PETLINK

1/04/2025

1 DE ABRIL DE 2025

MÉTODOS FORMALES EN INGENIERÍA DEL SOFTWARE
ADRIÁN MACHUCA MORILLA - JAVIER ARIAS FUENTES- ÁLVARO ORTA LÓPEZ – ADRIÁN GÓMEZ RODRÍGUEZ

Índice

Introducción

En este trabajo, nos enfocaremos en el diseño y análisis de una red social destinada a propietarios de mascotas, cuyo propósito es facilitar la conexión entre animales con intereses similares y ofrecer recomendaciones especializadas sobre su alimentación y cuidados. Para lograrlo, desarrollaremos un modelo detallado que represente la estructura del sistema y sus principales reglas de funcionamiento.

A lo largo del documento, presentaremos un diagrama de clases que describe los elementos clave de la plataforma y las relaciones entre ellos, justificando las decisiones tomadas en su diseño. También estableceremos un conjunto de restricciones en OCL que permitan asegurar la coherencia del sistema, abordando aspectos como la cantidad máxima de mascotas por propietario, los criterios de compatibilidad para establecer amistad entre animales y la validación de productos recomendados por profesionales veterinarios, entre otras.

Además, definiremos contratos de operaciones esenciales dentro del sistema, estableciendo condiciones previas y resultados esperados para garantizar que las interacciones cumplan con las reglas establecidas. Cada uno de estos contratos será acompañado de una explicación sobre su importancia y su aplicación dentro de la plataforma.

Por último, incluiremos una reflexión sobre los diferentes retos a los que nos hemos enfrentado durante el desarrollo de la práctica, los aprendizajes adquiridos y posibles mejoras que podrían implementarse en el futuro para hacer la red social más efectiva y funcional.

Diagrama de Clases

Para el desarrollo de esta aplicación, hemos identificado y estructurado diversas clases que representan los elementos esenciales para su funcionamiento.

En este apartado detallaremos la función de cada clase, la información que almacena y las asociaciones que establecen entre ellas.

Son las siguientes:

Usuario

Representa a los individuos que utilizan la plataforma. Contiene atributos básicos como un identificador y un nombre de usuario. Esta clase es la base para diferentes tipos de usuarios dentro del sistema.

Propietario

Extiende la clase Usuario y representa a los dueños de mascotas. Además de los atributos heredados, incluye datos como nombre, DNI y un contador de fotos publicadas diariamente. Un propietario puede tener hasta 4 mascotas asociadas.

Mascota

Representa a los animales registrados por los propietarios. Cada mascota tiene atributos como nombre, especie, raza, fecha de nacimiento, estado de salud y comportamiento. También puede estar marcada como "perfil suspendido" en caso de comportamiento agresivo, que sería otro atributo de esta clase. Además, la clase almacena información sobre si las vacunas están al día y permite la interacción con otras mascotas mediante una relación de amistad.

Vacuna

Almacena información sobre las vacunas registradas en la plataforma. Incluye un identificador y la edad recomendada para su aplicación. Está asociada a la clase Mascota, permitiendo verificar si una mascota tiene sus vacunas actualizadas, lo que es un requisito obligatorio para publicar anuncios de adopción.

Producto

Representa los productos disponibles en la plataforma para ser recomendados a las mascotas. Contiene atributos como identificador, nombre, imagen, categoría según la especie de la mascota, certificación veterinaria y un porcentaje de fiabilidad en la recomendación.

Veterinario

Esta clase representa a los profesionales veterinarios registrados en el sistema. Incluye un identificador, nombre y un número de certificación. Los veterinarios tienen la capacidad de aprobar productos dentro de la plataforma, asegurando que las recomendaciones sean seguras y adecuadas para las mascotas.

Publicación

Representa cualquier tipo de contenido publicado en la plataforma. Contiene información sobre la fecha de publicación, el número de "me gusta" y los comentarios asociados. Esta clase se especializa en distintos tipos de publicaciones, como eventos y anuncios.

Comentario

Los comentarios son elementos clave para fomentar la interacción entre los usuarios. Cada comentario tiene un identificador, un texto y un contador de "me gusta". Se pueden asociar a publicaciones y contar con operaciones para su publicación y eliminación.

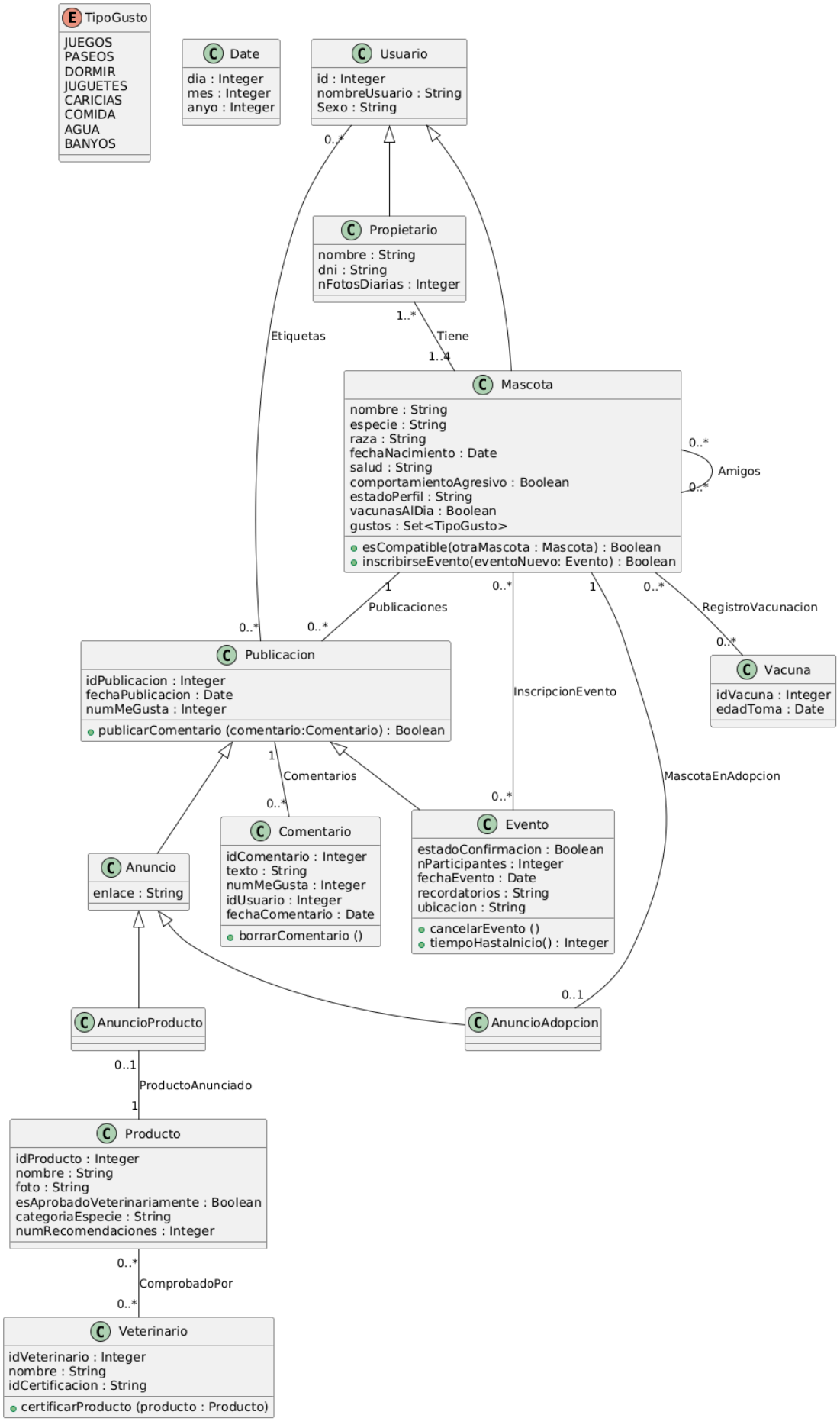
Evento

Extiende la clase Publicación y permite la organización de encuentros entre usuarios y mascotas. Un evento solo se puede confirmar si hay al menos seis usuarios interesados y debe cancelarse si faltan menos de siete días para su realización y no hay suficientes participantes. Además, cuenta con atributos como estado de confirmación, número de participantes y recordatorios.

Anuncio

También extiende la clase Publicación y representa los anuncios dentro de la plataforma. Se divide en dos tipos:

- **Anuncio de Adopción**: Relacionado con la adopción de mascotas. Solo se puede publicar si la mascota en cuestión tiene todas sus vacunas al día.
- **Anuncio de Producto**: Publicidad de productos específicos. Los productos anunciados deben ser compatibles con la especie de la mascota, asegurando que un perro no reciba anuncios de comida para gatos.


TipoGusto
JUEGOS
PASEOS
DORMIR
JUGUETES
CARICIAS
COMIDA
AGUA
BANYOS
Date
dia : Integer
mes : Integer
anyo : Integer
Usuario
id : Integer
nombreUsuario : String
Sexo : String
Propietario
nombre : String
dni : String
nFotosDiarias : Integer
Etiquetas
Tiene
Mascota
nombre : String
especie : String
raza : String
fechaNacimiento : Date
salud : String
comportamientoAgresivo : Boolean
estadoPerfil : String
vacunasAlDia : Boolean
gustos : Set<TipoGusto>
esCompatible(otraMascota : Mascota) : Boolean
inscribirseEvento(eventoNuevo: Evento) : Boolean
Amigos
Publicaciones
RegistroVacunacion
Vacuna
idVacuna : Integer
edadToma : Date
Publicacion
idPublicacion : Integer
fechaPublicacion : Date
numMeGusta : Integer
publicarComentario (comentario:Comentario) : Boolean
InscripcionEvento
MascotaEnAdopcion
Comentarios
Comentario
idComentario : Integer
texto : String
numMeGusta : Integer
idUsuario : Integer
fechaComentario : Date
borrarComentario ()
Evento
estadoConfirmacion : Boolean
nParticipantes : Integer
fechaEvento : Date
recordatorios : String
ubicacion : String
cancelarEvento ()
tiempoHastaInicio() : Integer
Anuncio
enlace : String
AnuncioProducto
AnuncioAdopcion
ProductoAnunciado
Producto
idProducto : Integer
nombre : String
foto : String
esAprobadoVeterinariamente : Boolean
categoriaEspecie : String
numRecomendaciones : Integer
ComprobadoPor
Veterinario
idVeterinario : Integer
nombre : String
idCertificacion : String
certificarProducto (producto : Producto)

Modelado en USE

Clases

Necesidades del sistema

Debido a la limitación de OCL para trabajar con fechas, hemos visto necesario implementar una clase fecha. Además, hemos creado una enumeración ("Enum" – Enumeration) para poder trabajar con los gustos de las mascotas. Para probar el funcionamiento del sistema hemos puesto algunos ejemplos de los muchos que podrían llegar a usarse.

1 Class Date & enum TipoGusto

```
enum TipoGusto {
  JUEGOS,
  PASEOS,
  DORMIR,
  JUGUETES,
  CARICIAS,
  COMIDA,
  AGUA,
  BANYOS
}
```

```
class Date
attributes
  dia : Integer
  mes : Integer
  anyo : Integer
end
```

2 Class Usuario

```
class Usuario
attributes
  id : Integer
  nombreUsuario : String
  Sexo : String
end
```

3 Class Propietario

```
class Propietario < Usuario
attributes
  nombre : String
  dni : String
  nFotosDiarias : Integer
end
```

4 Class Mascota

```
class Mascota < Usuario
attributes
  nombre : String
  especie : String
  raza : String
  fechaNacimiento : Date
  salud : String
  comportamientoAgresivo : Boolean
  estadoPerfil : String
  vacunasAlDia : Boolean
  gustos : Set(TipoGusto) -- coleccion de gustos

operations
  esCompatible(otraMascota : Mascota) : Boolean =
  -- Cogemos los gustos que tienen en comun del enum
  -- y si coindide en al menos 2 → true
  self.gustos→ intersection(otraMascota.gustos)→size() > 1
  inscribirseEvento(eventoNuevo: Evento) : Boolean
end
```

5 Class Vacuna

```
class Vacuna
attributes
  idVacuna : Integer
  edadToma : Date
end
```

6 Class Producto

```
class Producto
attributes
  idProducto : Integer
  nombre : String
  foto : String
  esAprobadoVeterinariamente : Boolean
  categoriaEspecie : String
  numRecomendaciones : Integer
end
```

7 Class Veterinario

```
class Veterinario
attributes
  idVeterinario : Integer
  nombre : String
  idCertificacion : String
operations
  certificarProducto (producto : Producto)
end
```

8 Class Publicacion

```
class Publicacion
attributes
  idPublicacion : Integer
  fechaPublicacion : Date
  numMeGusta : Integer
operations
  publicarComentario (comentario:Comentario) : Boolean
end
```

9 Class Comentario

```
class Comentario
attributes
  idComentario : Integer
  texto : String
  numMeGusta : Integer
  idUsuario : Integer
  fechaComentario : Date
operations
  borrarComentario()
end
```

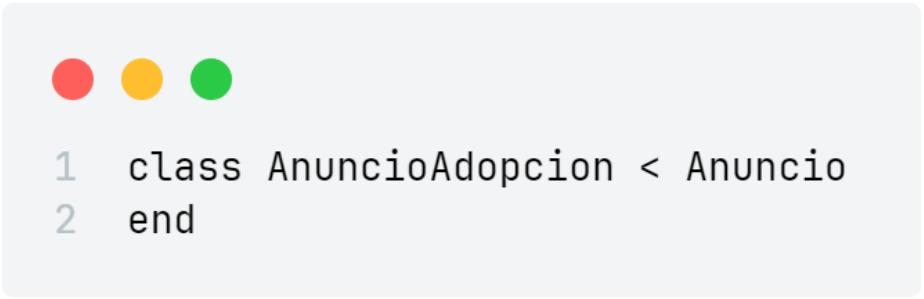
10 Class Evento

```
class Evento < Publicacion
attributes
  estadoConfirmacion : Boolean
  nParticipantes : Integer
  fechaEvento : Date
  recordatorios : String
  ubicacion : String
operations
  cancelarEvento ()
  tiempoHastaInicio() : Integer
end
```

11 Class Anuncio

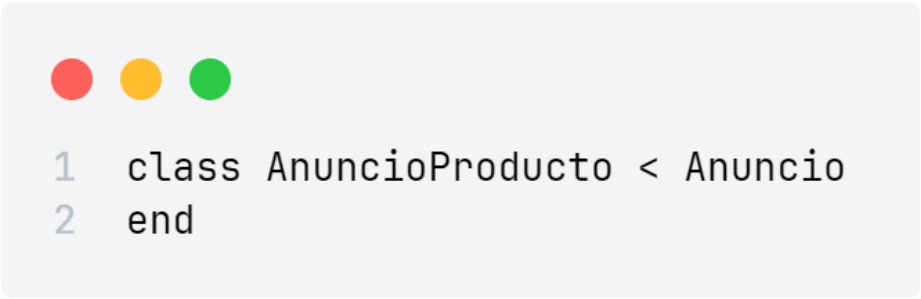
```
class Anuncio < Publicacion
attributes
  enlace : String
end
```

12 Class AnuncioAdopcion



```
class AnuncioAdopcion < Anuncio
end
```

13 Class AnuncioProducto



```
class AnuncioProducto < Anuncio
end
```

Asociaciones

14 Association Tiene

```
association Tiene between
  Propietario [1..*] role propietario
  Mascota [1..4] role mascota
end
```

2 Association RegistroVacunacion

```
association RegistroVacunacion between
  Mascota [0..*] role mascota
  Vacuna [0..*] role vacuna
end
```

3 Association Amigos

```
association Amigos between
  Mascota [0..*] role mascota1
  Mascota [0..*] role mascota2
end
```

4 Association Publicaciones

```
association Publicaciones between
  Mascota [1] role mascota
  Publicacion [0..*] role publicacion
end
```

515 Association Comentarios

```
association Comentarios between
  Publicacion [1] role publicacion
  Comentario [0..*] role comentario
end
```

6 Association Etiquetas

```
association Etiquetas between
  Publicacion [0..*] role publicacionEtiquetada
  Usuario [0..*] role usuario
end
```

16 Association ComprobadoPor

```
association ComprobadoPor between
  Producto [0..*] role producto
  Veterinario [0..*] role veterinario
end
```

8 Association InscripcionEvento

```
association InscripcionEvento between
  Evento [0..*] role evento
  Mascota [0..*] role mascotaInscrito
end
```

9 Association MascotaEnAdopcion

```
association MascotaEnAdopcion between
  AnuncioAdopcion [0..1] role anuncioAdopcion
  Mascota [1] role mascotaEnAdopcion
end
```

10 Association ProductoAnunciado

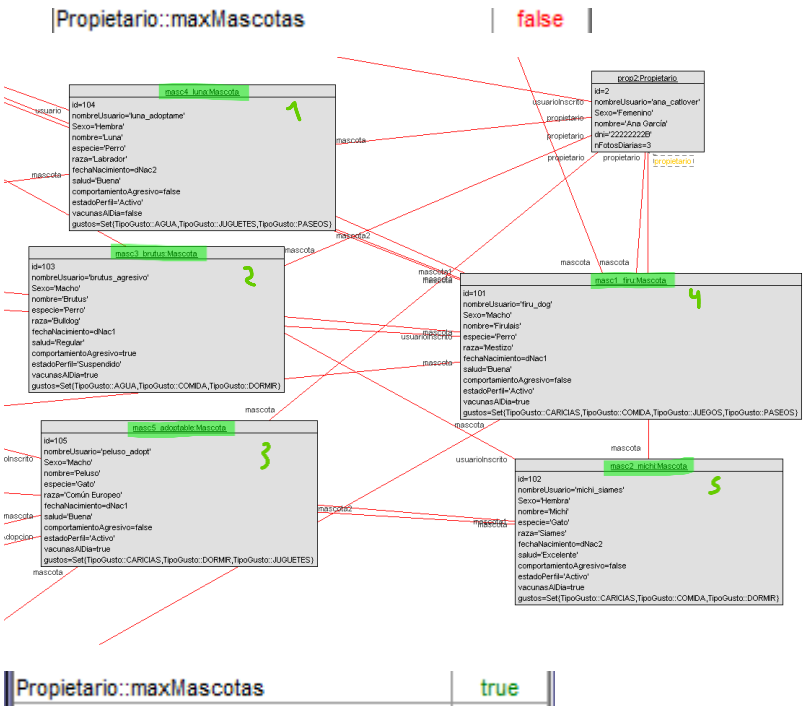
```
association ProductoAnunciado between
  AnuncioProducto [0..1] role anuncioProducto
  Producto [1] role producto
end
```

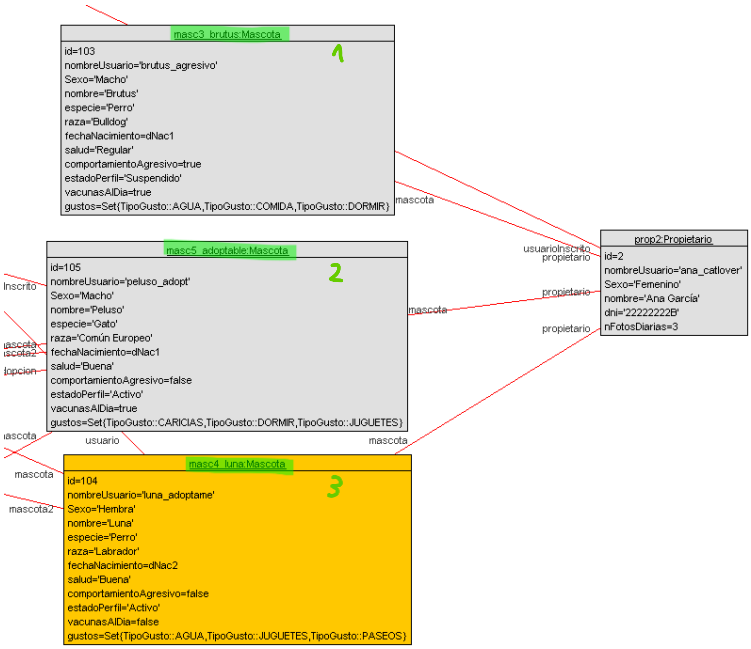
Restricciones

R1: Número máximo de macotas activas

```
context Propietario
inv maxMascotas: self.mascota→size() ⩽ 4
```

Un propietario puede tener registradas hasta 4 mascotas activas en la plataforma. Esto garantiza un control adecuado sobre la cantidad de animales gestionados por cada usuario.



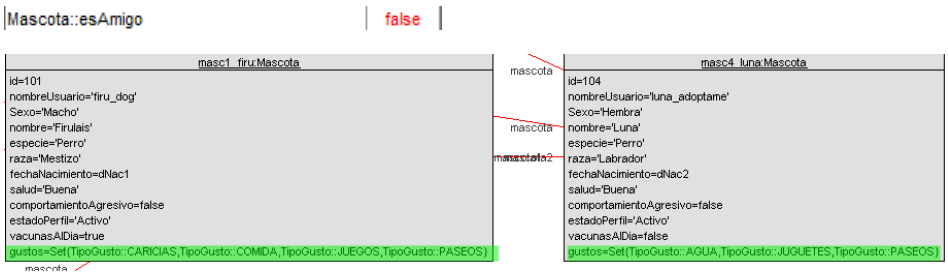


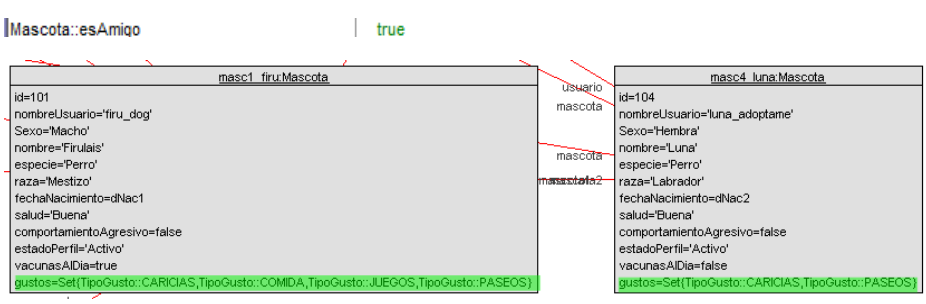
Comprobamos que funciona al primero instanciar 5 mascotas y después solo 3.

R2: Compatibilidades entre mascotas

```
context Mascota inv esAmigo:
self.mascota2→forAll(m: Mascota | m.oclIsUndefined() = false implies self.esCompatible(m))
```

Dos mascotas solo pueden establecer una relación de amistad si sus especies son compatibles y tienen mínimo 2 gustos similares. Esto evita interacciones no recomendadas entre ciertos tipos de animales.



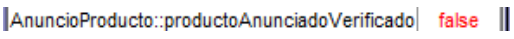


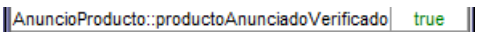
Se observa como salta la restricción al tener solo 1 gusto en común, después al coincidir en 2 gustos sale correcto.

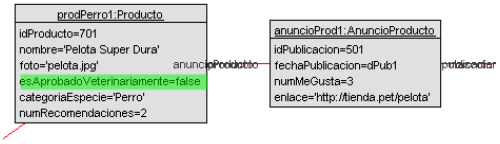
R3: Productos recomendados

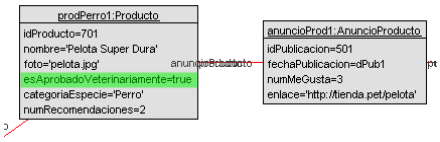
```
context AnuncioProducto
inv productoAnunciadoVerificado:
self.producto.esAprobadoVeterinariamente = true
```

Los productos recomendados en la plataforma deben contar con la aprobación de un veterinario y solo pueden ser sugeridos a mascotas de la especie correspondiente.







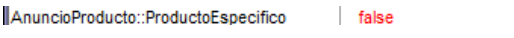


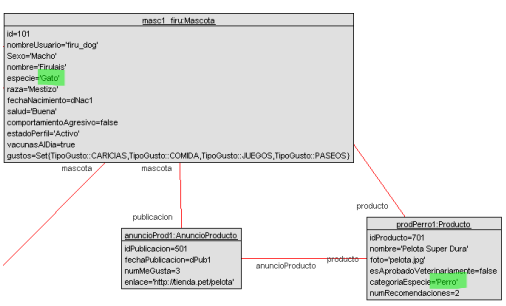
Se observa como falla cuando no está aprobado por un veterinario.

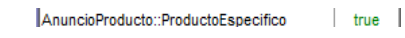
R4: Anuncios diferentes para cada especie

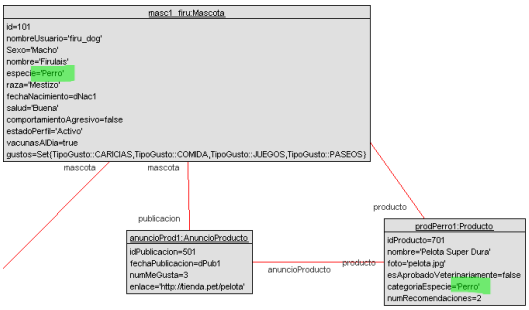
```
context AnuncioProducto
inv ProductoEspecifico:
self.producto.categoriaEspecie = self.mascota.especie
or self.producto.categoriaEspecie = 'todasLasEspecies'
```

Los anuncios de productos deben estar segmentados según la especie de la mascota. Un perro no debe recibir anuncios de comida para gatos, por ejemplo.







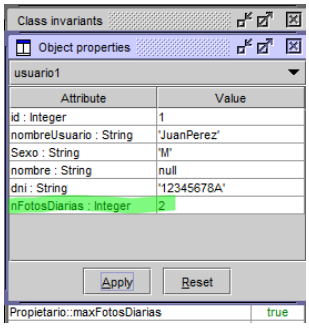


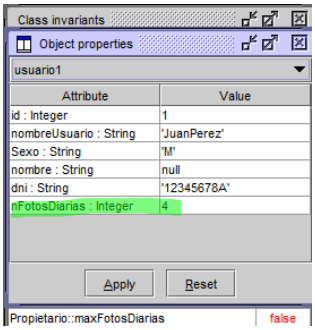
Se observa en primera instancia como salta la restricción cuando no coincide la especie de la mascota con la especie a la que va destinado el producto. Sin embargo, cuando coinciden está correcto.

R5: Número máximo de fotos diarias

```
context Propietario
inv maxFotosDiarias:
  self.nFotosDiarias ⩽ 3
```

Un propietario solo puede subir máximo 3 fotos al día. Esto ayuda a gestionar el almacenamiento y evita el spam de publicaciones.



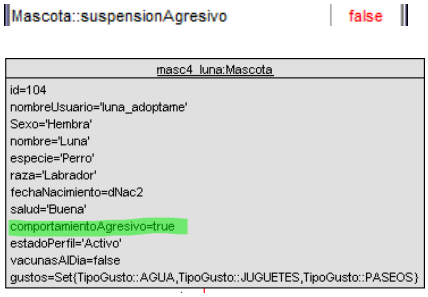


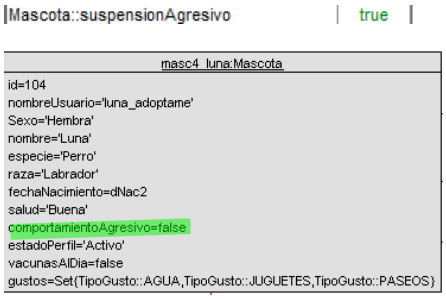
Se observa como la restricción salta cuando se supera el límite de las fotos diarias.

R6: Comportamiento agresivo

```
context Mascota
inv suspensionAgresivo:
  self.comportamientoAgresivo = true implies self.estadoPerfil = 'Suspendido'
```

Las mascotas con comportamiento agresivo no pueden interactuar con otras mascotas ni participar en eventos. Esto evita situaciones peligrosas y promueve un entorno seguro dentro de la plataforma.



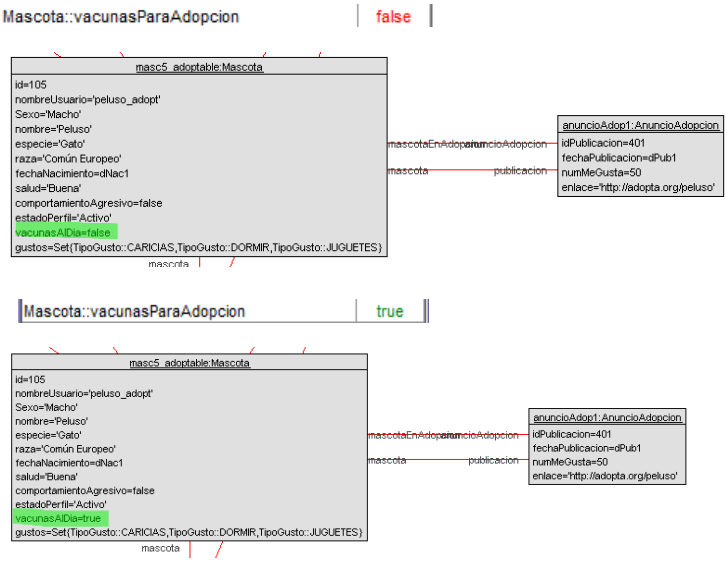


La restricción salta cuando una mascota tiene comportamiento agresivo.

R7: Vacunas para poder adoptar

```
context Mascota
inv vacunasParaAdopcion: self.anuncioAdopcion→notEmpty() implies self.vacunasAlDia=true
```

Para que una mascota pueda ser puesta en adopción, debe tener todas sus vacunas al día. Esto garantiza que los animales disponibles para adopción cumplan con requisitos de salud mínimos.

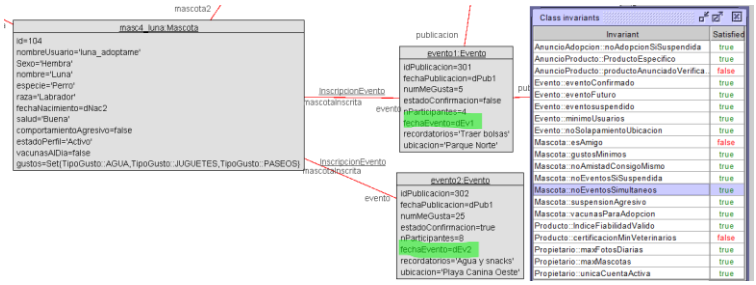


La restricción salta cuando una mascota no tiene las vacunas al día.

R8: Usuario puede apuntarse sólo a 1 evento en la misma fecha

```
context Mascota
inv noEventosSimultaneos: self.evento→forAll(e1, e2 | e1 <> e2 implies e1.fechaEvento <> e2.fechaEvento)
```

Un usuario no puede registrarse en más de un evento programado para el mismo día. Esto evita conflictos de agenda y mejora la organización de los eventos.

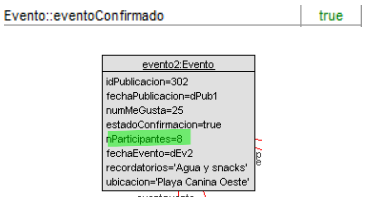


Cuando tenemos 2 eventos en la misma fecha la restricción salta.

R9: Para crear eventos hacen falta 6 propietarios

```
context Evento
inv eventoConfirmado:  estadoConfirmacion = true implies self.nParticipantes ⩾ 6
```

Un evento solo puede confirmarse si al menos seis propietarios están inscritos. Si faltan menos de siete días y no se alcanza este mínimo, el evento debe cancelarse automáticamente.

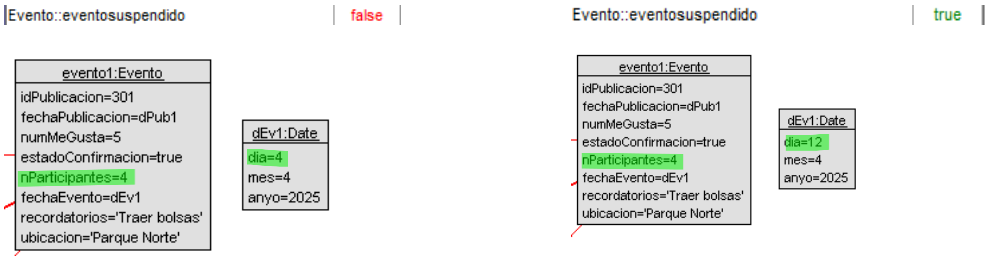


La restricción está presente hasta que se llega al número de participantes necesario.

R10: Si un evento no se confirma 7 días antes, se cancela

```
context Evento
inv eventosuspendido:
  -- Utilizamos directamente los valores de la fecha fija: 3/4/2025
  (self.fechaEvento.anyo * 365 + self.fechaEvento.mes * 30 + self.fechaEvento.dia)
  - (2025 * 365 + 4 * 30 + 3) < 7 -- Comparamos con una fecha fija que debe ser modificada aqui
  and self.nParticipantes ⩽ 6 implies estadoConfirmacion = false
```

Si un evento no reúne el número mínimo de participantes siete días antes de su realización, se cancela automáticamente. Esto previene la organización de eventos con baja participación.



Suponiendo que la fecha actual que tiene el sistema es 3/04/2025 cuando la fecha del evento es inferior a una semana y no tiene el número mínimo de participantes , la restricción salta, cuando queda más de una semana todavía la restricción no salta.

R11: Los usuarios no pueden tener más de 1 cuenta

```
context Propietario
inv unicaCuentaActiva:
  Propietario.allInstances()→forAll(u1, u2 | u1 <> u2 implies u1.dni <> u2.dni)
```

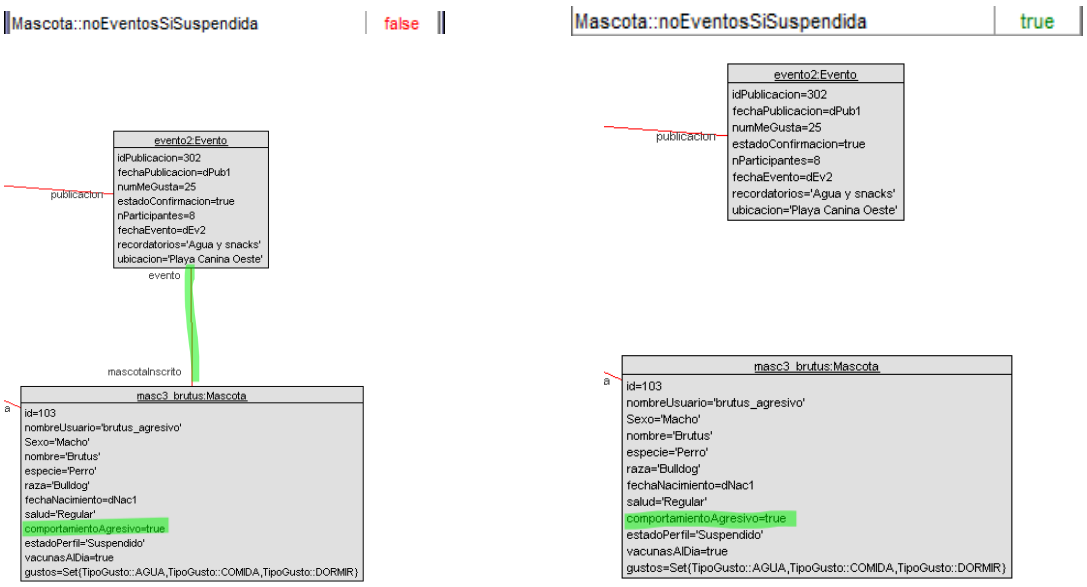
Cada usuario solo puede registrar una cuenta en la plataforma. Esta restricción evita cuentas duplicadas y posibles fraudes.

Cuando hay dos instancias de propietario con el mismo dni salta la restricción.

R12: Las mascotas suspendidas no se pueden inscribir en eventos

```
context Mascota
inv noEventosSiSuspendida:
self.estadoPerfil='Suspendido' implies self.evento→isEmpty()=true
```

Las mascotas que han sido marcadas como "perfil suspendido" no pueden inscribirse en eventos. Esta restricción ayuda a controlar el acceso de mascotas con problemas de comportamiento.

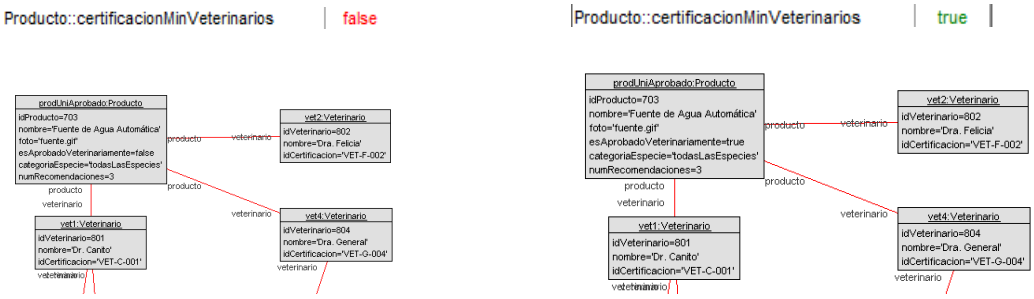


Se observa como salta la restricción cuando hay una relación de evento sobre una mascota suspendida.

R13: Para certificar un producto tienen que valorarlo 3 veterinarios

```
context Producto
inv certificacionMinVeterinarios:
((self.numRecomendaciones ⩾ 3) implies ((self.esAprobadoVeterinariamente = true) and (self.veterinario→size() ⩾ self.numRecomendaciones)))
```

Un producto solo puede recibir la certificación veterinaria si al menos tres veterinarios lo han valorado. Esto garantiza un mayor nivel de confianza en los productos recomendados.



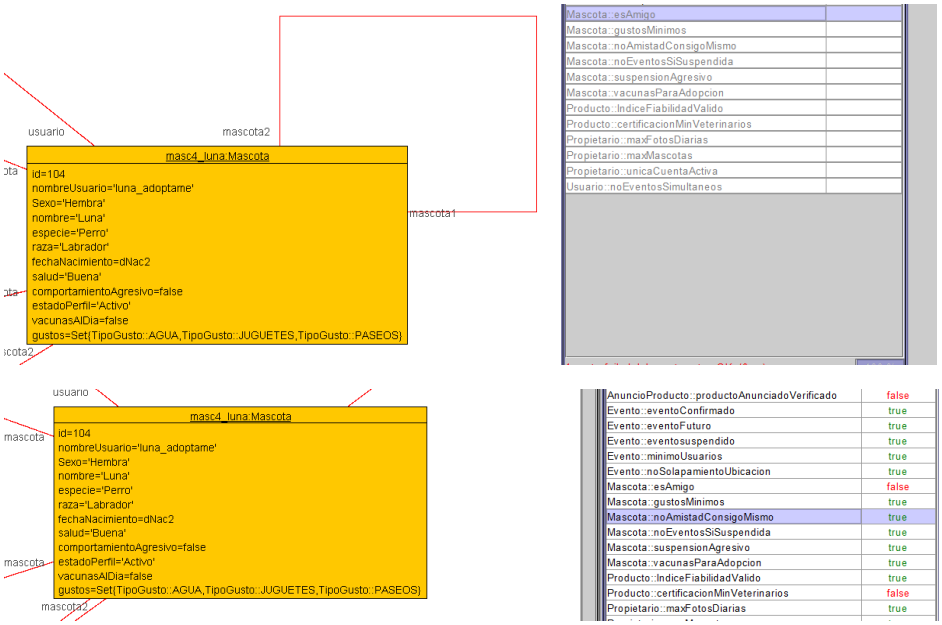
Se observa como cuando el atributo “esAprobadoVeterinariamente” está en false, salta la restricción.

R14: Una mascota no puede ser amiga de sí misma

```
context Mascota
inv noAmistadConsigoMismo: not self.mascota1→includes(self)
```

Una mascota no puede agregarse a sí misma como amiga. Esta restricción evita relaciones inválidas dentro del sistema.

```
use> ! insert(masc4_luna, masc4_luna) into Amigos
use>
[Warning] 1 precondition in operation call `Mascota::esCompatible(self:masc4_luna, otraMascota:masc4_luna)' does not hol
d:
  pre4: (otraMascota.id <> self.id)
```

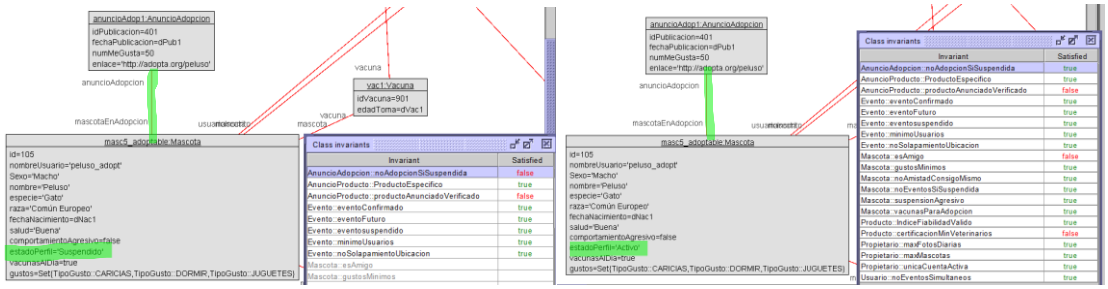


La precondición “esCompatible” verifica que una misma mascota no sea amigo de sí mismo.

R15: Las mascotas suspendidas no pueden ser adoptadas

```
context AnuncioAdopcion
inv noAdopcionSiSuspendida:
mascotaEnAdopcion.estadoPerfil <> 'Suspendido'
```

Si una mascota ha sido suspendida, no puede ser puesta en adopción. Esto garantiza que solo mascotas en condiciones adecuadas puedan ser adoptadas.



Se observa como salta la restricción cuando hay instanciado un anuncio de adopción sobre una mascota suspendida.

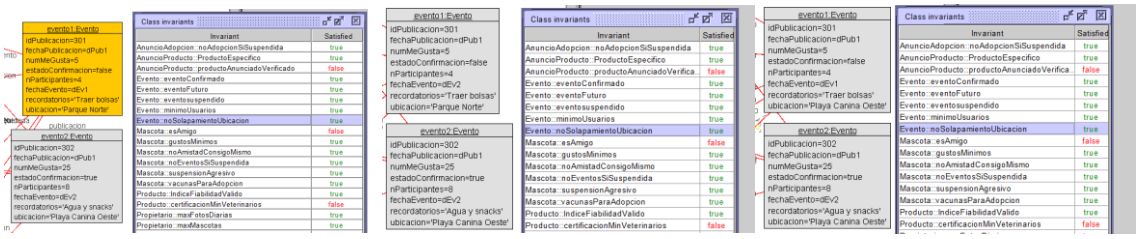
R16: No puede haber 2 eventos en la misma ubicación

```
context Evento
inv noSolapamientoUbicacion:
Evento.allInstances()→forAll(e1, e2 | e1 <> e2 implies e1.ubicacion <> e2.ubicacion or e1.fechaEvento <> e2.fechaEvento)
```

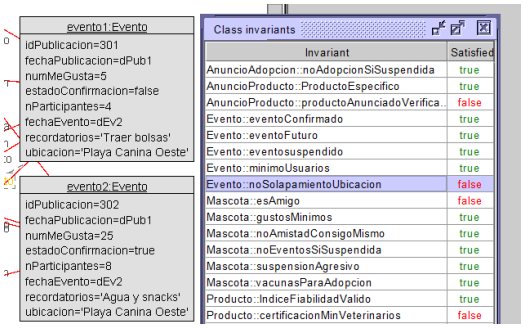
No pueden existir dos eventos en la misma ubicación y el mismo día. Esto evita conflictos de agenda y problemas logísticos.

R16: TRUE fecha distinta | ubicación distinta - R16: TRUE fecha igual | ubicación distinta

- R16: TRUE fecha distinta | ubicación igual



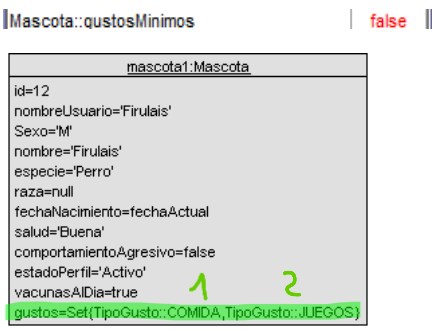
Comprobamos todas las opciones en 2 eventos instanciados, con fecha distinta con ubicación distinta, misma fecha con ubicación distinta, y fecha distinta con misma ubicación. En ninguno de los casos salta la restricción , esta salta cuando tanto las ubicaciones como las fechas son iguales en ambos eventos:

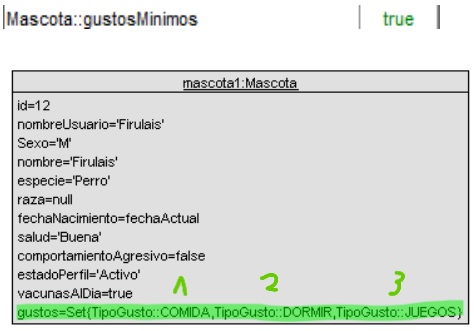


R17: Cada mascota debe tener mínimo 3 gustos

```
context Mascota
inv gustosMinimos:
self.gustos → size ⩾ 3
```

Cada mascota debe registrar al menos tres gustos. Esto permite mejorar las sugerencias de amistad y la personalización del contenido en la plataforma.



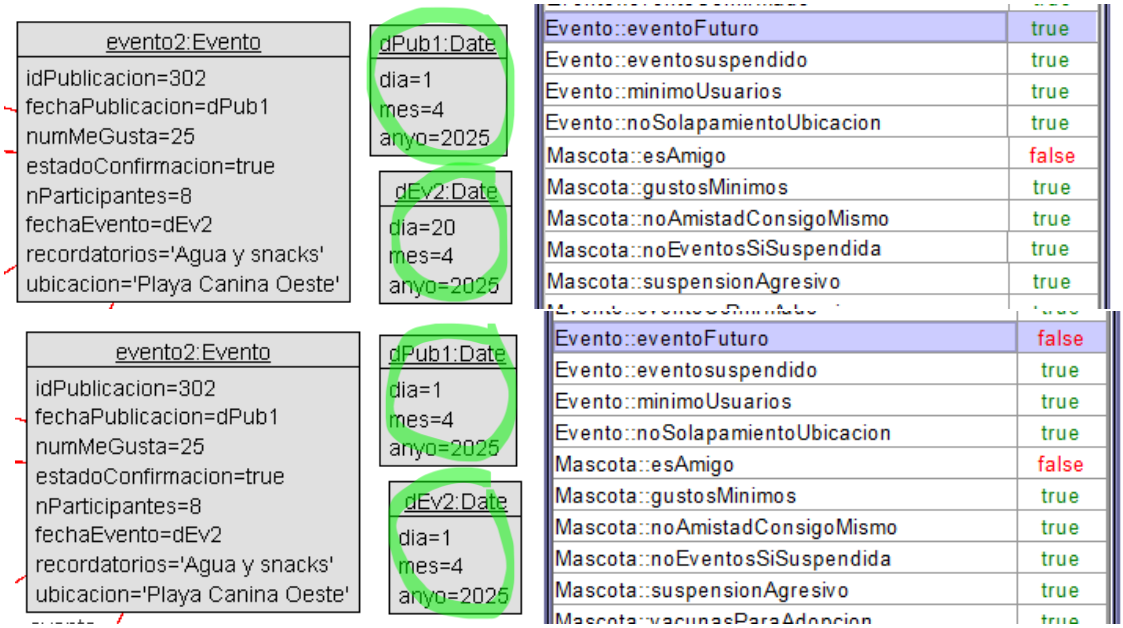


Comprobamos que salta la restricción cuando seleccionamos solo 2 gustos.

R18: Los eventos deben ser creados con una semana de antelación

```
context Evento
inv eventoFuturo:
self.fechaEvento.anyo > self.fechaPublicacion.anyo or
(self.fechaEvento.anyo = self.fechaPublicacion.anyo and self.fechaEvento.mes > self.fechaPublicacion.mes) or
(self.fechaEvento.anyo = self.fechaPublicacion.anyo and self.fechaEvento.mes = self.fechaPublicacion.mes and self.fechaEvento.dia ≥ (self.fechaPublicacion.dia + 7))
```

Los eventos deben ser creados con un mínimo de una semana de anticipación. Esto permite una mejor organización y da tiempo suficiente a los usuarios para inscribirse.



Se observa como salta la restricción cuando la fecha de publicación del evento no tiene la suficiente antelación.

Contratos

esCompatible(otraMascota:Mascota)

Este método determina si una mascota es compatible con otra, según varios criterios.

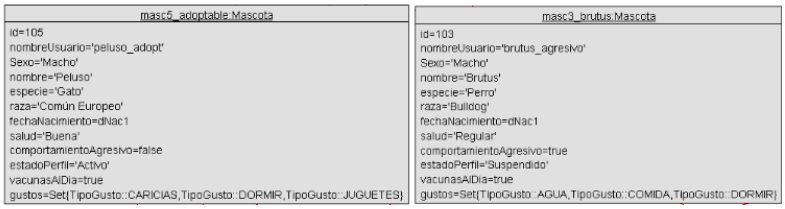
(En este método al tener asignada una expresión ocl, esta operación no puede ser evaluada con !openter para precondiciones y !opexit para las postcondiciones. En su lugar usamos '?' llamando a la operación a través de una instancia concreta (al igual que openter) y vemos el resultado que devuelve y los warnings indicando las precondiciones que no se cumplen.)

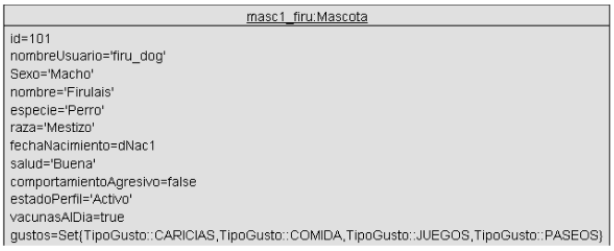
```
context Mascota::esCompatible(otraMascota:Mascota) : Boolean
  -- No calcular la compatibilidad consigo misma
  pre noElMismo: otraMascota.id <> self.id
  -- No calcular compatibilad con especies distintas
  pre otraEspecie: otraMascota.especie = self.especie
  -- Ninguna puede estar suspendida
  pre noSuspendidos: otraMascota.estadoPerfil <> 'Suspendido' and self.estadoPerfil <> 'Suspendido'
```

Precondiciones:

1) No se puede comparar una mascota consigo misma.
2) Ambas mascotas deben pertenecer a la misma especie.
3) Ninguna de las dos debe tener el perfil suspendido.

Contexto de instancias:






```
use> ?masc1_firu.esCompatible(masc1_firu)

[Warning] 1 precondition in operation call `Mascota::esCompatible(self:masc1_firu, otraMascota:masc1_firu)' does not hold:
  pre4: (otraMascota.id <> self.id)
```

Fallo PRE-1

```
use> ?masc1_firu.esCompatible(masc5_adoptable)

[Warning] 1 precondition in operation call `Mascota::esCompatible(self:masc1_firu, otraMascota:masc5_adoptable)' does not hold:
  pre5: (otraMascota.especie = self.especie)
```

Fallo PRE-2

```
[Warning] 1 precondition in operation call `Mascota::esCompatible(self:masc1_firu, otraMascota:masc3_brutus)' does not hold:
  pre6: ((otraMascota.estadoPerfil <> 'Suspendido') and (self.estadoPerfil <> 'Suspendido'))
```

Fallo PRE-3

```
use> ?masc1_firu.esCompatible(masc3_brutus)
-> false : Boolean
use>
```

Todo correcto

publicarComentario(comentario:Comentario)

Este método permite a una mascota publicar un comentario en una publicación, siempre que se cumplan ciertas condiciones.

```
context Publicacion::publicarComentario(comentario: Comentario): Boolean
  -- que la fecha del comentario no sea anterior a la fecha de la publicacion
  pre fechaValida: self.fechaPublicacion.anyo ≤ comentario.fechaComentario.anyo
  pre mascotaExistente: Mascota.allInstances()→exists(m : Mascota | m.id = comentario.idUsuario)
  pre comentarioNoVacio: comentario.texto.size() > 0
  post comprobarComentario: self.comentario→includes(comentario)
```

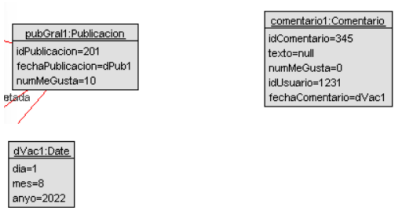
Precondiciones:

- La fecha del comentario no puede ser anterior al año de publicación.
- Debe existir una mascota cuyo ID coincida con el del comentario (autor válido).
- El comentario debe tener texto (no estar vacío).

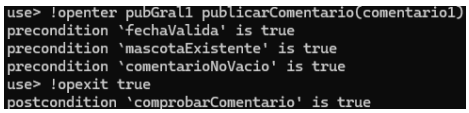
Postcondiciones:

- El comentario queda correctamente vinculado a la publicación.

Contexto fallo pre y post:



Contexto todo correcto:



inscribirseEvento(eventoNuevo: Evento)

Permite que una mascota se inscriba a un evento si aún no está inscrita y su perfil está activo.

```
context Mascota::inscribirseEvento(eventoNuevo: Evento) : Boolean
  pre: self.evento→includes(eventoNuevo) = false
  pre: self.estadoPerfil <> 'Suspendido'
  post: self.evento→includes(eventoNuevo) = true
```

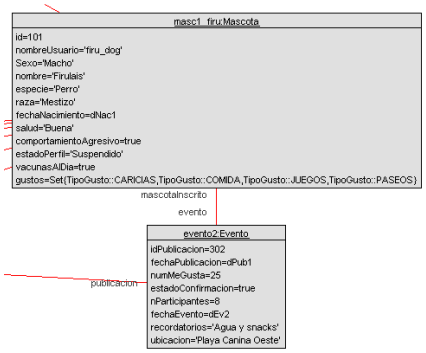
Precondiciones:

- La mascota no debe estar ya inscrita en el evento.
- Su perfil debe estar activo (no suspendido).

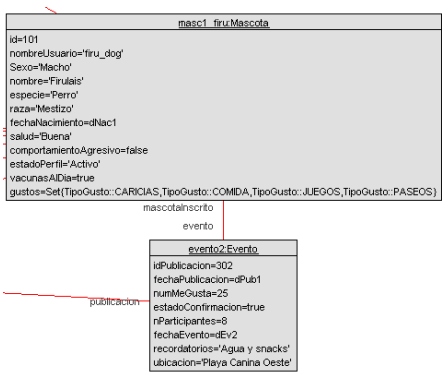
Postcondiciones:

- Tras ejecutarse, el evento aparece en la lista de eventos a los que está inscrita la mascota.

Contexto fallos:



Contexto todo correcto:




```
use> !openter masc1_firu inscribirseEvento(evento2)
precondition `pre4' is true
precondition `pre5' is true
use> !opexit true
postcondition `post1' is false
  self : Mascota = masc1_firu
  self.evento : Set(Evento) = Set{}
  eventoNuevo : Evento = evento2
  self.evento->includes(eventoNuevo) : Boolean = false
  true : Boolean = true
  (self.evento->includes(eventoNuevo) = true) : Boolean = false
Error: postcondition false in operation call `Mascota::inscribirseEvento(self:masc1_firu, eventoNuevo:evento2)'.
```

Falla Post

```
use> !openter masc1_firu inscribirseEvento(evento2)
precondition `pre4' is true
precondition `pre5' is true
use> !opexit true
postcondition `post1' is false
  self : Mascota = masc1_firu
  self.evento : Set(Evento) = Set{}
  eventoNuevo : Evento = evento2
  self.evento->includes(eventoNuevo) : Boolean = false
  true : Boolean = true
  (self.evento->includes(eventoNuevo) = true) : Boolean = false
Error: postcondition false in operation call `Mascota::inscribirseEvento(self:masc1_firu, eventoNuevo:evento2)'.
```

Fallan las Pre

```
use> !openter masc1_firu inscribirseEvento(evento2)
precondition `pre4' is true
precondition `pre5' is true
use> !insert(evento2,masc1_firu) into InscripcionEvento
use> !opexit true
postcondition `post1' is true
```

Todo correcto

certificarProducto(producto: Producto)

Este método permite a un veterinario certificar un producto. Al hacerlo, se incrementa el número de recomendaciones del producto.

```
context Veterinario::certificarProducto(producto: Producto)
  pre compruebaExistencias: self.producto→exists(p : Producto | p.idProducto = producto.idProducto) = true

  post post1: producto.numRecomendaciones = producto.numRecomendaciones@pre + 1
```

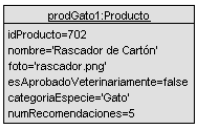
Precondiciones:

- El producto a certificar debe existir en el conjunto de productos ya registrados.

Postcondiciones:

- El número de recomendaciones del producto aumenta en 1 respecto a su valor anterior.

Contexto fallos:



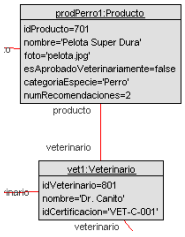

```
use> !openter vet1 certificarProducto(prodGato1)
precondition `compruebaExistencias' is false
Error: precondition false in operation call `Veterinario::certificarProducto(self:vet1, producto:prodGato1)'.
```

Fallo pre

```
use> !prodPerro1.numRecomendaciones:=2
use> !openter vet1 certificarProducto(prodPerro1)
precondition `compruebaExistencias' is true
use> !opexit
postcondition `post1' is false
  producto : Producto = prodPerro1
  producto.numRecomendaciones : Integer = 2
  producto : Producto = prodPerro1
  producto.numRecomendaciones@pre : Integer = 2
  1 : Integer = 1
  (producto.numRecomendaciones@pre + 1) : Integer = 3
  (producto.numRecomendaciones = (producto.numRecomendaciones@pre + 1)) : Boolean = false
Error: postcondition false in operation call `Veterinario::certificarProducto(self:vet1, producto:prodPerro1)'.
```

Fallo post

Contexto todo correcto:




```
use> !openter vet1 certificarProducto(prodPerro1)
precondition `compruebaExistencias' is true
```

Pre correcto

```
use> !openter vet1 certificarProducto(prodPerro1)
precondition `compruebaExistencias' is true
use> !prodPerro1.numRecomendaciones:=3
use> !opexit
postcondition `post1' is true
use>
```

Post correcto

Conclusiones

Realizando el proyecto nos han surgido ciertos problemas al tratar con fechas, en concreto, con la fecha actual. El principal problema es que OCL no te permite obtener la fecha actual del sistema y así poder comparar con las fechas de nuestros modelos de prueba. Hemos probado varias soluciones, llegando a una única solución: poner una fecha fija dentro del sistema.

En cuanto al proyecto, por mucho que hemos intentado abordar, una red social es algo bastante complejo y de una dimensión considerable. Se podría decir que hemos hecho un sistema preliminar al que aún faltan pulir muchos aspectos. Si fuera un proyecto con un plazo mayor (o un equipo más amplio), se podría llegar a conseguir un resultado bastante decente.