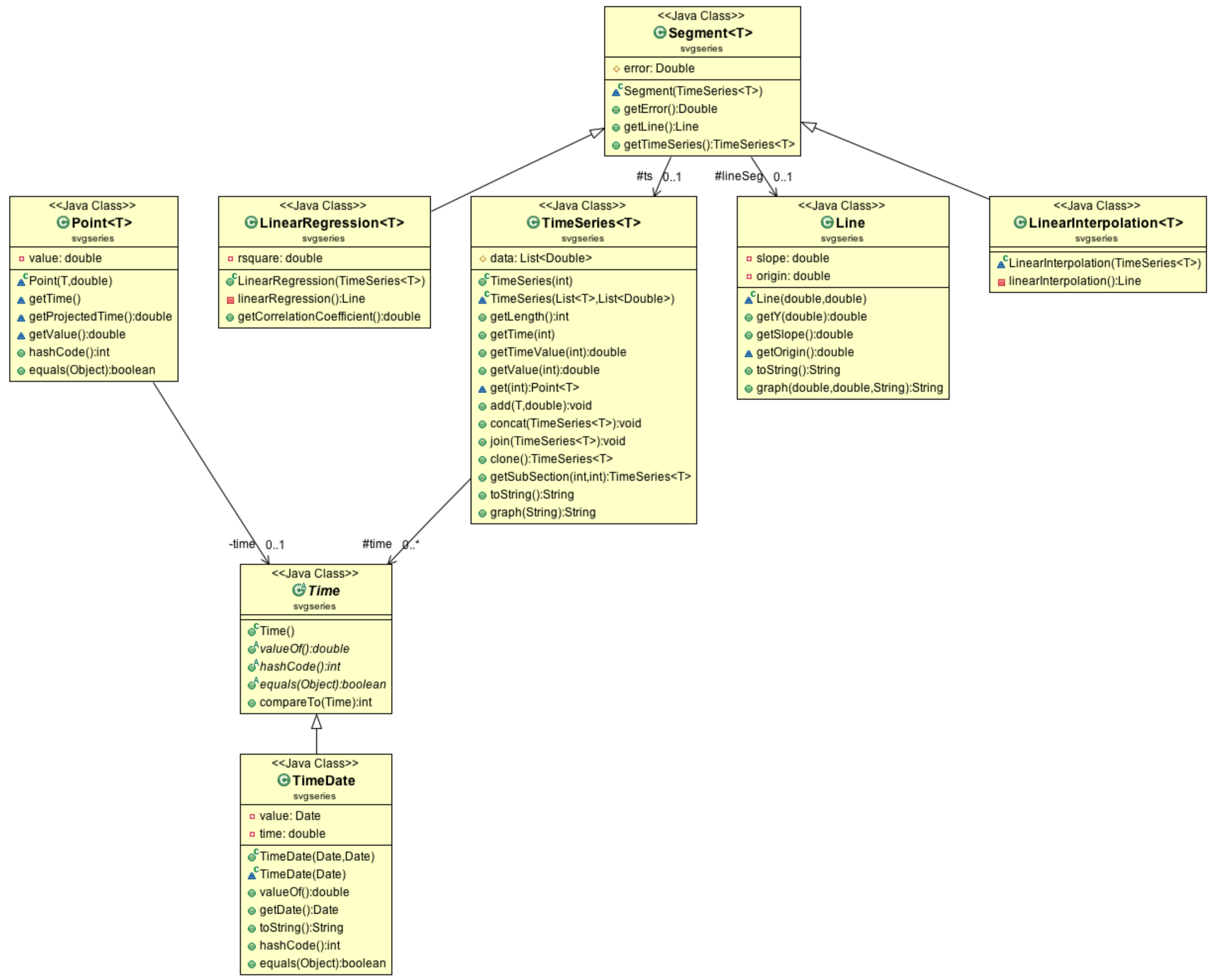


Figure 6.2: Class diagram of the classes created for time-series segmentation.

6.3 Conclusions

Regarding the REST API, we created an API that allows insertion and selection of any required information to the mobile application. It was also established an architecture which allows easy insertion of new enemy models in our system without having to make drastic changes to the code every time a new model is created.

7 Mobile Application

This chapter focuses on explaining how the mobile application was developed. It is not too much detailed since only a simple prototype was developed to test the interaction with our server.

First we start with some code samples about the interaction between the mobile application and the web server, then we explain the usability proposed in our application, followed by a sequence of screenshots which show how the usability is displayed.

7.1 Implementation

The Android application was developed for Android 2.2 using the Android SDK for the Eclipse IDE.

Since it was a prototype, the implementation was fairly simple. However we will present a brief explanation of the interaction with our web services.

7.1.1 JSON parser

We obtain and parse the data via an auxiliary class which was created to make HttpRequests and return the results in Java objects.

The class is called JSONParser and has 2 methods:

- getJSONObjectFromUrl(String url)

  Parses the result as a JSONObject.

- getJSONArrayFromUrl(String url)

  Parses the result as a JSONArray.

Sample code from JSONParser class:

```

public class JSONParser {

  static InputStream is = null;
  static String json = "";

  public JSONObject getJSONObjectFromUrl(String url) {

    // Making HTTP request
    try {
      // defaultHttpClient
      DefaultHttpClient httpClient = new DefaultHttpClient();
      HttpGet httpGet = new HttpGet(url);
      HttpResponse httpResponse = httpClient.execute(httpGet);
      HttpEntity httpEntity = httpResponse.getEntity();
      is = httpEntity.getContent();
    } catch (Exception e) {
      Log.e("Buffer␣Error", "Error␣converting␣result␣" + e.toString());
    }

    // Read from stream
    try {
      BufferedReader reader = new BufferedReader(new InputStreamReader(
          is, "utf-8"), 8);
      StringBuilder sb = new StringBuilder();
      String line = null;
      while ((line = reader.readLine()) != null) {
        sb.append(line + "n");
      }
      is.close();
      json = sb.toString();
    } catch (Exception e) {
      Log.e("Buffer␣Error", "Error␣converting␣result␣" + e.toString());
    }

    JSONObject jObj = null;
    // try parse the string to a JSON object
    try {
      jObj = new JSONObject(json);
    } catch (JSONException e) {
      Log.e("JSON␣Parser", "Error␣parsing␣data␣" + e.toString());
    }

    // return JSON String
    return jObj;

  }

  // Make request and parse to JSONArray
  public JSONArray getJSONArrayFromUrl(String url) {

    // Making HTTP request
    try {
      // defaultHttpClient
      DefaultHttpClient httpClient = new DefaultHttpClient();
      HttpGet httpGet = new HttpGet(url);
      HttpResponse httpResponse = httpClient.execute(httpGet);
      HttpEntity httpEntity = httpResponse.getEntity();
      is = httpEntity.getContent();
    } catch (Exception e) {
      Log.e("Buffer␣Error", "Error␣converting␣result␣1" + e.toString());
    }

    // Read from stream
    try {
      BufferedReader reader = new BufferedReader(new InputStreamReader(
          is, "utf-8"), 8);
      StringBuilder sb = new StringBuilder();
      String line = null;
      while ((line = reader.readLine()) != null) {
        sb.append(line + "n");
      }
      is.close();
      json = sb.toString();
    } catch (Exception e) {
      Log.e("Buffer␣Error", "Error␣converting␣result␣" + e.toString());
    }

    JSONArray jObj = null;
    // try parse the string to a JSON array
    try {
      jObj = new JSONArray(json);
    } catch (JSONException e) {
      Log.e("JSON␣Parser", "Error␣parsing␣data␣" + e.toString());
    }

    // return JSON String
    return jObj;
  }
}
```

7.1.2 Obtaining data

And then, this is how our Java code uses the JSON Parser to make requests and store its results into arrays.

```
    // Instantiate parser
    JSONParser parser = new JSONParser();

    // Obtain the values from HTTP Request
    JSONArray array = parser.getJSONArrayFromUrl("http://"
        + getString(R.string.host)
        + "/DiseaseWarningSystem/rest/users/getUserFarms?userid="
        + userid);

    // Read values from JSONArray and store in temporary array
    String[] farms = new String[array.length()];

    // The IDs are important to create the lists in the application
    farmIDs = new int[array.length()];
    try {
      // Looping through All Caches
      for (int i = 0; i < array.length(); i++) {
        JSONObject c = array.getJSONObject(i);
        farms[i] = c.getString("farmName");
        farmIDs[i] = c.getInt("farmID");
      }
    } catch (JSONException e) {
      // Do something with exception. Log error for example.
    }
```

7.2 Usability

As far as our mobile application, we created a sequence of screens where we can move along the information regarding one user.

At this moment the application is made in Portuguese since our targets are Portuguese farmers, but a possible future work is to make an international version using Android's capabilities.

7.2.1 Login screen

The first screen is the login screen which only allows the user to access its information if he is previously registered. When the login is made the main menu enables the used to choose between seeing it's farms or updating his account.

7.2.2 Farms and plots

After choosing to view the list of farms, it is possible to select one to:

- View a list of farm plots.
- Add a new plot.
- Remove the association of the farm to the user.

The same can then later be done to the plot. After selecting the plot from a list of plots that belong to a farm, it is possible to:

- List the plot's crops.
- Add a new crop.
- View a list of its traps.
- Add a new trap.
- Remove a plot from the current farm.

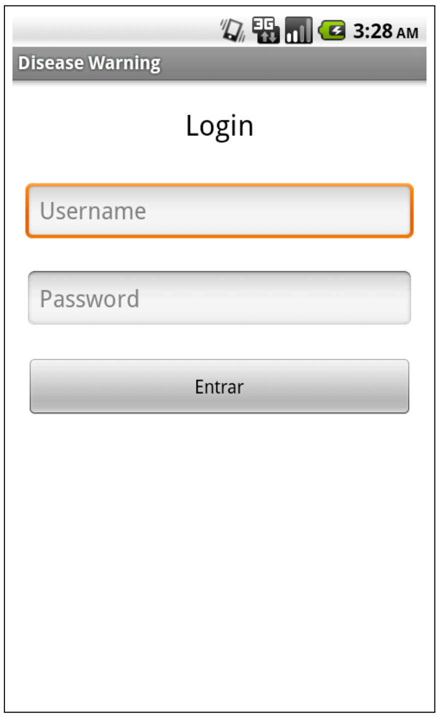
7.2.3 Crops and enemies

Finally, after choosing the crop from a list of crops it is possible to:

- Show the results of plant enemy models for that particular plot location.
- Add a disease observation.
- Add a plant phenological stage observation.
- Add a treatment made.
- Delete crop from the current plot.

7.3 Screenshots

The following are screenshots taken to the Android application to illustrate the possible options that the users should have in our application.



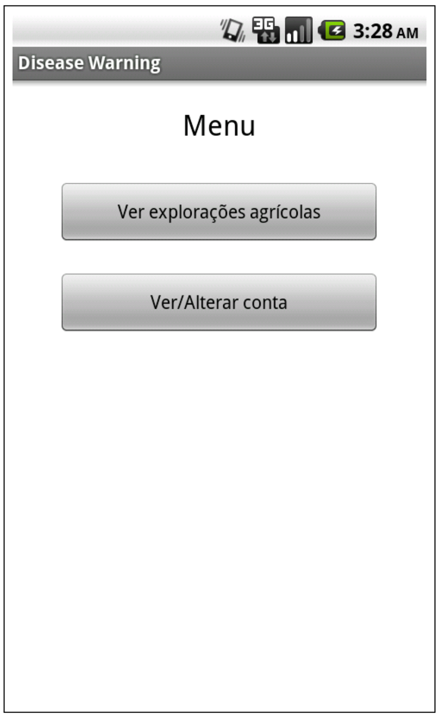
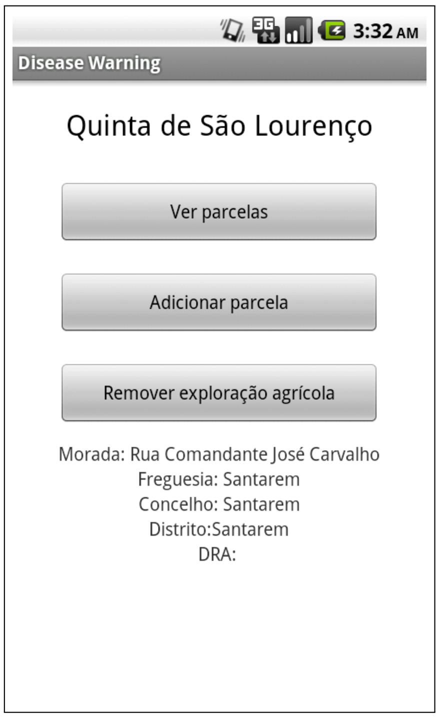
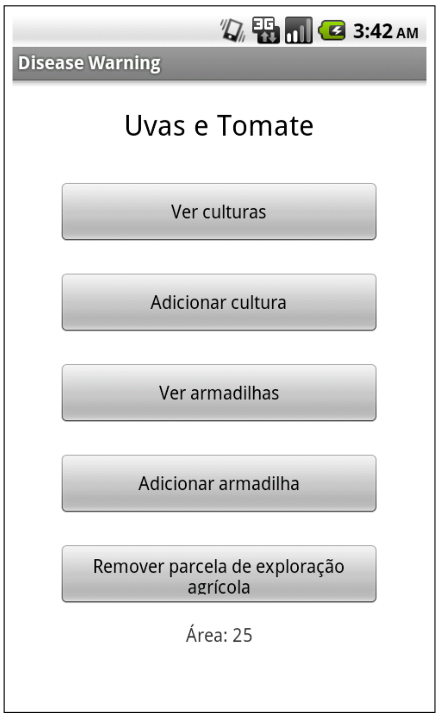


Figure 7.1: Login screen.

Figure 7.2: Main menu.



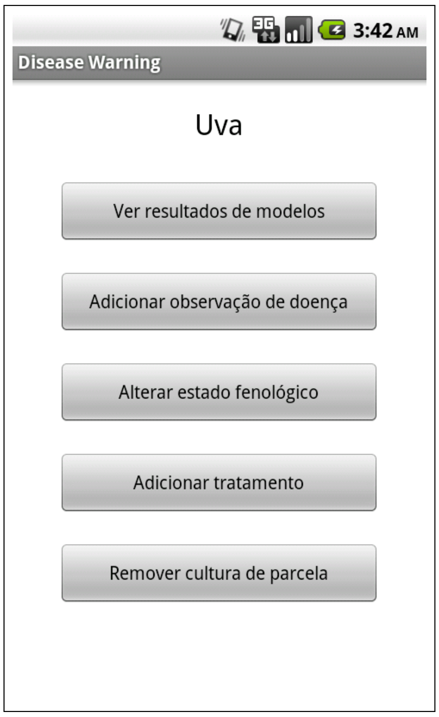
Morada: Rua Comandante José Carvalho
Freguesia: Santarem
Concelho: Santarem
Distrito:Santarem
DRA:

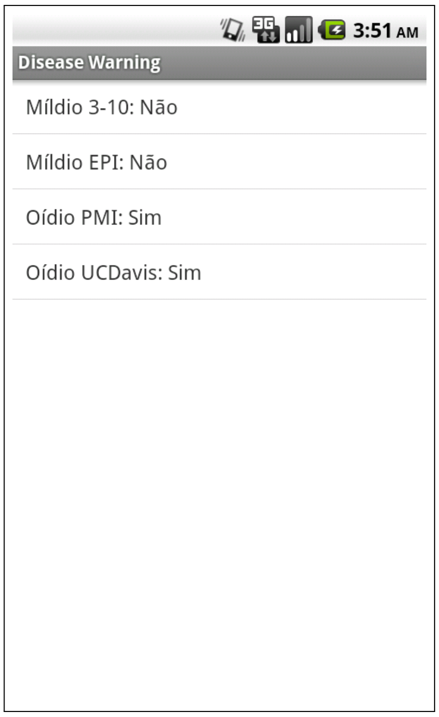


Área: 25

Figure 7.3: Example of a view of a farm.

Figure 7.4: Example of a view of a plot.


3:42 AM
Disease Warning
Uva
Ver resultados de modelos
Adicionar observação de doença
Alterar estado fenológico
Adicionar tratamento
Remover cultura de parcela



Míldio 3-10: Não

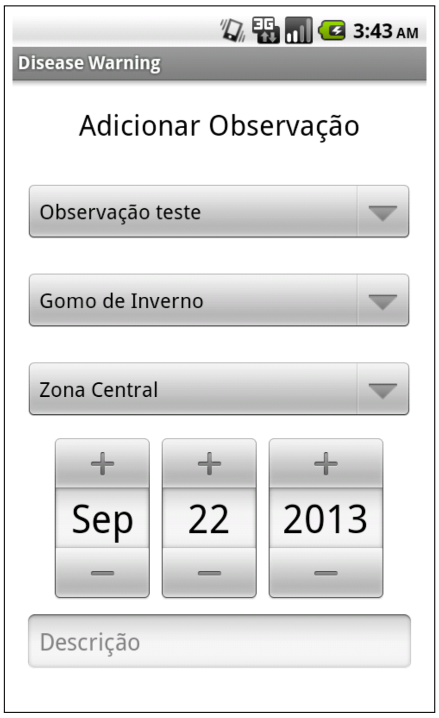
Míldio EPI: Não

Oídio PMI: Sim

Oídio UCDavis: Sim

Figure 7.5: Example of a view of a crop.

Figure 7.6: Example of a view of plant enemy models results.



Observação teste

Gomo de Inverno

Zona Central

Sep 22 2013

Descrição

Figure 7.7: Example of a view for submitting a plant enemy observation.

8 Performance tests

In this chapter we provide the results of this work obtained while testing our system. Since models validation wasn't the objective, these tests are mostly performance tests to verify if the system is able to process a large number of plots and how long it does it take.

8.1 System characteristics

All tests were made on a MacBook Pro. The system characteristics are:

- Processor: 2.3 GHz Intel Core i5
- Memory: 4 GB 1333 Mhz DDR3
- Graphics: Intel HD Graphics 3000 384 MB
- Operating System: OS X 10.8.4
- Disk: 320 SATA disk, with approximately 30 Gb free

Since tests were made on a laptop with limited computing performance, it is possible to conclude that when the platform is tested on a server with much higher memory and processor the results are expected to be much better.

8.2 Data insertion

8.2.1 Satellite

8.2.1.1 From FTP Server

Each file from the FTP server has a maximum of 6000 values which correspond to the 6000 locations of Portugal.

While executing our script to download files and insert them, the highest number of values in one file we encountered was 6099 which took around 5 seconds to insert.

If we multiply that value for 24 we get exactly 2 minutes, which is the approximate time it takes to insert all values corresponding to a day of satellite data.

8.2.1.2 From text file

Each text file has around 6000 lines corresponding to the locations in Portugal and 96 columns corresponding to the 15 minute intervals in one day.

Therefore, the maximum number of values to be inserted are: $6000 * 96 = 576000$. However since some values might be not available (due to cloud coverage for example) these values can be different. We made five tests for five different days.

- 10 January 2008
  - Number of values: 206377
  - Duration: 43 seconds
- 11 January 2008
  - Number of values: 238859
  - Duration: 48 seconds
- 12 January 2008
  - Number of values: 407504
  - Duration: 1 minute and 10 seconds
- 13 January 2008
  - Number of values: 147216
  - Duration: 30 seconds
- 14 January 2008
  - Number of values: 315165
  - Duration: 49 seconds

8.2.2 Weather stations

8.2.2.1 From IPMA website

There are around 125 weather stations with data available on the IPMA website. Some stations have different data types available but almost all of them have 6 values for each hour.

So, since the script should be also executed once every day, we will have a maximum of 18000 values to insert.

These 18000 values took 18 seconds to be downloaded and inserted in the database.

8.2.2.2 From Excel file

Each Excel file has all data from 2008 to 2010 from one weather station. We executed our script for several different files and recorded how long it took.

- Sines
  - Number of values: 1133644
  - Duration: 6 minutes and 29 seconds
- Porto
  - Number of values: 1076577
  - Duration: 5 minutes and 29 seconds
- Coimbra
  - Number of values: 1130877
  - Duration: 4 minutes and 40 seconds

With a 5-6 minute average for each station, and since we have historic files of 59 stations, it would take around 5 hours to insert all data from 59 weather stations from 2008 to 2010.

8.3 Data selection

Due to the indexes created on our `temporaldata` table, all selections are very fast to be executed.

The most common query to be made to our database is to select all temporal values from all locations of one specific data type and in one day.

A test was made to select all land surface temperature values in January 12, 2008. It took 8 seconds to obtain 407504 values.

8.4 Models execution

In these tests we execute our main script that will run several enemy models and store their results in the database. This script should be executed one time each day, right after downloading and inserting the latest meteorological data.

An important factor is that this script makes all data selections, interpolations, executions of models and insertions of their results.

Several tests were made by varying the number of models.

8.4.1 1000 models

In this test we made a simple test. Since we have implemented 5 models, we created 200 test plots and executed their models for one day.

Duration: 35 seconds

8.4.2 10000 models

Then, we tried the same 5 models but with 2000 plots for one specific day.

Duration: 6 minutes and 7 seconds

8.4.3 100000 models

Finally, we created 20000 fictional plots and executed 5 models for each.

Duration: 1 hour, 2 minutes and 35 seconds

8.5 Time-series segmentation

Some tests were also made to verify how long would take for time-series segmentation to execute depending on the number of days requested and if the user chose to have interpolation or not.

8.5.1 Interpolation

We tried with interpolation for several different numbers of days:

- 1 day: 32 seconds
- 5 days: 3 minutes and 5 seconds
- 15 days: 8 minutes and 36 seconds
- 30 days: 14 minutes and 48 seconds

8.5.2 No interpolation

We also made some tests to verify how long the same executions would take without interpolation:

- 1 day:13 seconds
- 5 days: 32 seconds
- 15 days: 1 minute and 71 seconds
- 30 days: 3 minutes and 7 seconds

8.6 Conclusions

Before making conclusions, it is necessary to take into consideration that all tests were performed on a laptop with limited processing and memory, and with other applications running at the same time.

In data insertions, it is possible to verify that with our implementation of the COPY statement the insertion times are extremely fast and efficient. Each day it would take a couple of minutes to insert all values from the current sources available.

In data selections, as was previously appointed, the indexes created allow fast data access to any values we might want to read.

In the models executions, although some executions took around one hour, they are supposed to be executed only once each day, and even if this laptop was going to be the server executing them, they would still be all executed under a couple of hours. Also, the last test executed a large amount of models corresponding to 20000 plots. This is probably around the maximum number of plots that a system like this could have if it is restricted to Portugal.

In conclusion, with a good server hosting our platform, we conclude that there won't be any major efficiency problems with our scripts executions and our platform is ready to be deployed and start running.

9 Final conclusions

First it was necessary to understand the agricultural domain and know what was the problem in order to list the necessary requirements for our implementation. Based on the problem, a solution was planned. It involved a database, a web server and a mobile application.

The database was created in PostgreSQL with PostGIS extension and stores all the meteorological data available from our satellite and weather station sources. It also stores information regarding the system users, their farms and their plots, as well as interactions made and the enemy models outcomes.

A server was created using Apache Tomcat which enables it to be constantly running. The server has many functions. The first is to download and insert meteorological data in the database. To do this, we created scripts that are able to insert satellite data from a FTP server or from text files, and weather station data from the IPMA website or from Excel files.

Secondly, we created a script that iterates trough all plots that belong to each user, and, for each plot, executes each associated enemy model storing its results in the database. The structure developed for the enemy models allows easy insertion of new models to the system.

It was also created a RESTfull API to interact with the mobile application. This API has methods that allow the mobile application to display all information regarding the user and to submit his interactions.

Some specific scripts were also constructed, as requested by the PROTOMATE project to allow future analysis of data in our system. One of them is the segmentation of time series, which generates a linear interpolated representation of two different data types for future comparisons (usually LST and Tsp).

We also developed a mobile application to interact with our system. This mobile application is just a prototype but allows most of the required interaction between the user and the system, such as showing the model results for the user's plots and allowing the user to submit different types of observations.

After the development, performance tests were made to verify if our platform is efficient and scalable.

In summary, we have created and tested a platform that is a starting point for a complex system that can handle billions of agricultural data at the same time, executes empirical models every day and serves as an analysis application for future researches.

9.1 Future work

Since one of our main goals for our platform is to be scalable and flexible, there are obviously several subjects that can/might be done in the future.

9.1.1 Validation of models with real data

It can be possible to validate the models implemented. For that it is necessary to execute these models on a daily basis for a long period of time, and then, compare its results with real time observations made by the agronomists.

9.1.2 Development of new enemy models

Apart from validating models it can be also possible to create new models for other diseases. All that is necessary is to verify at which meteorological circumstances infections occurred and use the weather values stored in the database to create new models.

9.1.3 Creation of risk maps from the platform and connection to a GIS

At this moment the infection results are only being stored in the database and presented to the users via the mobile application. However since we used a geospatial database as PostgreSQL with PostGis, it is possible to integrate our database with an geographical information system, such as QuantumGIS.

These systems are able to read geographical data from the database and generate risk maps with its values to be visualised.

9.1.4 Robust interpolation methods

It can be possible to implement more robust interpolation methods in order to better detect correlation between, for example, the relative humidity index and relative humidity verified at weather stations.

9.1.5 Improving mobile interface

Since the mobile application created was just a prototype to receive and send information to the web server, it is necessary to improve its interface in case a more serious application is required. This improved mobile application should also obey the Android guidelines.

9.1.6 Add more data sources and enemy models

In the future, new data sources such as different satellites might be available and our platform is able to store any kind of data.

Besides, it is also possible to integrate new enemy models into our system, allowing the possibility of having dozens or hundreds of models in the future.

9.2 Work done

Since one of the goals of this dissertation was to create a platform to be used by the PROTOMATE project, a stable version was deployed early and started being used.

9.2.1 Developments made

Routines for analysing and exploring the data have been implemented in the PostgresSQL server by developing specific pg/PLSQL stored procedures and functions. Moreover, it is now possible to visualise the data via a QuantumGIS module developed in Phyton, which remotely connects to the database.

The schema of the database did not suffer any substantial changes, except special tables to store the results of long running SQL queries to process the data.

9.2.2 Results obtained

At this stage, the team of the PROTOMATE project is analysing the LST data since 2008 in order to check their quality.

It was possible to detect some problems in the existing data, namely impossible temperature values. This was a problem that was corrected in late July by the LSA SAF. Other more subtle problems in the satellite LST data are being currently studied.

Additionally, the correlation of LST with weather station parameters is being studied, and it was possible to identify very interesting patterns on the data that will be published by the PROTOMATE team.

Bibliography

[1] *Plant disease management*. [Online]. Available: http://www.apsnet.org/edcenter/intropp/topics/Pages/PlantDiseaseManagement.aspx (visited on 02/01/2013).

[2] *Agronomy day - the morrow plots: a landmark for agriculture*. [Online]. Available: http://agronomyday.cropsci.illinois.edu/2001/morrow-plots/ (visited on 09/22/2013).

[3] *Cadernos de campo*. [Online]. Available: http://www.dgadr.mamaot.pt/sustentavel/producao-integrada/cadernos-de-campo (visited on 09/22/2013).

[4] Wikipedia, *Phenology — wikipedia, the free encyclopedia*, [Online; accessed 3-December-2012], 2012. [Online]. Available: \url{http://en.wikipedia.org/wiki/Phenology}.

[5] *Vineyard ipm scouting report for week of 3 may 2010*. [Online]. Available: http://door.uwex.edu/files/2010/10/IPMReportweek5.3.10.pdf.

[6] *Tomato*. [Online]. Available: http://www.sqm.com/en-us/unidadesdenegocios/nutrici%C3%B3nvegetaldeespecilidad/cultivos/tomate.aspx (visited on 09/22/2013).

[7] P Amaro, *A protecção integrada*. Portuguese. Lisboa, 2003, ISBN: 9728669100 9789728669102.

[8] *Bayer CropScience portugal*. [Online]. Available: http://www.bayercropscience.pt/internet/problemas/problema.asp?id_problema=154 (visited on 02/01/2013).

[9] *Models: powdery mildew of grape–UC IPM*. [Online]. Available: http://www.ipm.ucdavis.edu/DISEASE/DATABASE/grapepowderymildew.html (visited on 02/01/2013).

[10] C. Thomas, W. Gubler, and G. Leavitt, "Field testing of a powdery mildew disease forecast model on grapes in california", *Phytopathology*, vol. 84, p. 1070, 1994.

[11] E. Weber, W. Gubler, and A. Derr, "Powdery mildew controlled with fewer fungicide applications", *Winegrowing. Jan./Feb*, pp. 13–16, 1996.

[12] *Control of powdery mildew using the UC davis powdery mildew risk index.* [Online]. Available: `http://www.apsnet.org/publications/apsnetfeatures/Pages/UCDavisRisk.aspx` (visited on 09/22/2013).

[13] R. Snyder, P. La Vine, M. Sall, J. Wrysinski, and F. Schick, "Grape mildew control in the central valley of california using the powdery mildew index.", *Leaflet-University of California*, 1983.

[14] *Downy mildew of grape.* [Online]. Available: `http://www.apsnet.org/edcenter/intropp/lessons/fungi/Oomycetes/Pages/DownyMildewGrape.aspx` (visited on 09/22/2013).

[15] T. Caffi, V. Rossi, A. Cossu, and F. Fronteddu, "Empirical vs. mechanistic models for primary infections of plasmopara viticola", *EPPO Bulletin*, vol. 37, no. 2, pp. 261–271, Aug. 2007, ISSN: 0250-8052, 1365-2338. DOI: `10.1111/j.1365-2338.2007.01120.x`. [Online]. Available: `http://doi.wiley.com/10.1111/j.1365-2338.2007.01120.x` (visited on 09/22/2013).

[16] *The university of maine - UMaine extension: insect pests & plant diseases - early blight of tomato.* [Online]. Available: `http://umaine.edu/ipm/ipddl/publications/5087e/` (visited on 09/22/2013).

[17] *The university of maine - UMaine extension: insect pests & plant diseases - septoria leaf spot of tomato.* [Online]. Available: `http://umaine.edu/ipm/ipddl/publications/5088e/` (visited on 09/22/2013).

[18] [Online]. Available: `http://www.extension.umn.edu/yardandgarden/ygbriefs/p250tomatoanthracnose.html` (visited on 09/22/2013).

[19] *NEWA - using tomcast and blitecast forecasts effectively.* [Online]. Available: `http://newa.cornell.edu/index.php?page=Using-Tomcast-and-Blitecast-Forecasts-Effectively#simcast` (visited on 09/22/2013).

[20] R. Cowling, P. Rundel, B. Lamont, M. Kalin Arroyo, and M. Arianoutsou, "Plant diversity in mediterranean-climate regions", *Trends in Ecology & Evolution*, vol. 11, no. 9, pp. 362–366, 1996.

[21] J. Robinson and J. Harding, *The Oxford companion to wine.* Oxford University Press Oxford, 1994, vol. 56.

[22] M. do Céu Godinho, F. Amaro, E. Figueiredo, and A. Mexia, "Protecção integrada em tomate de indústria", in, I. N. de Investigação Agrária e das Pescas, Ed. 2006, ch. 6.1 - Mildio, pp. 22–31, ISBN: 972-579-032-4.

[23] M. Gleason, K. Duttweiler, J. Batzer, S. Taylor, P. Sentelhas, J. Monteiro, and T. Gillespie, "Obtaining weather data for input to crop disease-warning systems: leaf wetness duration as a case study", *Scientia Agricola*, vol. 65, no. SPE, pp. 76–87, 2008.

[24] *Why do dew drops do what they do on leaves?* [Online]. Available: `http://www.terradaily.com/reports/Why_do_dew_drops_do_what_they_do_on_leaves_999.html` (visited on 09/22/2013).

[25] K. Kim, S. Taylor, M. Gleason, and K. Koehler, "Model to enhance site-specific estimation of leaf wetness duration", *Plant Disease*, vol. 86, no. 2, pp. 179–185, 2002.

[26] P. Sentelhas, A. Dalla Marta, S. Orlandini, E. Santos, T. Gillespie, and M. Gleason, "Suitability of relative humidity as an estimator of leaf wetness duration", *Agricultural and Forest Meteorology*, vol. 148, no. 3, pp. 392–400, 2008.

[27] J. Yang, Y. Wang, and P. August, "Estimation of land surface temperature using spatial interpolation and satellite-derived surface emissivity", *Journal of Environmental Informatics*, vol. 4, no. 1, pp. 37–44, 2004.

[28] K. Stahl, R. Moore, J. Floyer, M. Asplin, and I. McKendry, "Comparison of approaches for spatial interpolation of daily air temperature in a large region with complex topography and highly variable station density", *Agricultural and Forest Meteorology*, vol. 139, no. 3, pp. 224–236, 2006.

[29] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series", in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, 2001, pp. 289–296.

[30] *General solar position calculations*. [Online]. Available: `http://www.esrl.noaa.gov/gmd/grad/solcalc/solareqns.PDF`.

[31] S. Kisilevich, F. Mansmann, M. Nanni, and S. Rinzivillo, *Spatio-temporal clustering*. Springer, 2010. [Online]. Available: `http://link.springer.com/chapter/10.1007/978-0-387-09823-4_44` (visited on 09/22/2013).

[32] *Product user manual - land surface temperature (lst)*. [Online]. Available: `http://landsaf.meteo.pt/GetDocument.do?id=304`.