ESCUELA SUPERIOR DE INFORMÁTICA
UNIVERSIDAD DE CASTILLA-LA MANCHA

SOFTWARE ENGINEERING II

Animalia
Curso 2015/16

AUTHORS:
Miriam Castillo Torroba
Jaime Cepeda Villamayor
Jose Antonio Martinez López

Contents

1 Introduction

1.1 Previous Considerations

Our team is conformed by four people distributed in this way:

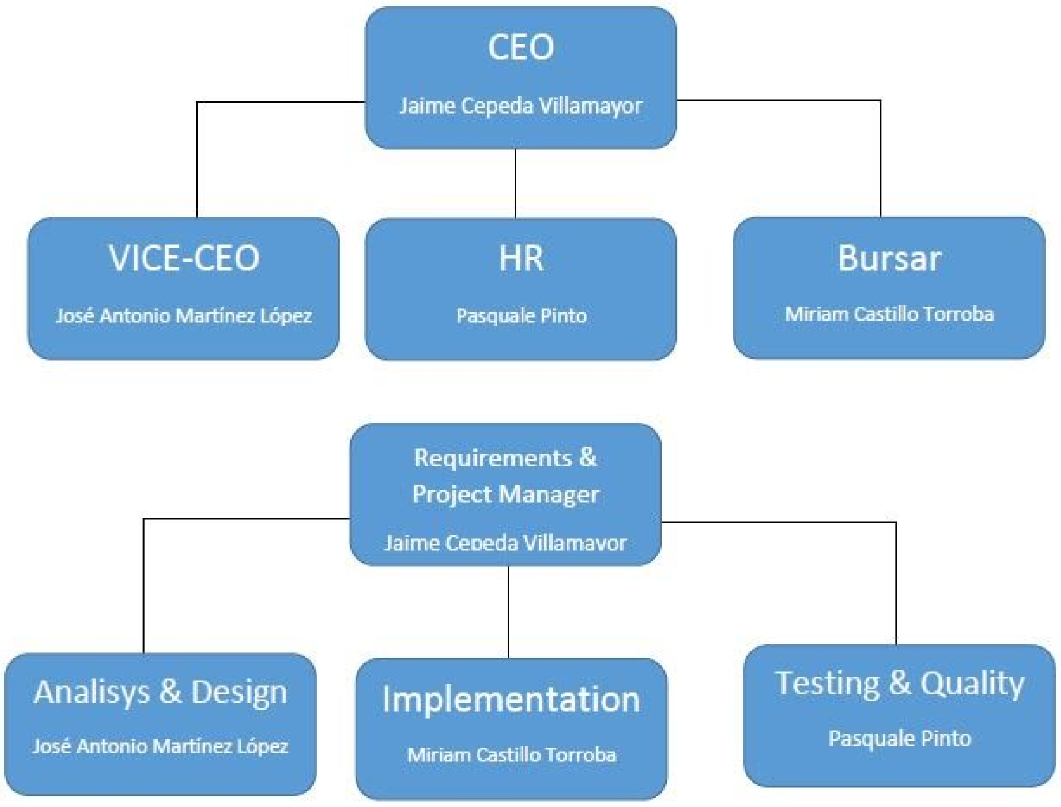


Figure 1: Company Organization

The cost per hour is:

- Analyst: 90€/hour
- Designer: 60€/hour
- Developer: 30€/hour
- Tester: 20€/hour

1.2 Client Description

The client ask for a monolithic application for the management of the animals of a zoo. The system has one type of user: the administrator, which is the one that manage the animals that are in the zoo and their main information like breed, name, ID, weight, height, sex and others.Also, he is the one that manage the nutritional information (food, drink, medication...) of each animal.

The way to access to the software is via desktop app, and each person has to login to the system with a user and password in order to identify him.

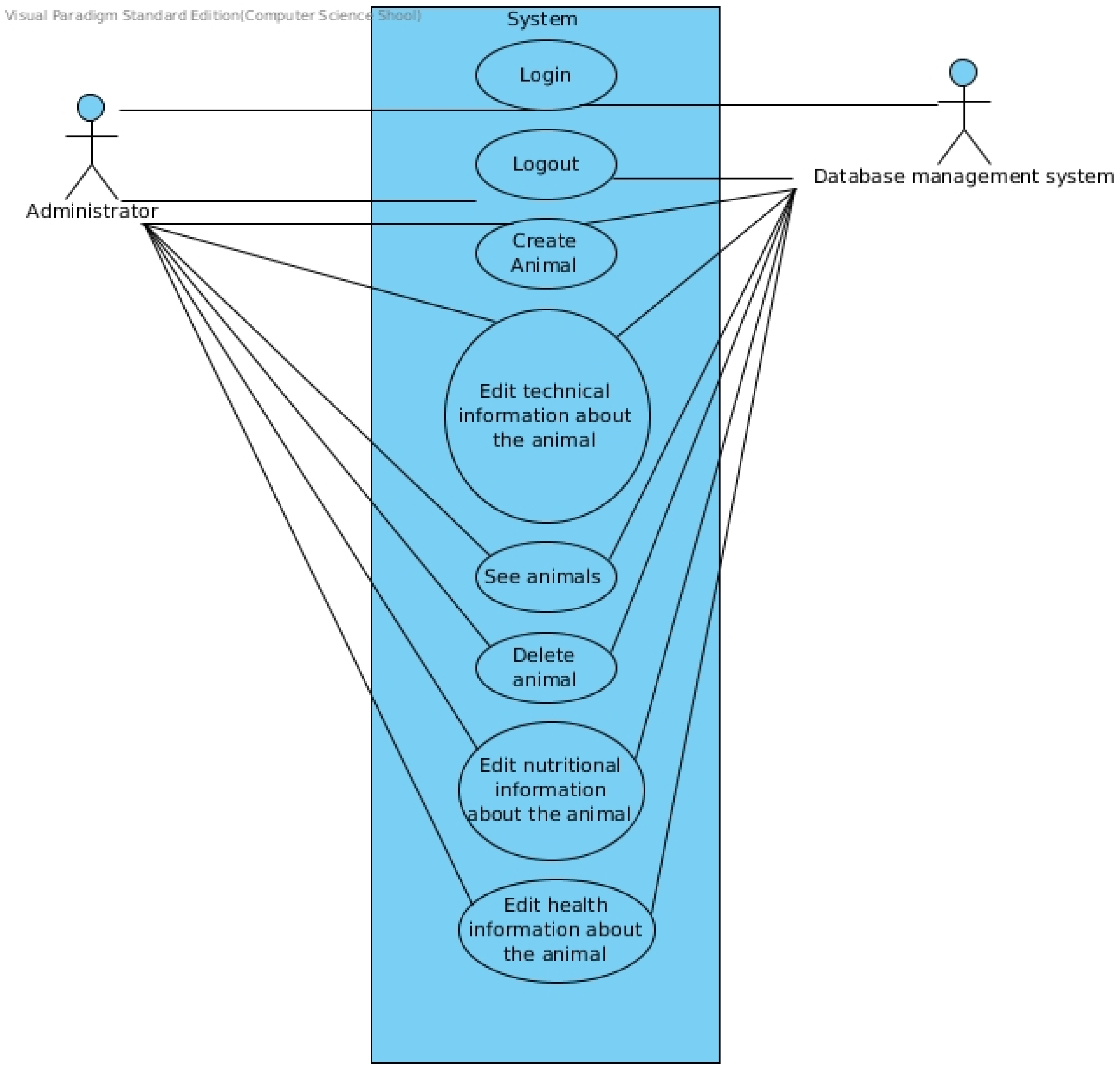
All the information of the animals and users is going to be saved in a database that have specific capacities of redundancy.

2 Functional Requirements

Functional Requirements	Objective	Priority
1	Authenticate to the system	1
2	Create an animal	2
3	Consult the different animals of the system	3
4	Edit the technical information of the animal	4
5	Edit the health information of the animal	5
6	Edit the food information of the animal	6
7	Delete an animal	7
8	Close session	8

Note: the priority of the requirements is in descendant order, the lower the number the higher the priority.

2.1 Use Cases



Visual Paradigm Standard Edition(Computer Science Shool)
System
Login
Logout
Create Animal
Edit technical information about the animal
See animals
Delete animal
Edit nutritional information about the animal
Edit health information about the animal
Administrator
Database management system

Figure 2: Use case Diagram

3 Architecture

Administrator computer

- Computer with Ubuntu Desktop 16.04 of 64 bits
- 8GB of RAM
- 1TB SSD
- Local IP

4 Project Development Software

- Programming Language: Java 8
- Developing UML Environment: Visual Paradigm Community Edition for UML v12.2
- Programming Environment: Eclipse Mars 2 with Maven support

5 Functional Groups

Functional Group 1 → Authenticate to the system

Functional Group 2 → Create an animal

Functional Group 3 → Consult the different animals of the system

Functional Group 4 → Edit the technical information of the animal

Functional Group 5 → Edit the health information of the animal

Functional Group 6 → Edit the food information of the animal

Functional Group 7 → Delete an animal

Functional Group 8 → Close session

6 Iterations

Iteration 0 → Planning

Iteration 1 → Functional group 1

Iteration 2 → Functional group 2

Iteration 3 → Functional group 3

Iteration 4 → Functional group 4

Iteration 5 → Functional group 5

Iteration 6 → Functional group 6

Iteration 7 → Functional group 7

Iteration 8 → Functional group 8

Iteration 9 → Integration and deploy

6.1 Phases and Iterations

Initial phase → Iterations 0, 1 and 2

Development phase → Iterations 3 and 4

Construction phase → Iterations from 5 to 10

Transition phase → Iteration 11

6.2 Duration of the Development and Construction Phases

	Iter 3	Iter 4	Iter 5	Iter 6	Iter 7	Iter 8	Iter 9	Iter 10
Requirements	1	1	1	1	1	1	1	1
Analysis	4	3	1	2	2	2	2	1
Design	9	7	4	5	5	5	4	3
Implementation	15	20	6	10	10	10	8	4
Testing	1	1	1	1	1	1	1	1
Total	30	32	13	19	19	19	16	10

The total duration is 158 hours, equivalent to 19 days and 6 hours.

The starting date of the project is on June 8th and the end of the project is on July 5th. The Gantt Diagram of the project is in section 7.2

7 Schedule

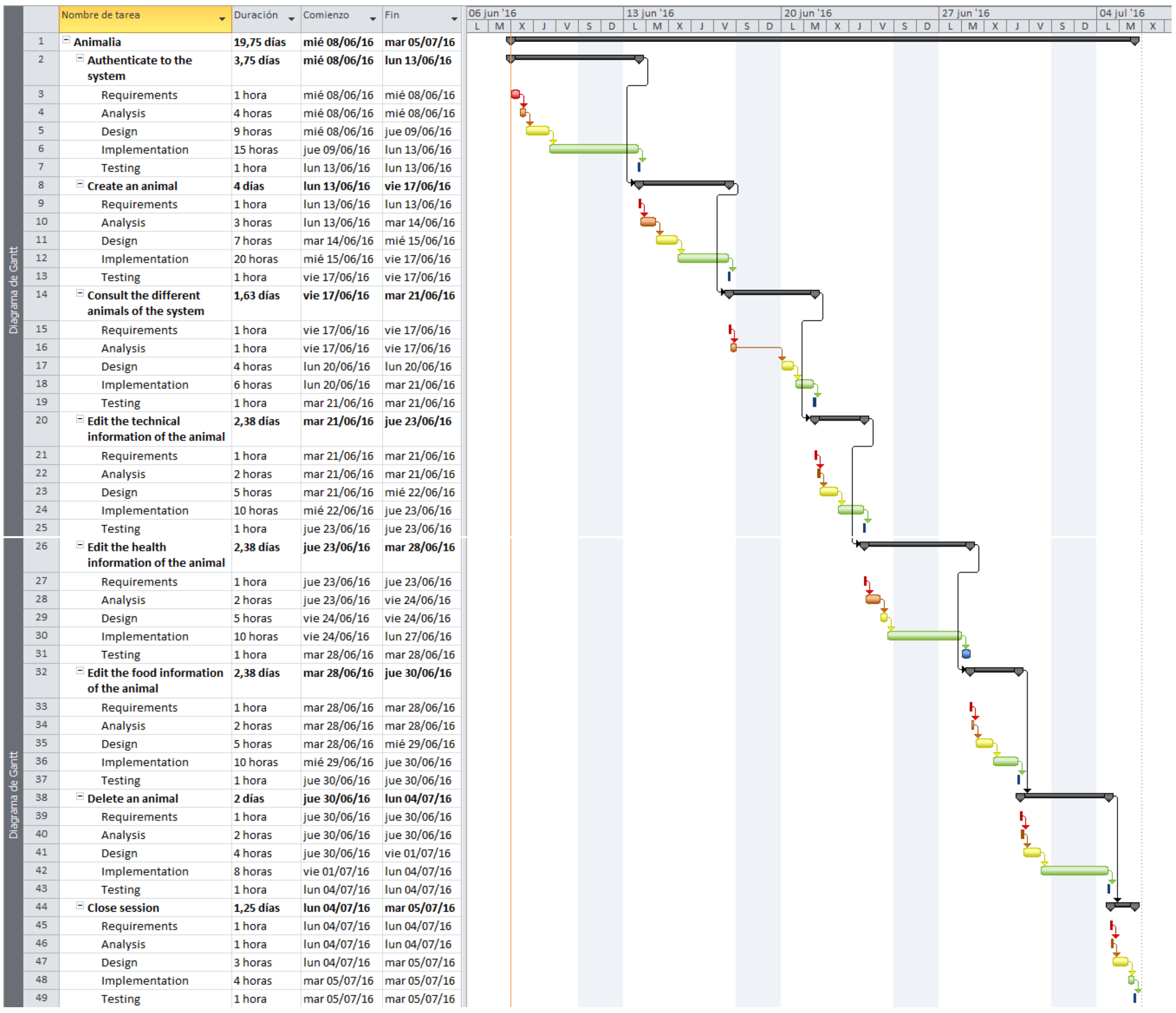
7.1 Transition Phase Schedule

The Configuration Management Plan should be done before the project starts, but the final documentation is going to be done at the same time as the project.

The integration and deploy of the software is going to be done on July 6th, 7th and 8th.

7.2 Development and Construction Schedule

The scheduling of the development and construction phase has been done and the Gantt Diagram obtained is the next one:



	Nombre de tarea	Duración	Comienzo	Fin
1	**Animalia**	**19,75 días**	**mié 08/06/16**	**mar 05/07/16**
2	**Authenticate to the system**	**3,75 días**	**mié 08/06/16**	**lun 13/06/16**
3	Requirements	1 hora	mié 08/06/16	mié 08/06/16
4	Analysis	4 horas	mié 08/06/16	mié 08/06/16
5	Design	9 horas	mié 08/06/16	jue 09/06/16
6	Implementation	15 horas	jue 09/06/16	lun 13/06/16
7	Testing	1 hora	lun 13/06/16	lun 13/06/16
8	**Create an animal**	**4 días**	**lun 13/06/16**	**vie 17/06/16**
9	Requirements	1 hora	lun 13/06/16	lun 13/06/16
10	Analysis	3 horas	lun 13/06/16	mar 14/06/16
11	Design	7 horas	mar 14/06/16	mié 15/06/16
12	Implementation	20 horas	mié 15/06/16	vie 17/06/16
13	Testing	1 hora	vie 17/06/16	vie 17/06/16
14	**Consult the different animals of the system**	**1,63 días**	**vie 17/06/16**	**mar 21/06/16**
15	Requirements	1 hora	vie 17/06/16	vie 17/06/16
16	Analysis	1 hora	vie 17/06/16	vie 17/06/16
17	Design	4 horas	lun 20/06/16	lun 20/06/16
18	Implementation	6 horas	lun 20/06/16	mar 21/06/16
19	Testing	1 hora	mar 21/06/16	mar 21/06/16
20	**Edit the technical information of the animal**	**2,38 días**	**mar 21/06/16**	**jue 23/06/16**
21	Requirements	1 hora	mar 21/06/16	mar 21/06/16
22	Analysis	2 horas	mar 21/06/16	mar 21/06/16
23	Design	5 horas	mar 21/06/16	mié 22/06/16
24	Implementation	10 horas	mié 22/06/16	jue 23/06/16
25	Testing	1 hora	jue 23/06/16	jue 23/06/16
26	**Edit the health information of the animal**	**2,38 días**	**jue 23/06/16**	**mar 28/06/16**
27	Requirements	1 hora	jue 23/06/16	jue 23/06/16
28	Analysis	2 horas	jue 23/06/16	vie 24/06/16
29	Design	5 horas	vie 24/06/16	vie 24/06/16
30	Implementation	10 horas	vie 24/06/16	lun 27/06/16
31	Testing	1 hora	mar 28/06/16	mar 28/06/16
32	**Edit the food information of the animal**	**2,38 días**	**mar 28/06/16**	**jue 30/06/16**
33	Requirements	1 hora	mar 28/06/16	mar 28/06/16
34	Analysis	2 horas	mar 28/06/16	mar 28/06/16
35	Design	5 horas	mar 28/06/16	mié 29/06/16
36	Implementation	10 horas	mié 29/06/16	jue 30/06/16
37	Testing	1 hora	jue 30/06/16	jue 30/06/16
38	**Delete an animal**	**2 días**	**jue 30/06/16**	**lun 04/07/16**
39	Requirements	1 hora	jue 30/06/16	jue 30/06/16
40	Analysis	2 horas	jue 30/06/16	jue 30/06/16
41	Design	4 horas	jue 30/06/16	vie 01/07/16
42	Implementation	8 horas	vie 01/07/16	lun 04/07/16
43	Testing	1 hora	lun 04/07/16	lun 04/07/16
44	**Close session**	**1,25 días**	**lun 04/07/16**	**mar 05/07/16**
45	Requirements	1 hora	lun 04/07/16	lun 04/07/16
46	Analysis	1 hora	lun 04/07/16	lun 04/07/16
47	Design	3 horas	lun 04/07/16	mar 05/07/16
48	Implementation	4 horas	mar 05/07/16	mar 05/07/16
49	Testing	1 hora	mar 05/07/16	mar 05/07/16

Figure 3: Gantt Diagram

8 Costs

8.1 Staff Costs

According to the scheduled of the project the costs of the staff is the following:

	€/h	Total Hours	Coste Total
Requirements	90	8	720
Analyst	90	17	1530
Designer	60	42	2520
Developer	30	83	2490
Tester	20	8	160
		TOTAL	7420

8.2 License Costs

1. Visual paradigm: 1000€

TOTAL COSTS → 1000€

8.3 Architecture Costs

The architecture used was specified in section 3. The cost of the different computers are:

- Administrator Computer → 750€

TOTAL COSTS → 750€

9 Decisions

- our monolithic application will use MVC pattern in which there will be a persistance in order to communicate with the database
- The login and the animal management will use the same database
- In the diagrams, we have not included the constructor and the getters/setters of the objects, in order to simplify the design class diagram.

10 Iterations

10.1 Iteration2

10.1.1 Analysis class diagram

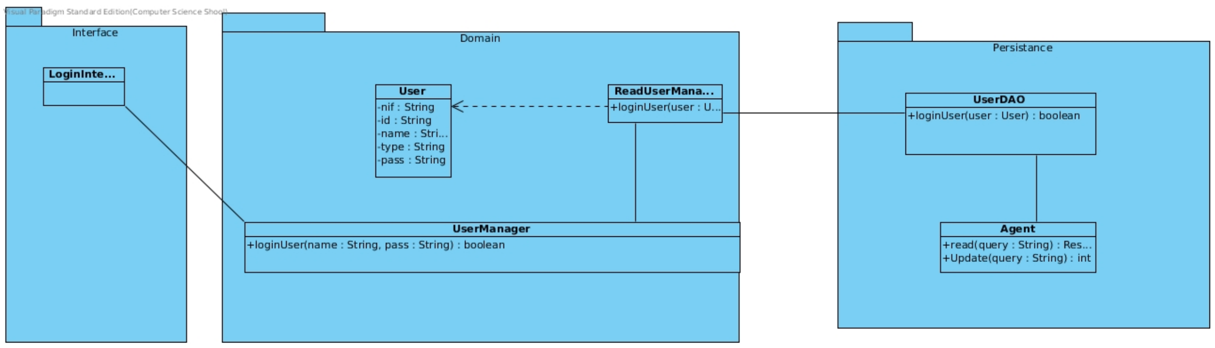

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
LoginInte...
Domain
User
-nif : String
-id : String
-name : Stri...
-type : String
-pass : String
ReadUserMana...
+loginUser(user : U...
UserManager
+loginUser(name : String, pass : String) : boolean
Persistance
UserDAO
+loginUser(user : User) : boolean
Agent
+read(query : String) : Res...
+Update(query : String) : int

Figure 4: Login User Analysis Diagram

10.1.2 Design class diagram

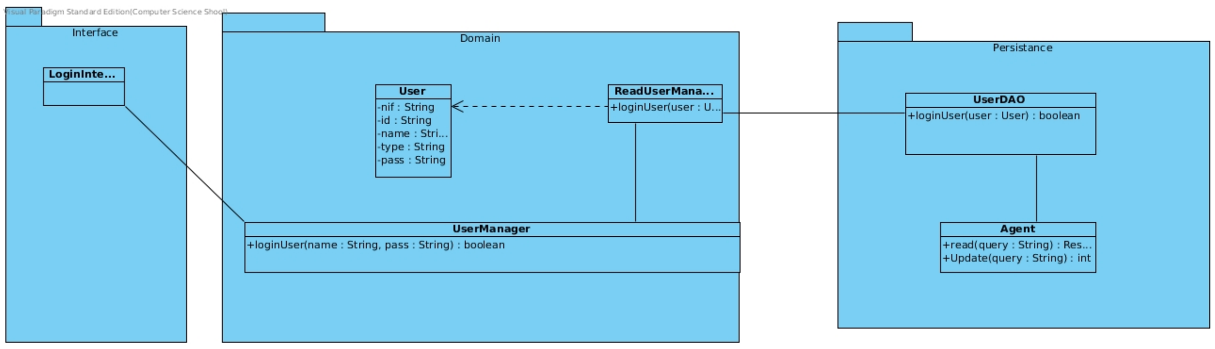

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
LoginInte...
Domain
User
-nif : String
-id : String
-name : Stri...
-type : String
-pass : String
ReadUserMana...
+loginUser(user : U...
UserManager
+loginUser(name : String, pass : String) : boolean
Persistance
UserDAO
+loginUser(user : User) : boolean
Agent
+read(query : String) : Res...
+Update(query : String) : int

Figure 5: Login User Class Diagram

10.1.3 Sequence design diagram principal scenario

The administrator introduce his login data (user and password) into the login interface, and click the login button. **LoginInterface** creates an **UserManager** object and calls the **login** login function, which parameters are the user and the password.

Then, the **UserManager** creates a **ReadUserManager** object which is the one that communicates with the Persistante. This object creates an **User** object, and then, creates a **UserDAO** object and calls **loginUser** function, passing the **User** object in the arguments.

Finally, **UserDAO** object communicates with the **Agent**, calling **read** function, and passing the sql query in the argument. If the operation is correct, it will pase true through all the functions until it gets to the **LoginInterface**

10.1.4 Sequence design diagram alternative scenario

The alternative scenario follows the same flow of the principal scenario, but it doesn't log in the system correctly, so it returns false until it gets to the **LoginInterface**, and it shows a message indicating that the login is incorrect.

10.1.5 Implementation

We have created the following classes:

- LoginInterface
- UserManager
- LoginManager

- DaoUser
- Agent

10.2 Iteration 3

10.2.1 Analysis class diagram

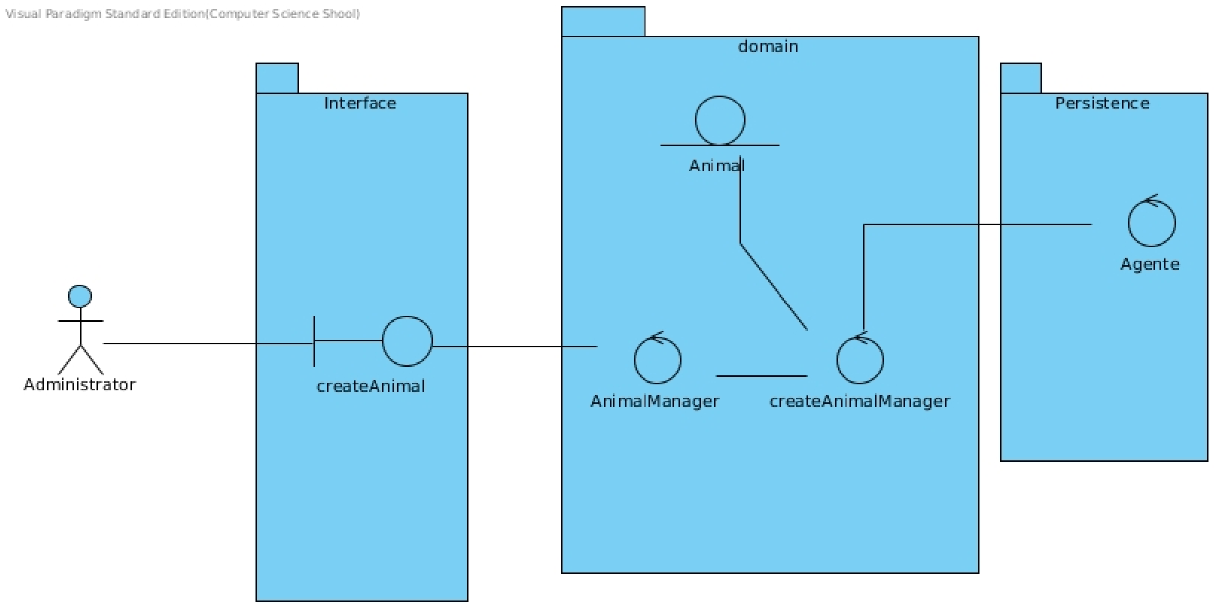


Figure 6: Create Animal Analysis Diagram

10.2.2 Design class diagram

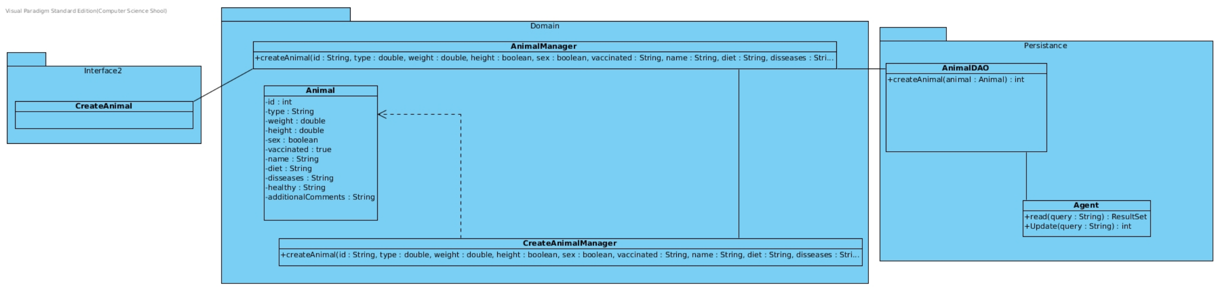

Visual Paradigm Standard Edition(Computer Science Shool)
Interface2
CreateAnimal
Domain
AnimalManager
+createAnimal(id : String, type : double, weight : double, height : boolean, sex : boolean, vaccinated : String, name : String, diet : String, disseases : Stri...
Animal
-id : int
-type : String
-weight : double
-height : double
-sex : boolean
-vaccinated : true
-name : String
-diet : String
-disseases : String
-healthy : String
-additionalComments : String
CreateAnimalManager
+createAnimal(id : String, type : double, weight : double, height : boolean, sex : boolean, vaccinated : String, name : String, diet : String, disseases : Stri...
Persistance
AnimalDAO
+createAnimal(animal : Animal) : int
Agent
+read(query : String) : ResultSet
+Update(query : String) : int

Figure 7: Create Animal Class Diagram

10.2.3 sequence design diagram

The administrator insert information about the animal through the **CreateAnimal** interface. Then, **CreateAnimal** sends to **AnimalManager** the information about the animal, and **AnimalManager** calls to CreateAnimalManager through the function *createAnimal*. Then, **CreateAnimal** calls to **AnimalDAO** throught the function *createAnimal* and then it calls to the **Agent**'s function, **update**, which returns an integer: 1 if the animal has been inserted and 0 if not.

10.2.4 Implementation

We have created the following classes:

- CreateAnimal
- AnimalManager
- CreateAnimalManager
- AnimalDAO

Also, we have added a new function to the Agent, called *update*.

10.3 Iteration 4

10.3.1 Analysis class diagram

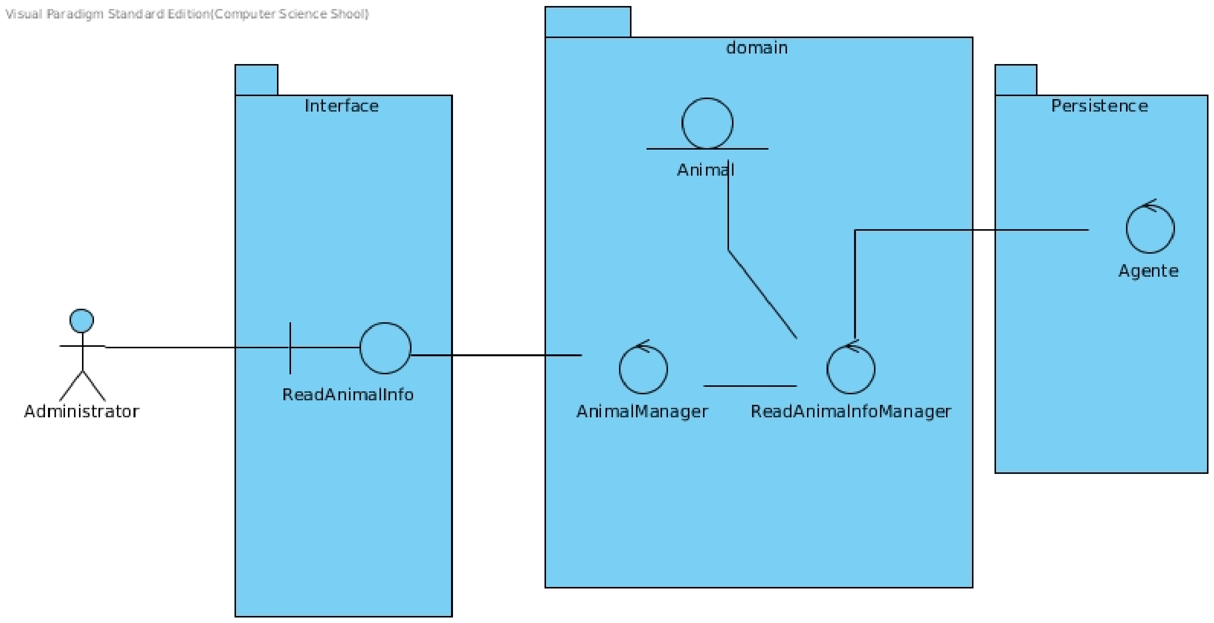


Figure 8: Read Animal Analysis Diagram

10.3.2 Design class diagram

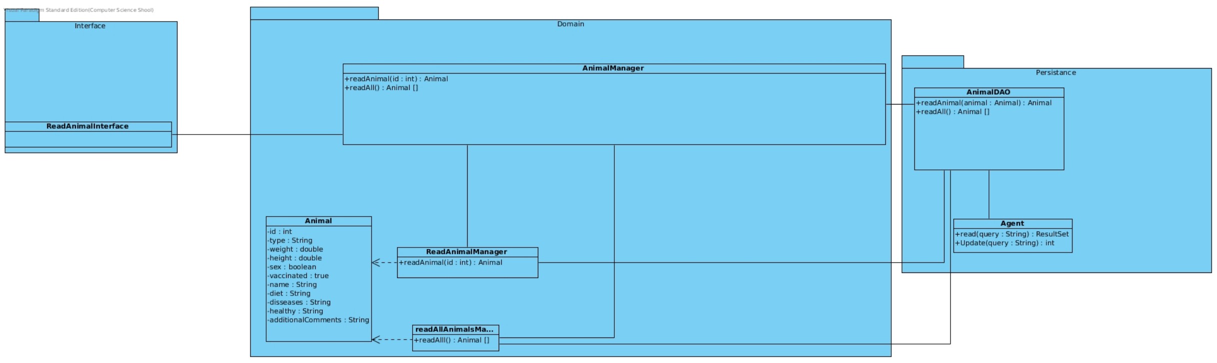


Figure 9: Read Animal Class Diagram

10.3.3 sequence design diagram principal scenarios

The **ReadAnimalInterface** calls to *readAll* function which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *readAll* which belongs to **ReadAllAnimalsManager**. Then, **ReadAllAnimalsManager** calls to *readAll* function through **AnimalDAO** object. Finally, **AnimalDAO** calls to *read* function which belongs to the **Agent** and returns the list of animals.

The other scenario is when the administrator clicks on an animal contained in the list. The **ReadAnimalInterface** calls to *readAnimal* function(pass the id of the animal) which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *readAnimal* which belongs to **ReadAnimalManager**(we pass the id of the animal). Then, **ReadAnimalManager** calls to *readAnimal* function through **AnimalDAO** object. Finally, **AnimalDAO** calls to *read* function which belongs to the **Agent** and returns the animal.

10.3.4 Implementation

We have created the following classes:

- ReadAnimalInterface
- ReadAllAnimalsManager
- ReadAnimalManager

Also, we have modified the classes AnimalDao and AnimalManager

10.4 Iteration 5

10.4.1 Analysis class diagram

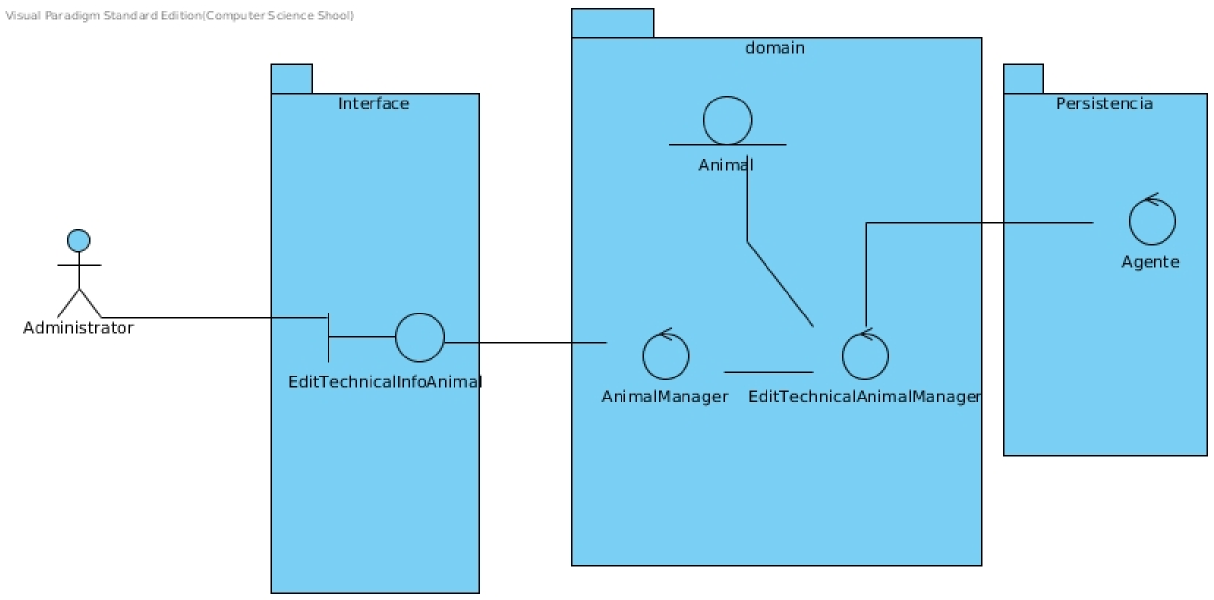


Figure 10: Edit Technical info of the Animal Analysis Diagram

10.4.2 Design class diagram

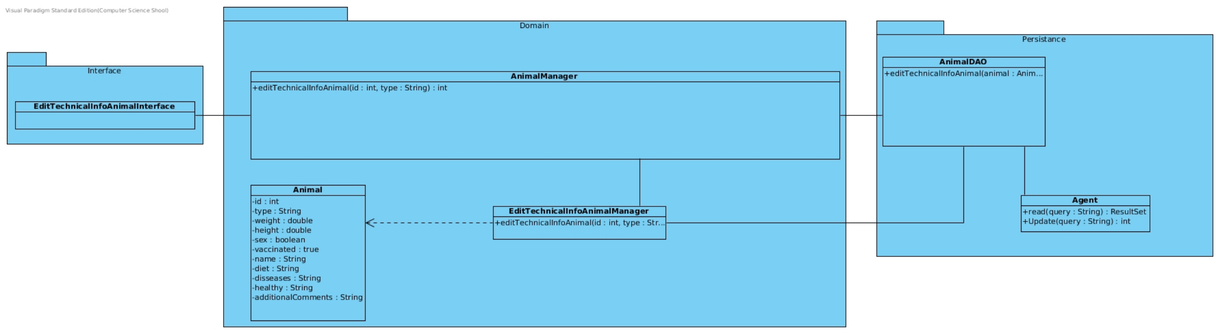

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
EditTechnicalInfoAnimalInterface
Domain
AnimalManager
+editTechnicalInfoAnimal(id : int, type : String) : int
Animal
-id : int
-type : String
-weight : double
-height : double
-sex : boolean
-vaccinated : true
-name : String
-diet : String
-disseases : String
-healthy : String
-additionalComments : String
EditTechnicalInfoAnimalManager
+editTechnicalInfoAnimal(id : int, type : Str...
Persistance
AnimalDAO
+editTechnicalInfoAnimal(animal : Anim...
Agent
+read(query : String) : ResultSet
+Update(query : String) : int

Figure 11: Edit Technical info of the AnimalClass Diagram

10.4.3 sequence design diagram

When the administrator puts the technical info of the animal and clicks in the 'update' button in the interface to modify the animal, **EditTechnicalInfoAnimalInterface** calls to *editTechnicalInfoAnimal* function which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *editTechnicalInfoAnimal* which belongs to **editTechnicalInfoAnimalManager**. Then, **editTechnicalInfoAnimalManager** calls to *editTechnicalInfoAnimal* function through **AnimalDAO** object. Finally, **AnimalDAO** calls to *update* function which belongs to the **Agent** and returns an integer: 1 if the file has been modified succesfully, or 0 if not.

10.4.4 Implementation

We ahve created the following classes:

- EditTechnicalInfoAnimalInterface
- EditTechnicalInfoAnimalManager

Also, we have modified the following classes: AnimalManager and AnimalDAO

10.5 Iteration 6

10.5.1 Analysis class diagram

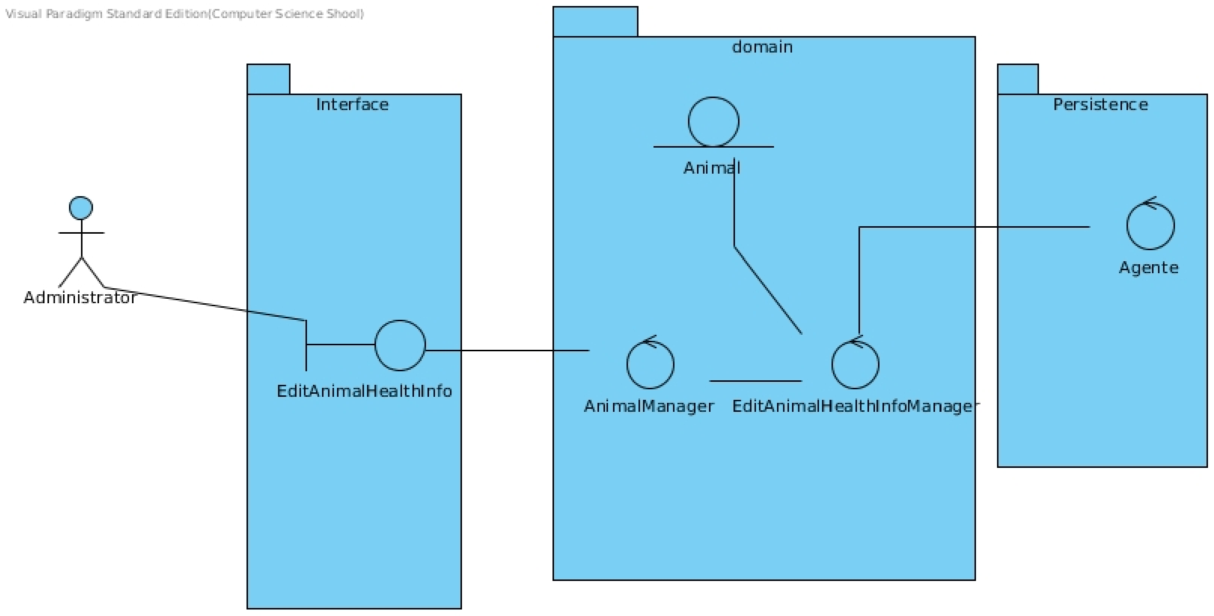


Figure 12: Edit health info Animal Analysis Diagram

10.5.2 Design class diagram

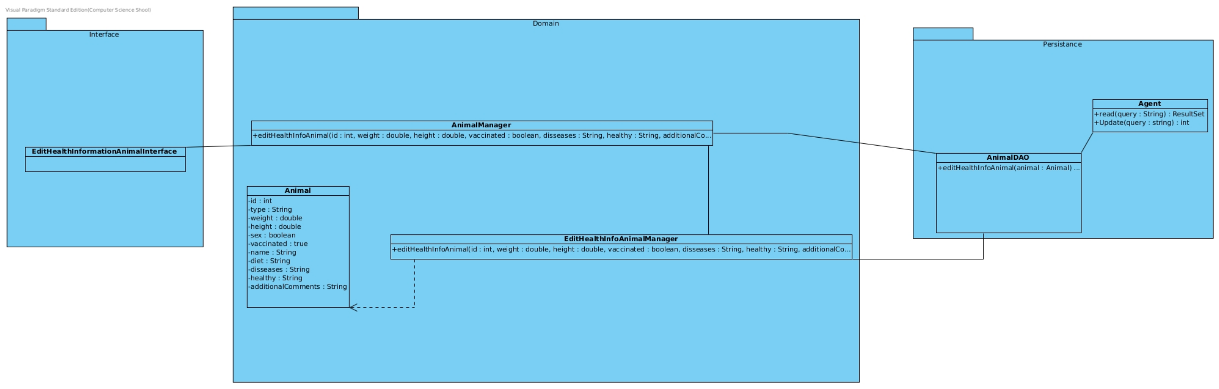

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
EditHealthInformationAnimalInterface
Domain
AnimalManager
+editHealthInfoAnimal(id : int, weight : double, height : double, vaccinated : boolean, disseases : String, healthy : String, additionalCo...
Animal
-id : int
-type : String
-weight : double
-height : double
-sex : boolean
-vaccinated : true
-name : String
-diet : String
-disseases : String
-healthy : String
-additionalComments : String
EditHealthInfoAnimalManager
+editHealthInfoAnimal(id : int, weight : double, height : double, vaccinated : boolean, disseases : String, healthy : String, additionalCo...
Persistance
Agent
+read(query : String) : ResultSet
+Update(query : string) : int
AnimalDAO
+editHealthInfoAnimal(animal : Animal) ...

Figure 13: Edit Health info Animal Class Diagram

10.5.3 sequence design diagram

When the administrator puts the health info of the animal and clicks in the 'update' button in the interface to modify the animal, **EditHealthInfoAnimalInterface** calls to *editHealthInfoAnimal* function which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *editHealthInfoAnimal* which belongs to **editHealthInfoAnimalManager**. Then, **editHealthInfoAnimalManager** calls to *editHealthInfoAnimal* function through **AnimalDAO** object. Finally, **AnimalDAO** calls to *update* function which belongs to the **Agent** and returns an integer: 1 if the file has been modified succesfully, or 0 if not.

10.5.4 Implementation

We have created the following classes:

- editHealthInfoAnimalInterface
- editHealthInfoAnimalManager

Also, we have modified AnimalDAO and AnimalManager

10.6 Iteration 7

10.6.1 Analysis class diagram

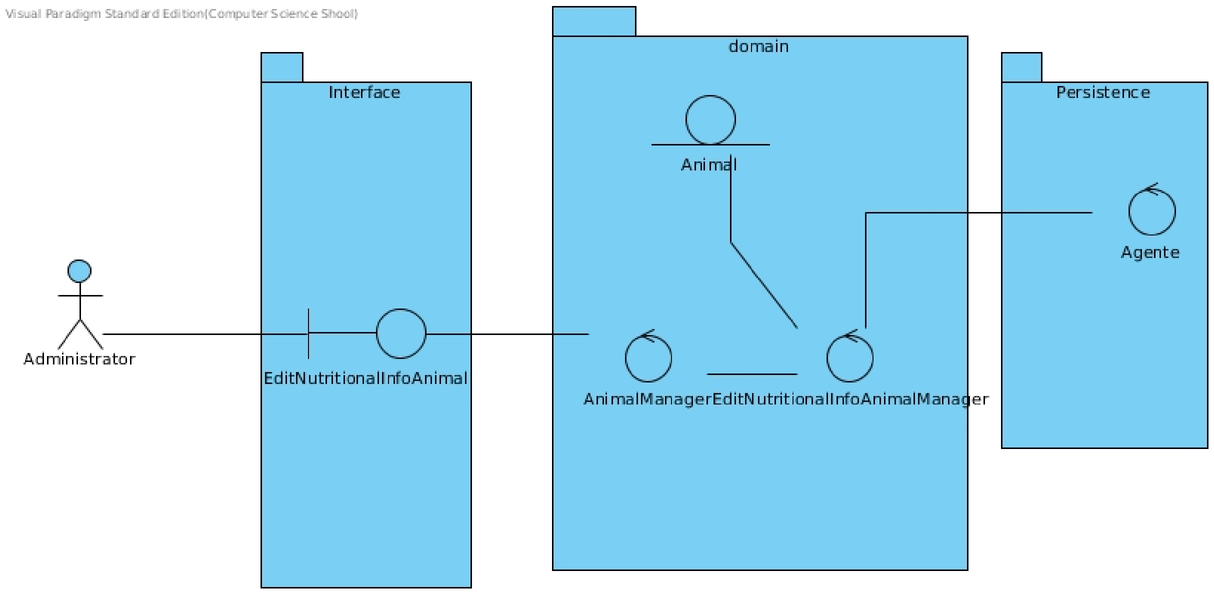

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
domain
Persistence
Animal
Agente
Administrator
EditNutritionalInfoAnimal
AnimalManager
EditNutritionalInfoAnimalManager

Figure 14: Edit nutritional info Animal Analysis Diagram

10.6.2 Design class diagram

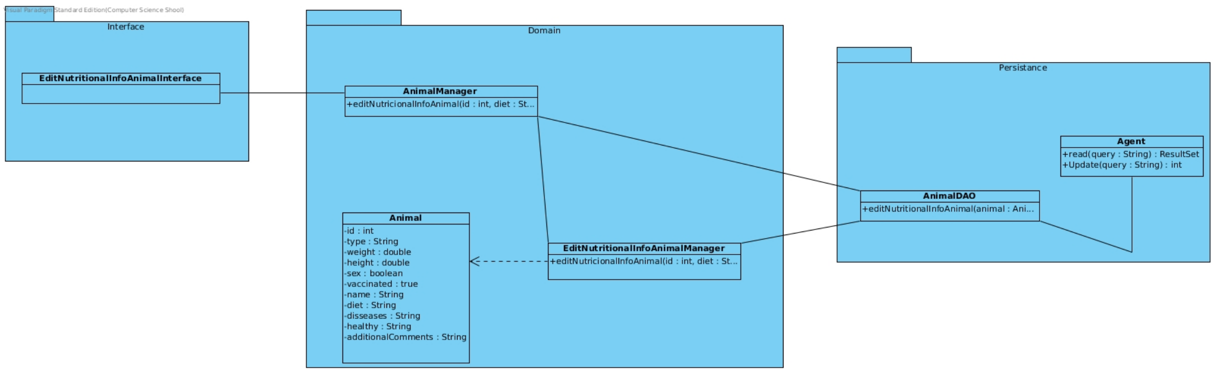

Visual Paradigm Standard Edition(Computer Science Shool)
Interface
EditNutritionalInfoAnimalInterface
Domain
AnimalManager
+editNutricionalInfoAnimal(id : int, diet : St...
Animal
-id : int
-type : String
-weight : double
-height : double
-sex : boolean
-vaccinated : true
-name : String
-diet : String
-disseases : String
-healthy : String
-additionalComments : String
EditNutritionalInfoAnimalManager
+editNutricionalInfoAnimal(id : int, diet : St...
Persistance
Agent
+read(query : String) : ResultSet
+Update(query : String) : int
AnimalDAO
+editNutritionalInfoAnimal(animal : Ani...

Figure 15: Edit nutritional info Animal Class Diagram

10.6.3 sequence design diagram

When the administrator puts the nutritional info of the animal and clicks in the 'update' button in the interface to modify the animal, **editNutricionalInfoAnimalInterface** calls to *editNutricionalInfoAnimal* function which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *editNutricionalInfoAnimal* which belongs to **editNutricionalInfoAnimalManager**. Then, **editNutricionalInfoAnimalManager** calls to *editNutricionalInfoAnimal* function through **AnimalDAO** object. Finally,

AnimalDAO calls to *update* function which belongs to the **Agent** and returns an integer: 1 if the file has been modified succesfully, or 0 if not.

10.6.4 Implementation

We have created the following classes:

- editNutricionalInfoAnimalInterface
- editNutricionalInfoAnimalManager

Also, we have modified AnimalDAO and AnimalManager.

10.7 Iteration 8

10.7.1 Analysis class diagram

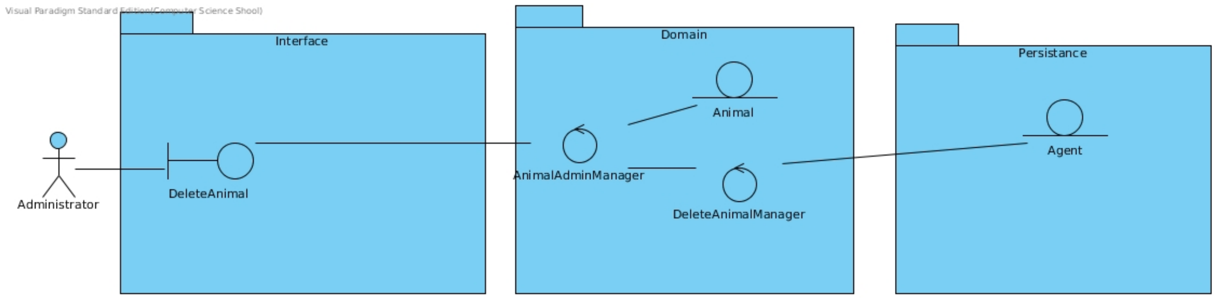


Figure 16: Remove Animal Analysis Diagram

10.7.2 Design class diagram

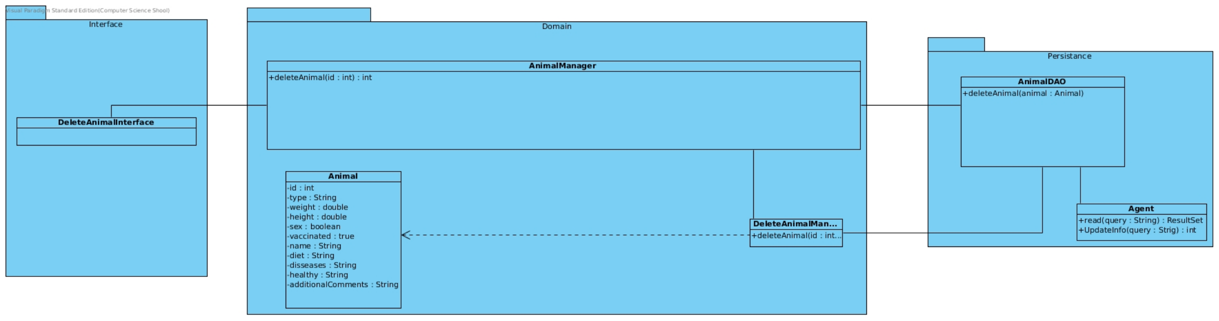


Figure 17: Remove Animal Class Diagram

10.7.3 sequence design diagram

When the administrator puts the nutritional info of the animal and clicks in the 'update' button in the interface to modify the animal, **deleteAnimalInterface** calls to *deleteAnimal* function which belongs to **AnimalManager** class. Then, **AnimalManager** calls to the function *deleteAnimal* which belongs to **deleteAnimalManager**. Then, **deleteAnimalManager** calls to *deleteAnimal* function through **AnimalDAO** object. Finally, **AnimalDAO** calls to *update* function which belongs to the **Agent** and returns an integer: 1 if the file has been modified succesfully, or 0 if not.

10.7.4 Implementation

We have created the following clases:

- deleteAnimalInterface

- deleteAnimalManager

Also, we have modified AnimalManager and AnimalDAO classes

10.8 Iteration 9

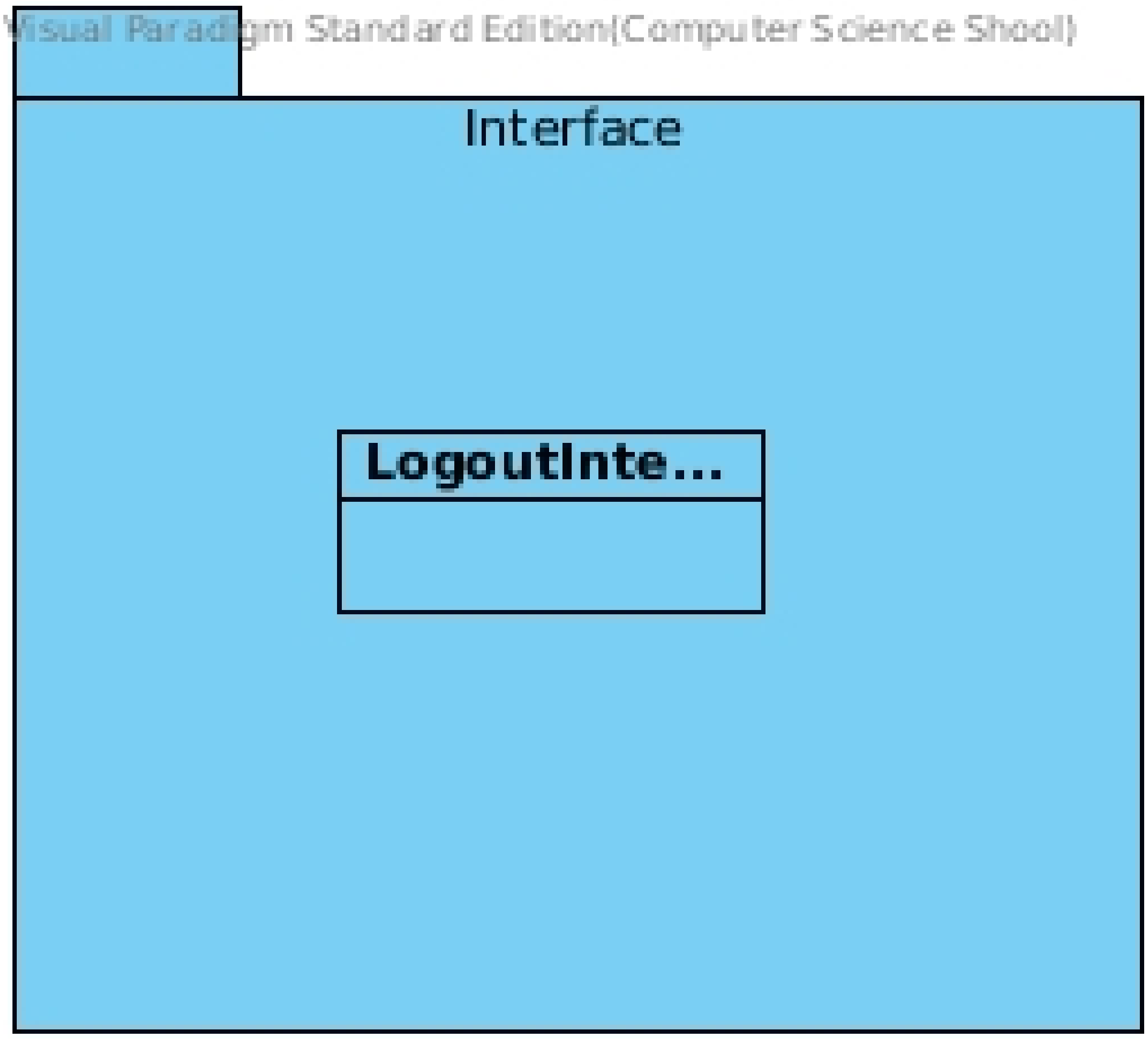


Figure 18: Logout Diagram

10.8.1 sequence design diagram principal scenario

This diagram is the most simple. It dereference the object and goes to the **loginInterface** without going into de Domain or the Persistance.

10.8.2 Implementation

We have created LogoutInterface.