Evaluación de prestaciones de algoritmos conscientes de la red para la planificación de contenedores ligeros sobre infraestructuras paralelas distribuidas

Ángel Roberto García Serpa

Asignatura: 71013035 – Diseño de Software
Año de la práctica: 2019-2020
Centro Asociado al que pertenece: 160016 - Bogotá,
Tutor que le da asistencia:
Datos personales:
Nombre y apellidos: Ángel Roberto García Serpa
DNI: 52539216V
Contacto: 683596728 - argserpa@gmail.com
Localidad de residencia: Ciempozuelos, Madrid
Datos de coste:
Horas de dedicación al estudio de los contenidos: 30
Nº de actividades no evaluables realizadas y horas de dedicación: 0.0
Horas de dedicación para realizar esta actividad: 20

25 de agosto de 2020

Contenidos

Lista de Figuras

Parte I
Enunciado y planteamiento del caso de estudio.

El escenario en el que se situará el funcionamiento del caso de uso (pregunta 2), consiste en un paquete para la **gestión sanitaria de una explotación ganadera.** (SanGranja).

Dicho módulo (o paquete) forma parte de una aplicación dedicada a la gestión integral ganadera (i-FarM).

El negocio general del usuario de estas aplicaciones es el cuidado y la cría de animales para el consumo humano de sus productos.

Esta descripción del escenario, en el que se ubica el caso de uso que se va a implementar en el ejercicio, se refiere al módulo SanGranja. Lógicamente, interactúa con algunos de los servicios que provee i-FarM (considerado externo a SanGranja) pero, también, con otros servicios aportados por sistemas de apoyo externos (como el Sistema de Gestión del Almacén común con toda la Información –Datos– del sistema integral que, además, también daría soporte a i-FarM).

De esta forma, el propósito fundamental del módulo SanGranja es ocuparse de:

- La gestión Sanitaria de los animales y de las instalaciones. Es decir, de la monitorización, diagnóstico, tratamiento y seguimiento de la salud de cada animal.

El ciclo vital de cada individuo suele transcurrir en unas instalaciones o recintos (corrales) en los que se supervisan sus condiciones ambientales, la alimentación especializada o los tratamientos sanitarios que recibe. Para realizar esta funcionalidad, se va a suponer que cada animal lleva implantado un dispositivo biométrico que recoge la información necesaria. Estos datos se mantienen en el Sistema de Información global de la explotación, en la parte que le corresponde tanto a i-FarM como a SanGranja,.

De la misma manera que ocurre con la aplicación general i-FarM, **el objetivo fundamental de la construcción del módulo SanGranja es que**, además de funcionar colaborativa e independientemente de i-FarM, **sea fácilmente escalable y adaptable a la dimensión de la explotación, a la estructura organizativa con que se realiza, a la naturaleza de su ganadería, a las enfermedades que les afectan y a sus tratamientos**.

Parte II
Resolución del problema.

1 Sección 1. Evaluación de los Casos de Uso

1.1 Ej 1. (0'5 puntos)

Represente, en un diagrama UML de casos de uso, los casos de uso primarios (Elementary Business Process) más importantes, sus actores principales, los de apoyo y las interacciones correspondientes para el módulo SanGranja.

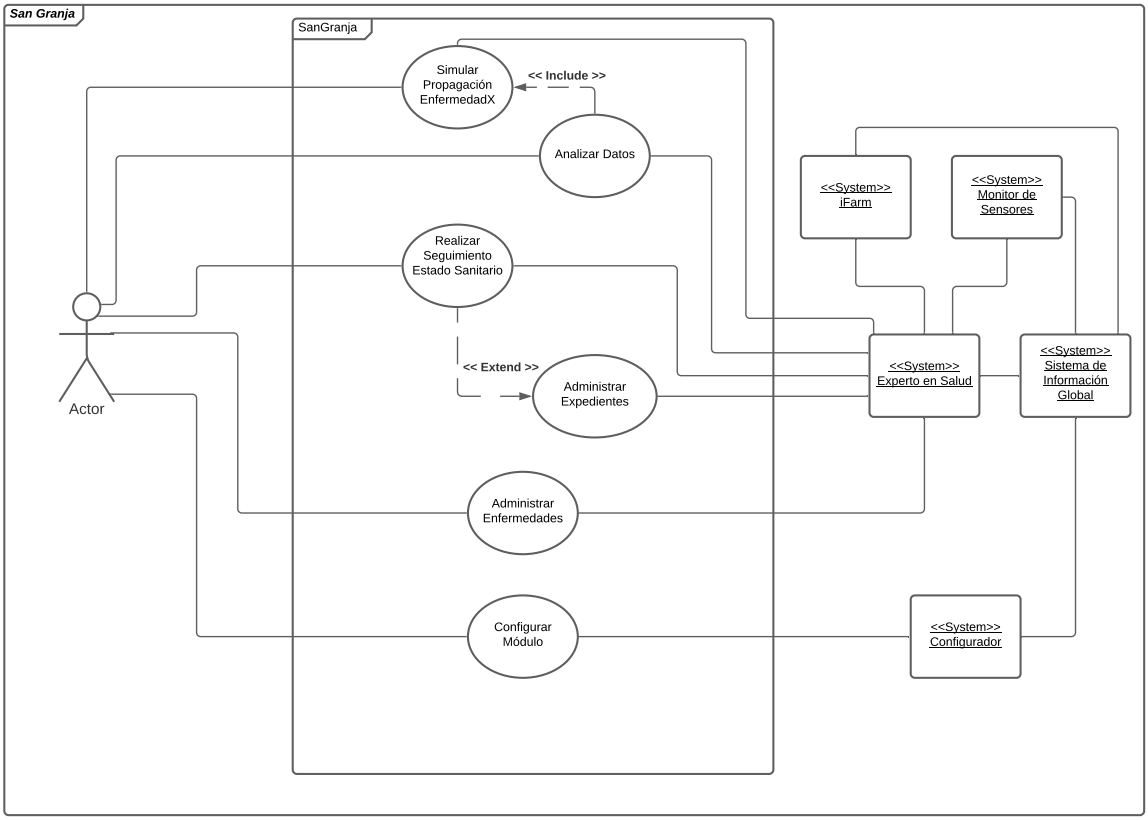

Figura 1: Evaluación de casos de uso

1.2 Ej 2. (1 punto)

Con la siguiente descripción del caso de uso ≪SimularPropagaciónEnfermedad_X ≫, escríbalo en un formato completo (se recomienda la variante 'en dos columnas') y un estilo esencial (excluyendo los detalles técnicos de nivel bajo). Incluya tanto el flujo en el escenario principal de éxito como 2 extensiones o flujos alternativos que pudieran ser frecuentes:

Caso de uso: SimularPropagaciónEnfermedad_X
Formato completo (variante 'a dos columnas'), estilo esencial.
Evolución típica de los acontecimientos

Acciones del actor	Acciones del sistema
1 - El usuario ingresa al sistema para realizar una simulación sobre una enfermedad definida.	
	2 - El sistema muestra los datos de la simulación (corrales, días y caracterización de la enfermedad con, entre otros datos, su vector de contagio).
3 - El usuario ejecuta la simulación con los valores por defecto asignados por el sistema para los atributos *número de días*, *corrales* y *vector de contagio* de la simulación.	
	4 - El sistema realiza la simulación con los parámetros recibidos, muestra los resultados y pregunta si desea realizar otra simulación.
5- Revisa los resultados y da por terminada la simulación.	

Alternativas

- 3a El **usuario** modifica los valores de los atributos que desea y da comienzo a la simulación, el flujo continúa en **4**.
- 5b El **usuario** desea realizar una nueva simulación con valores diferentes, con lo que selecciona dicha opción y pulsa continuar. El flujo sigue desde **2**.

2 Sección 2. Evaluación del Modelado Conceptual

2.1 Ej 3. (3 puntos)

En relación con el caso de uso anterior,≪SimularPropagaciónEnfermedad_X ≫, construya un Modelo de Dominio y represéntelo en notación UML. Represente los objetos conceptuales, las relaciones relevantes entre ellos, su cardinalidad y los atributos candidatos de los objetos.

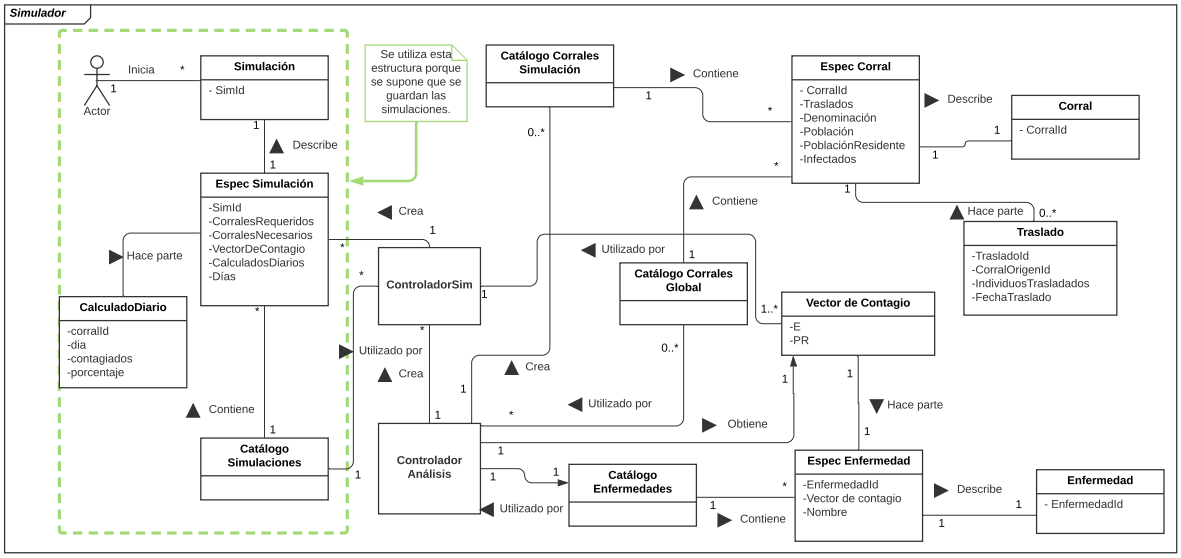

Figura 2: Modelo de Dominio.

3 Sección 3. Evaluación de la Asignación de Responsabilidades y Diseño de la Interacción.

3.1 Ej 4. (3 puntos)

Circunscrito al caso de uso que nos ocupa, ≪SimularPropagaciónEnfermedad_X≫, construya un Diagrama de Interacción en UML. Represente el actor, sus eventos y el paso de mensajes entre cada instancia de las clases software que componen el sistema para este caso de uso.

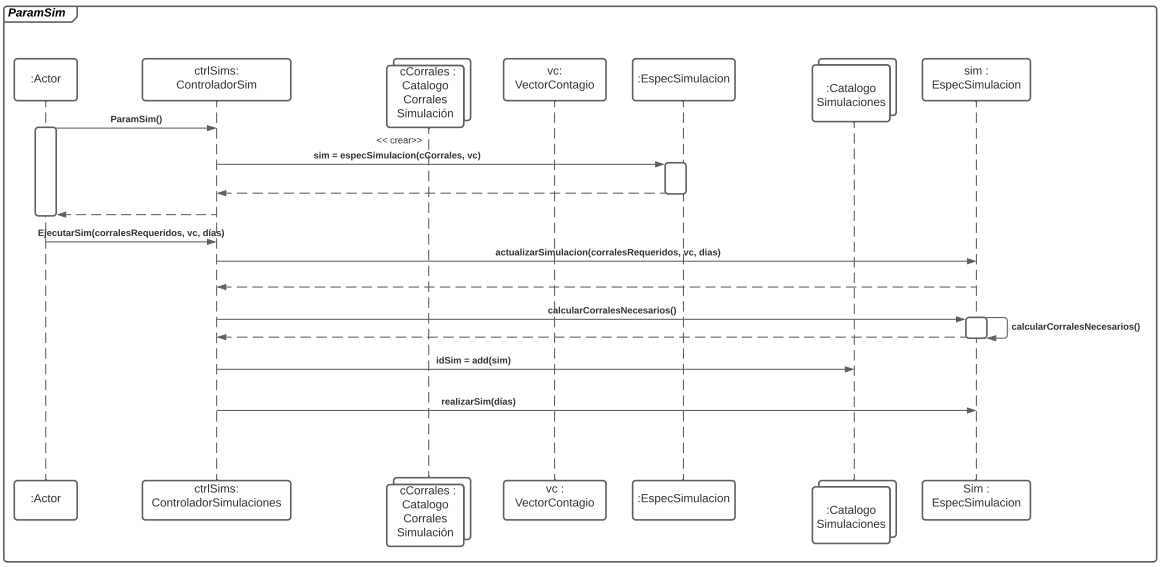

Figura 3: Diagrama de Secuencia ParamSim.

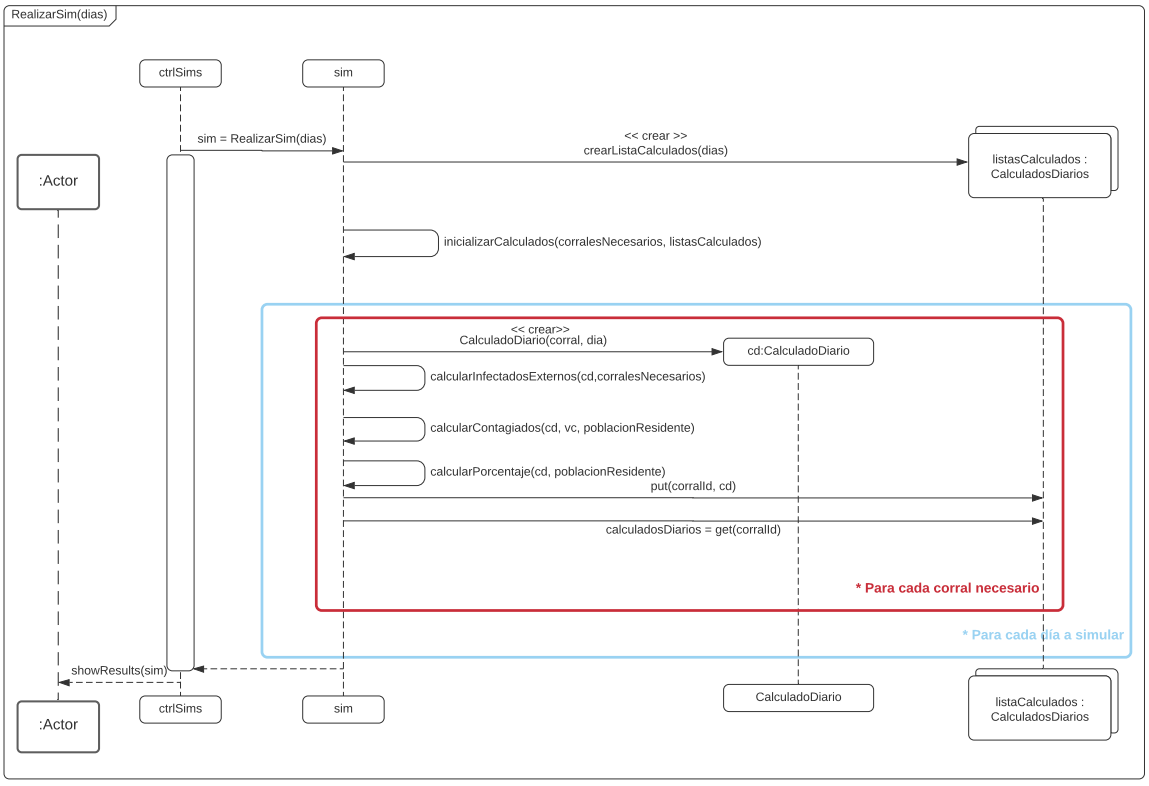

Figura 4: Diagrama de Secuencia RealizarSim.

3.2 Ej 5. (1 punto)

A partir del Diagrama de Interacción presentado en la pregunta 4, escriba y desarrolle el contrato de la operación '*ParamSim*' que corresponde a la inicialización del caso de uso (la creación e inicialización de las instancias que se requieran en ese momento) y a la asignación de los valores requeridos para realizar los cálculos de la simulación.

Contrato CO1: ParamSim

Operación:	paramSim()
Referencias cruzadas:	SimularPropagaciónEnfermedad_X
Precondiciones:	Hay una instancia ControladorAnálisis creado. Hay una instancia ControladorSim creada. Hay una instancia del CatálogoCorralesSimulación creado con el cálculo de la población residente de cada corral realizado. Hay una instancia de VectorDeContagio creada, lo que quiere decir que la enfermedad para la que se realizará la simulación ya ha sido elegida. Hay una instancia de CatalogoDeSimulaciones creada.
Postcondiciones:	Se creó una EspecSimulación, sim con los parámetros por defecto. Se actualizó sim, con la información recibida por el usuario. Se rellenó la lista de corrales CorralesNecesarios del objeto sim. Se obtuvo el id de la Presa y se asoció a su descripción. Se añadió sim al catálogo de simulaciones.

4 Sección 4. Evaluación del Diagrama de Clases de diseño.

4.1 Ej 6. (1 punto)

Elabore un diagrama de clases para el caso de uso que se está tratando ≪SimularPropagaciónEnfermedad_X≫ (DCD), centrado en la clase que va a implementar la responsabilidad más característica del caso de uso, la que mejor defina la naturaleza de lo que se hace en él (ControlSim, Calculador, o como lo haya llamado en los diagramas precedentes). Represente los nombres de todos los atributos, asociaciones (con su navegabilidad) y los métodos (excepto 'setters' y 'getters' irrelevantes), tanto de esa clase como de las que estén directamente involucradas con ella en el funcionamiento del caso de uso.

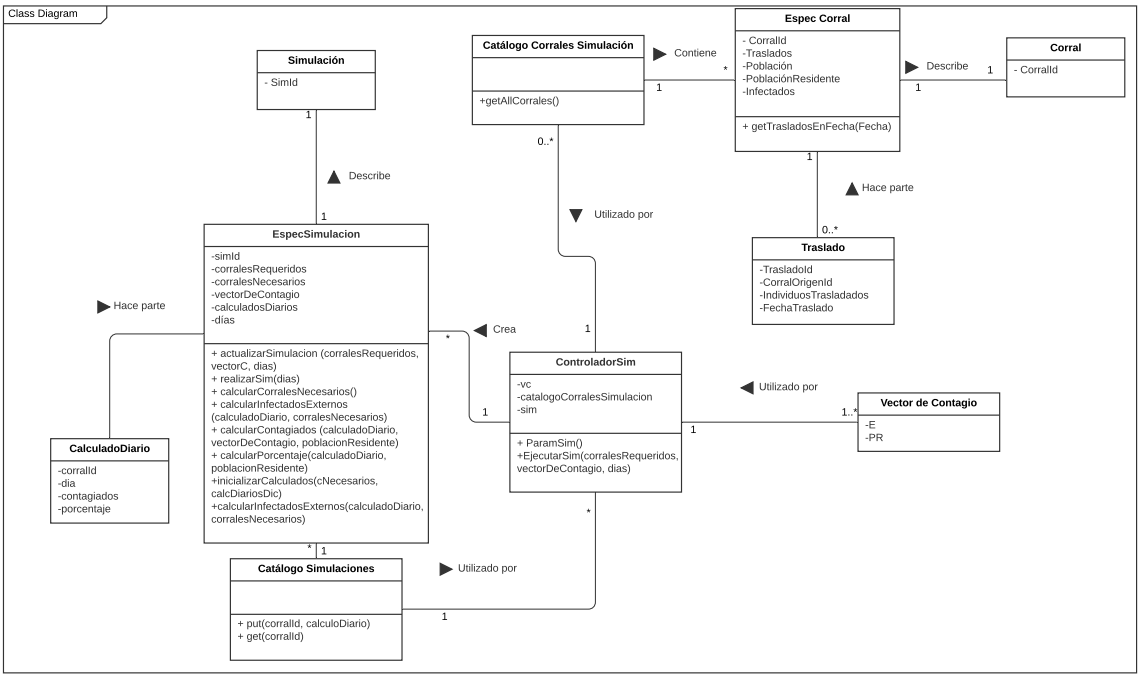

Figura 5: Diagrama de Clases SimularPropagaciónEnfermedad_X.

5 Sección 5. Evaluación de la Transformación del Diseño en Código

5.1 Ej 7. (0'5 puntos)

A partir de los anteriores diagramas de clases y colaboraciones, elabore y defina la clase que haya establecido, en el desarrollo anterior, como responsable de controlar la correcta secuencia de acciones en el caso de uso ≪SimularPropagaciónEnfermedad_X≫. Incluya las definiciones de todas las variables que la componen (miembros), pero escriba solamente la definición completa del cuerpo para el método principal o los métodos más significativos: ≪se omite el método≫. Ignore los pequeños detalles de sintaxis -el objetivo es evaluar la capacidad fundamental para transformar el diseño en código-. Utilice la sintaxis de Java.

5.1.1 ControladorSim.java

```
1  package sanGranja;
2
3  import java.util.ArrayList;
4
5  public class ControladorSim {
6
7    VectorDeContagio vc;
8    CatalogoCorralesSimulacion  catalogoCorralesSimulacion;
9    EspecSimulacion sim;
10
11   public EspecSimulacion ParamSim(){
12     this.sim = new EspecSimulacion(catalogoCorralesSimulacion,vc);
13     return this.sim;
14   }
15
16   public void EjecutarSim(ArrayList<Integer> corralesRequeridos,
17                VectorDeContagio vc, Integer dias){
18     this.sim.actualizarSimulacion(corralesRequeridos, vc, dias);
19   }
20 }
```

5.1.2 EspecSimulacion.java

```
1  package sanGranja;
2
3  import java.util.ArrayList;
4  import java.util.Calendar;
5  import java.util.Date;
6  import java.util.HashMap;
7  import java.util.Map;
8
9  public class EspecSimulacion {
10
11   Long simId;
12   ArrayList<Integer> corralesRequeridos;
13   HashMap<Integer, EspecCorral> corralesNecesarios;
14   VectorDeContagio vectorContagio;
15   ArrayList<CalculadoDiario>calculadosDiarios;
16   Integer dias;
17
18   public EspecSimulacion(CatalogoCorralesSimulacion catalogoCorralesSimulacion,
19               VectorDeContagio vc2) {
20     catalogoCorralesSimulacion.getAllCorrales().forEach(c->{
21       this.corralesNecesarios.put(c.corralId, c);
22     });
23     this.vectorContagio = vc2;
24   }
25
26
27   public Long actualizarSimulacion (ArrayList<Integer> corralesRequeridos,
28                   VectorDeContagio vectorC, Integer dias)
29   {
30     this.corralesRequeridos = corralesRequeridos;
31     this.vectorContagio = vectorC;
32     this.dias = dias;
33     return this.simId;
34   }
35
36   public void calcularCorralesNecesarios(){
37     ArrayList<EspecCorral> corrales = new ArrayList<EspecCorral>();
38     this.corralesRequeridos.forEach( c->{
39       corrales.add(this.corralesNecesarios.get(c));
40     });
41     this.corralesNecesarios.clear();
42     corrales.forEach(c->{
43       this.corralesNecesarios.put(c.corralId, c);
44     });
45   }
46
47   public void realizarSim(Integer dias){
48     HashMap<Integer, ArrayList<CalculadoDiario>> calculadosDiarios =
49     new HashMap<Integer, ArrayList<CalculadoDiario>>();
50     inicializarCalculados(this.corralesNecesarios, calculadosDiarios);
51     for (int i = 1; i<=dias; ++i){
52     /*para cada dia, calculo el numero de infectados de cada corral*/
53       int dia = i;
54       corralesNecesarios.values().forEach( c ->{
55         CalculadoDiario cd = new CalculadoDiario(c.corralId, dia);
56         Integer contagiadosExternos =
57         calcularInfectadosExternos(cd, this.corralesNecesarios);
58
59         Integer contagiados = calcularContagiados(cd,
60         this.vectorContagio, c.poblacionResidente, contagiadosExternos);
61
62         cd.contagiados = contagiados;
63         Double pct = calcularPorcentaje(cd, contagiados);
64         cd.porcentaje = pct;
```

```
65          ArrayList<CalculadoDiario> listaCalculadosDiario =
66          calculadosDiarios.get(c.corralId);
67
68          if (listaCalculadosDiario == null){
69            listaCalculadosDiario = new ArrayList<CalculadoDiario>();
70          }
71          listaCalculadosDiario.add(cd);
72          calculadosDiarios.put(c.corralId, listaCalculadosDiario);
73          this.calculadosDiarios = calculadosDiarios.get(c.corralId);
74        });
75      }
76
77    }
78
79    private void inicializarCalculados(HashMap<Integer, EspecCorral> cNecesarios,
80            HashMap<Integer, ArrayList<CalculadoDiario>> calcDiariosDic) {
81
82      corralesNecesarios.values().forEach( c ->{
83        ArrayList<CalculadoDiario> calculadosPorCorral = new ArrayList<CalculadoDiario>();
84        CalculadoDiario cd = new CalculadoDiario(c.corralId, c.infectados,
85        (double)c.infectados/c.poblacionResidente);
86        calculadosPorCorral.add(cd);
87        calcDiariosDic.put(cd.corralId, calculadosPorCorral);
88      });
89    }
90
91    public Integer calcularInfectadosExternos(CalculadoDiario calculadoDiario,
92                          HashMap<Integer, EspecCorral> corralesNecesarios){
93      Integer infectados = 0;
94      /*obtengo el dia que estamos calculando */
95      Calendar calendar = Calendar.getInstance();
96      calendar.add(Calendar.DAY_OF_YEAR, calculadoDiario.dia);
97      Date fecha = calendar.getTime();
98      /*para dicho dia, calculo el numero de infectados de cada corral  */
99      for(Map.Entry<Integer, EspecCorral> entry : corralesNecesarios.entrySet()){
100       ArrayList<Traslado> traslados = entry.getValue().getTrasladosEnFecha(fecha);
101       for (int j = 0;j < traslados.size();++j){
102         Traslado t = traslados.get(j);
103         EspecCorral corral = corralesNecesarios.get(t.corralOrigenId);
104         Double porcentaje = (double)corral.infectados/(double)corral.poblacion;
105         Double trasladadosInfectados = t.individuosTrasladados.doubleValue() * porcentaje;
106         infectados += trasladadosInfectados.intValue();
107       }
108       infectados += calculadoDiario.contagiados;
109     }
110     return infectados;
111   }
112
113   public Integer calcularContagiados (CalculadoDiario calculadoDiario,
114     VectorDeContagio vectorDeContagio, Integer poblacionResidente, Integer cExternos){
115     //comprobamos que tenemos otros elementos es la lista de calculados diarios de cada corra
116     Integer contagiadosActuales = this.calculadosDiarios.size() == 0 ?
117      this.calculadosDiarios.get(this.calculadosDiarios.size()-1).contagiados: 0;
118
119     Double res = (cExternos *contagiadosActuales * vectorDeContagio.e * vectorDeContagio.pR)
120      * (1 - contagiadosActuales/poblacionResidente);
121
122     return res.intValue();
123   }
124   public Double calcularPorcentaje (CalculadoDiario calculadoDiario, Integer poblacionReside
125     return (double)calculadoDiario.contagiados / poblacionResidente;
126   }
127 }
```

5.1.3 CatalogoCorralesSimulacion.java

```
package sanGranja;

import java.util.ArrayList;

public class CatalogoCorralesSimulacion {

  ArrayList<EspecCorral> corrales;

  public ArrayList<EspecCorral> getAllCorrales()
  {
    return corrales;
  }
}
```

5.1.4 EspecCorral.java

```
package sanGranja;

import java.util.ArrayList;
import java.util.Date;

public class EspecCorral {
  Integer corralId;
  ArrayList<Traslado> traslados;
  Integer poblacion;
  Integer poblacionResidente;
  Integer infectados;

  /**
  * Devuelve los traslados de un corral para una fecha determinada*/
  public ArrayList<Traslado> getTrasladosEnFecha(Date fecha){
    ArrayList<Traslado> trasladosHoy = new ArrayList<Traslado>();
    traslados.forEach(t ->{
    /*Seguramente la comparacion sea diferente, porque solo hay que comparar dia,mes y ano,
    pero se entiende el concepto.*/
      if (t.fechaTraslado.compareTo(fecha) == 0){
        trasladosHoy.add(t);
      }
    });
    return trasladosHoy;
  }

  public EspecCorral() {}

}
```

5.1.5 CalculadoDiario.java

```
package sanGranja;

public class CalculadoDiario {
  Integer corralId;
  Integer dia;
  Integer contagiados;
  Double porcentaje;

  public CalculadoDiario() {
    this.corralId = 0;
    this.dia = 0;
    this.contagiados = 0;
    this.porcentaje = 0.0;
  }
  public CalculadoDiario(Integer cId, Integer dia) {
    this.corralId = cId;
    this.dia = dia;
```

```
18    }
19    public CalculadoDiario(Integer corralId2, Integer infectados,
20    double d) {
21      this.corralId = corralId2;
22      this.dia = 0;
23      this.contagiados = infectados;
24      this.porcentaje = d;
25    }
26  }
```

5.1.6 VectorDeContagio.java

```
1  package sanGranja;
2
3  public class VectorDeContagio {
4    Double e;
5    Double pR;
6
7  }
```

5.1.7 Traslado.java

```
1  package sanGranja;
2
3  import java.util.Date;
4
5  public class Traslado {
6    Long TrasladoId;
7    Integer corralOrigenId;
8    Integer individuosTrasladados; /* Cantidad de individuos implicados
9     en el traslado .*/
10   Date fechaTraslado;
11 }
```

6 Sección 6. Preguntas opcionales BP. Motivación.

6.1 Ej 8. (0'5 puntos)

Indique qué principios GRASP ha utilizado en el ejercicio y qué responsabilidades ha asignado guiándose por ellos.

EspecSimulacion es un experto ya que contiene toda la información necesaria para saber el resultado del caso de uso en estudio.

ControladorAnálisis es creador ya que se encarga de crear tanto el *ControladorSim* como los objetos que necesita éste para realizar la simulación(*catálogoCorrarlesSimulación* y *vectorDeContagio*). *ControladorSim* también es creador ya que es el encargado de crear y agregar (actualizar) los objetos *EspecSimulacion*.

Se ha seguido el principio de bajo acoplamiento y alta cohesión al utilizar catálogos de especificaciones de objetos, con esto se logra que no se vean comprometidos los objetos que se comparten con el resto de la aplicación. El hecho de que se genere acoplamiento en la clase *EspecSimulacion* viene dado, principalmente, por la necesidad de que algunos objetos, como el *VectorDeContagio* y los corrales que se necesitan para la realización de la simulación, puedan ser modificados "localmente" y que dicha modificación sea inocua para los mismos objetos en el resto de la aplicación.

Tanto ControladorAnalisis como ControladorSim siguen el patrón Controlador. El primero se encarga de preparar los objetos necesarios para que se pueda llevar a cabo el caso de uso, manejando los eventos del sistema (por ejemplo el manejo del sistema de alarmas que sirve de disparo para la selección de la enfermedad en el caso de uso) y delegando en el controlador de caso de uso cuando ha conseguido los datos necesarios. Por su parte, ControladorSimulacion se encarga de recibir y enviar mensajes al actor y de enviar los mensajes a los obejtos que se encargarán de las operaciones que llevan a cabo el caso de uso.

6.2 Ej 9. (0'5 puntos)

Indique qué patrones GoF ha utilizado en el ejercicio y qué mejoras ha obtenido, con su uso, en la elaboración de este desarrollo o en el comportamiento final del código que ha diseñado.

Al ser un caso de uso reducido, donde la mayor parte de objetos han sido creados antes de entrar al caso de uso, no existen entidades o sistemas externos a los que nos debamos conectar y la resolución es realtivamente sencilla de impolementar, no se han seguido patrones de este tipo y no me ha sido fácil encontrarlos una vez realizada la práctica.