MEMORIA PRÁCTICA 2
GRUPO GR3-2

APARTADO A

RESTRICCIONES

Distancia entre ciudades
Cada ciudad debe tener una distancia de al menos 5 Km para que se considere un viaje (invariante USE).

Relaciones entre ciudades
En cada viaje habrá 2 ciudades asociadas, una de origen y una de destino (controlado con multiplicidades del modelo).

Fecha revisión y matriculación
La fecha de la revisión del coche debe ser posterior a la fecha de matriculación del coche (invariante USE).

Revisiones coche
Un coche no podrá pasar 2 revisiones a la vez. (invariante USE).

Talleres en ciudades
Como mucho habrá un taller oficial en una ciudad, pero podrá haber varios no oficiales (multiplicidades del modelo)

Revisión en ciudad
Si un coche está siendo sometido a revisión en un taller, el coche se debe encontrar en la ciudad de ese mismo taller (invariante USE).

Kilómetros de coches
Los kilómetros de cada coche deberán ser la suma de los recorridos de los viajes realizados (controlado con el atributo *derive*).

Ciudad origen debe ser la anterior ciudad destino
Si un coche se encuentra en una ciudad, su próximo viaje debe empezar desde esa ciudad y no desde otra (invariante USE).

Ciudad origen distinta a ciudad destino
La ciudad de origen debe ser distinta a la ciudad destino (invariante USE)

Fechas de Viajes
2 viajes no pueden solaparse en el tiempo, la fecha de inicio de un viaje no puede estar entre la fecha de inicio y final de otro (invariante USE).

Garantía coche
Un coche está en garantía si han pasado menos de 4 años desde que se matriculó o si han pasado menos días que los indicados en la garantía del taller oficial donde el coche tuvo alguna revisión (invariante USE).

Frecuencia revisiones de mantenimiento
Un coche no pasará revisiones hasta pasados 4 años desde su matriculación. A partir de ahí, no necesitará mantenimiento hasta pasado 1 año después de su respectiva revisión de mantenimiento (controlado con el atributo *derive*).

Coche viajando o en ciudad
Un coche se encontrará en todo momento bien realizando un viaje determinado o bien en una ciudad (invariante USE).

Coche en ciudad de llegada del último viaje
Si el coche ha completado al menos un viaje y no se encuentra viajando, entonces debe encontrarse en la ciudad a la que llegó en su último viaje (invariante USE).

DIAGRAMA DE CLASES (USE)

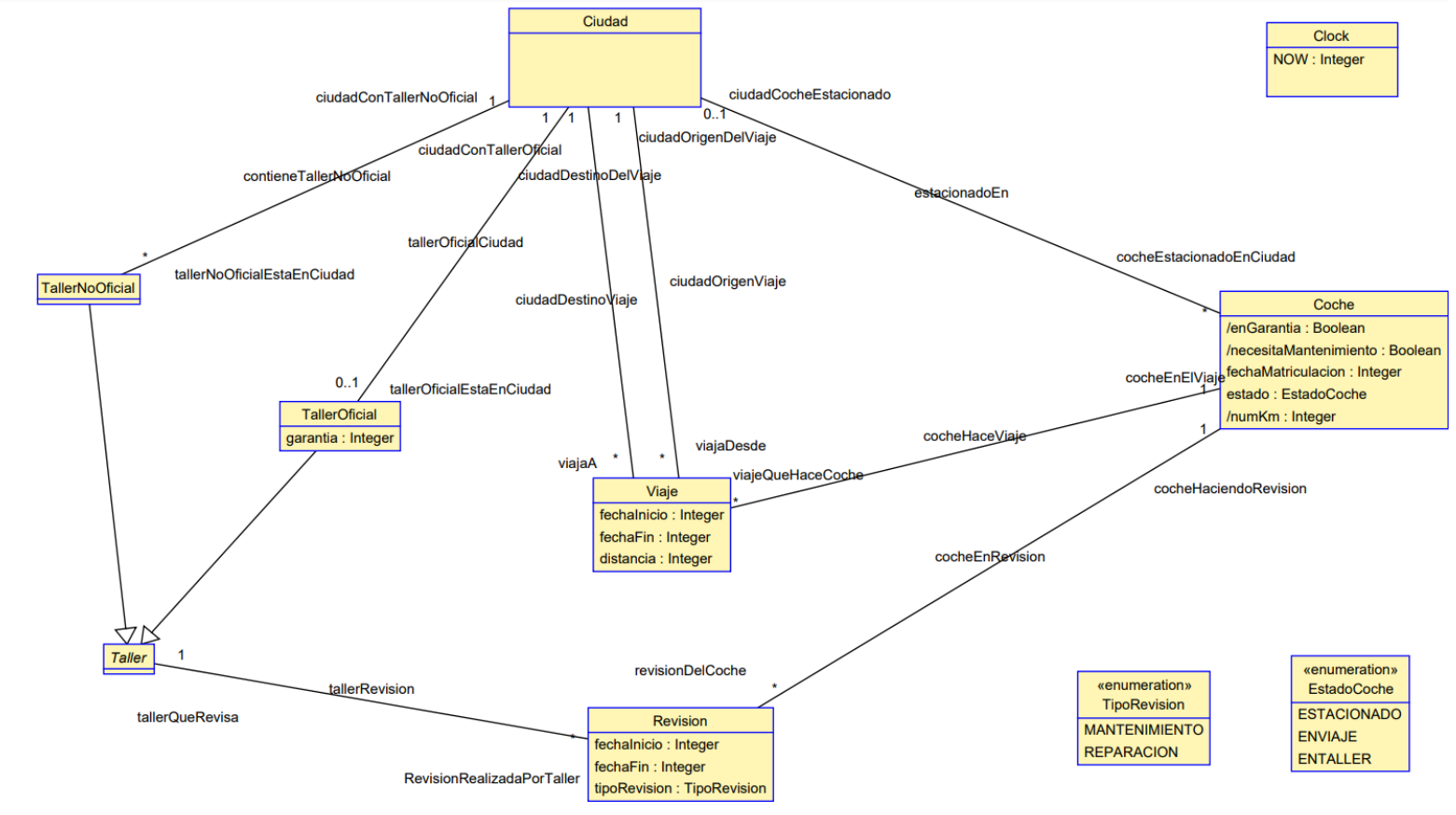
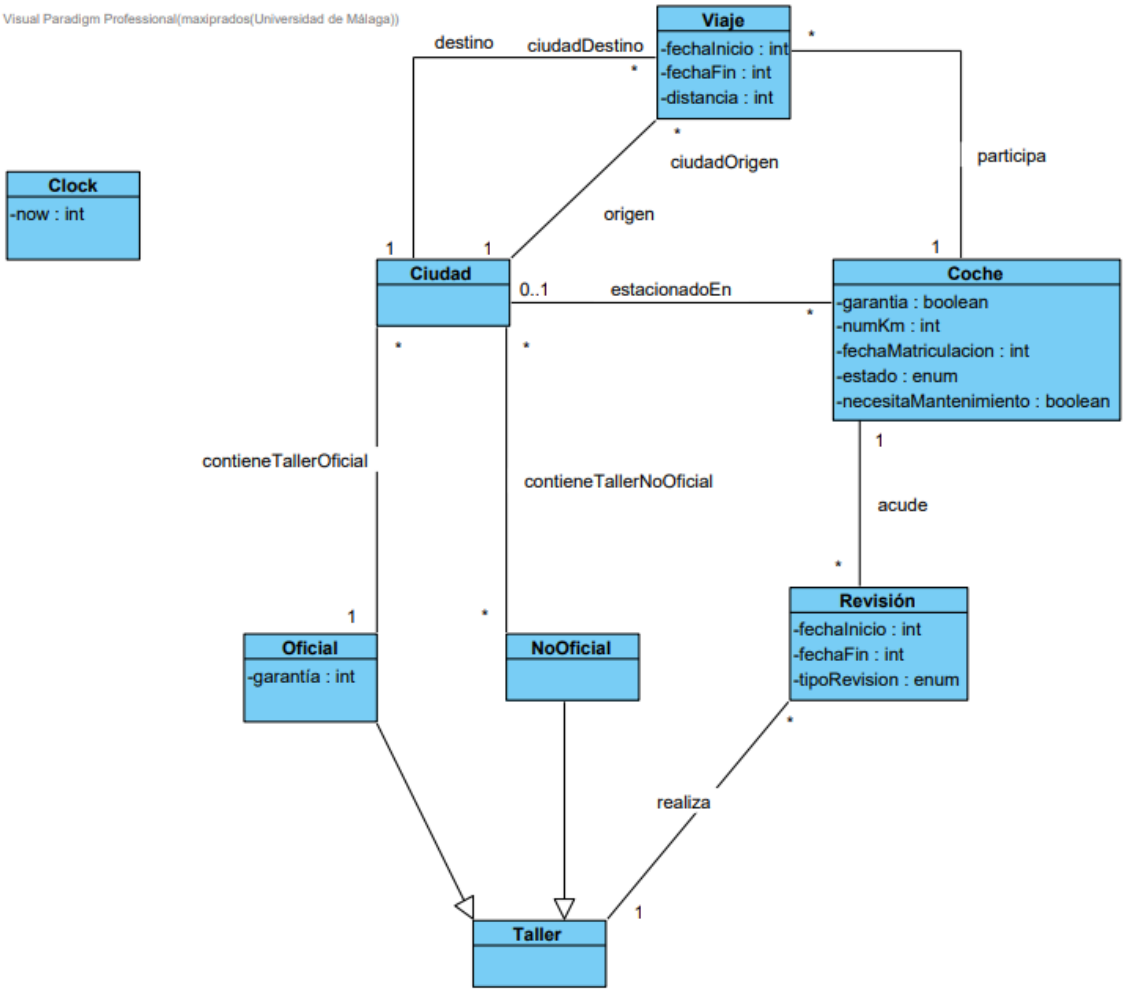


DIAGRAMA DE CLASES (VISUAL PARADIGM)



CLASES Y ROLES

COCHE

Será la clase que modele los coches en nuestro sistema, que realizarán los diversos viajes entre 2 ciudades.

- garantía: Este atributo controlará si el coche está en garantía o no.
- numKM: (derive) Nos indicará el nº de kms que tiene el coche, que se irá actualizando a medida que se realicen los viajes.
- fechaMatriculacion: Número entero que nos dice a partir de qué dia empieza a poder funcionar el coche (tenemos una concepción del tiempo lineal, considerando que cada 100 días se completa un año).

- estado: Con este tipo enumerado controlaremos si el coche se encuentra realizando un viaje, estacionado en una ciudad o pasando una revisión en algún taller.
- necesitaMantenimiento: booleano *derive* que controlará si el coche necesita mantenimiento o no.

VIAJE

Esta clase va a modelar las condiciones del desplazamiento de un coche entre 2 ciudades, una de ellas será la ciudad desde la que se parte (relación orien; rol ciudadOrigen) y la otra será la ciudad a la que se quiere llegar (relación destino; rol ciudadDestino).

- fechaInicio: Atributo que indica el día en el que empieza el viaje.
- fechaFin: Entero que indica el día en el que se finalizó el viaje (si el coche está aún realizando este viaje, la fechaFin estará a null). Una vez que el coche esté estacionado en una ciudad (estado = 'estacionado'), la fechaFin de su último viaje no podrá estar a null.
- distancia: Entero que representa los kilómetros que separan las 2 ciudades asociadas.

CIUDAD

Clase que representa un lugar donde podremos encontrar talleres, donde podrán estar estacionados los coches o donde podrán partir en sus viajes (o a donde se podrán dirigir).

REVISIÓN

Clase que detalla los atributos relacionados con las distintas visitas de los coches a los talleres.

- fechaInicio: Entero que nos indica el día en el que se comienza a realizar la revisión.
- fechaFin: Entero que representa cuando acabará la revisión (debe ser posterior a fechaInicio).

Estas fechas las emplearemos para controlar que las revisiones no se solapan para un mismo coche (un coche no puede estar en 2 revisiones al mismo tiempo).

- tipoRevision: Este tipo enumerado nos servirá para saber si la revisión que se realiza es o bien un mantenimiento o bien una reparación.

TALLER

Esta clase modelará el lugar donde se van a llevar acabo las revisiones (tanto reparaciones como mantenimientos). Además, estos talleres se deben encontrar en las ciudades y podremos tener talleres oficiales o no oficiales (detallados a continuación).

OFICIAL

Esta clase, que hereda de la clase Taller, modelará los talleres oficiales, de los que solo puede haber uno por cada ciudad.

- garantia: Entero que nos indica el estado de la garantía del coche que pasa la revisión.

NOOFICIAL

Clase que modela los talleres que no son oficiales y de los que podemos encontrar varios en una misma ciudad (estos talleres no tienen restricciones de multiplicidad).

CLOCK

Clase que vamos a emplear para modelar el paso del tiempo y los cambios del sistema.

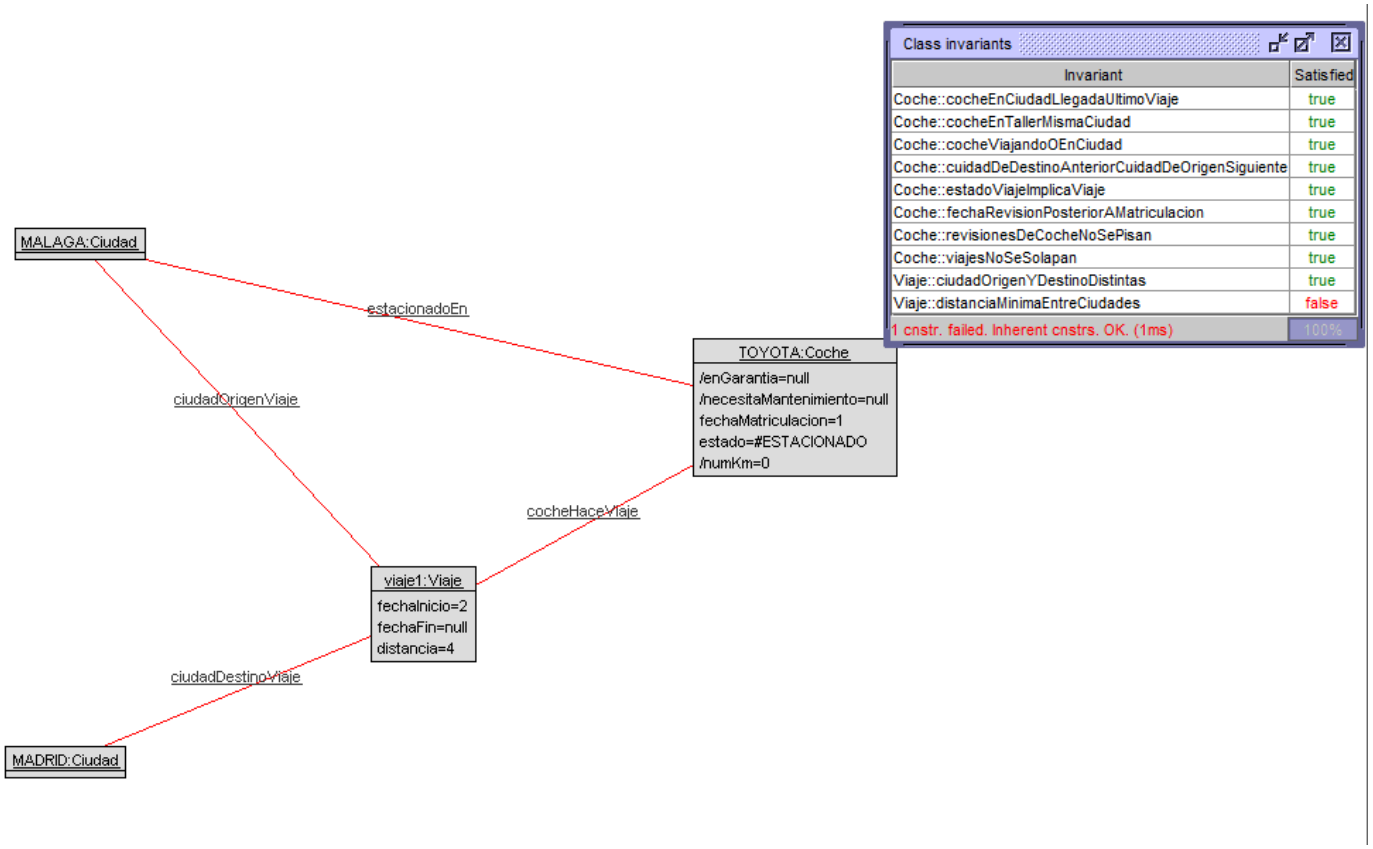
- now: Entero que representará el día en el que nos encontramos dentro de nuestro sistema.

DIAGRAMAS DE OBJETOS (.SOIL)

En este apartado, mostraremos diferentes ejemplos de diagramas de objetos desarrollados en archivos .soil para demostrar que, efectivamente, se cumplen las invariantes controladas en nuestros archivo .use.

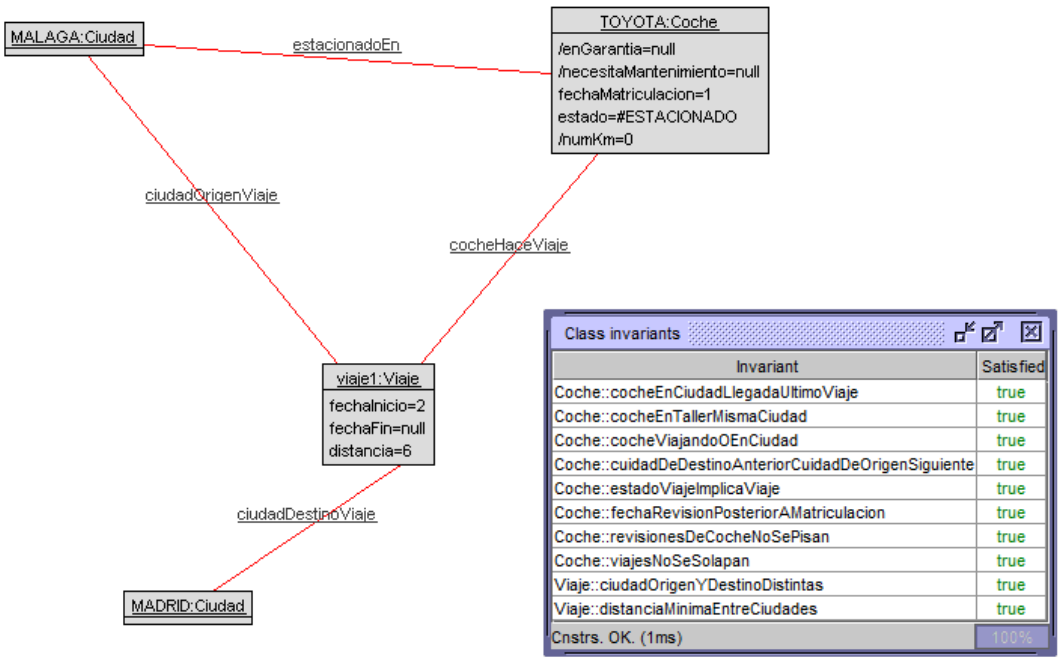
distanciaMinimaEntreCiudades

Comprobamos que la distancia mínima entre 2 ciudades debe ser al menos de 5 Km



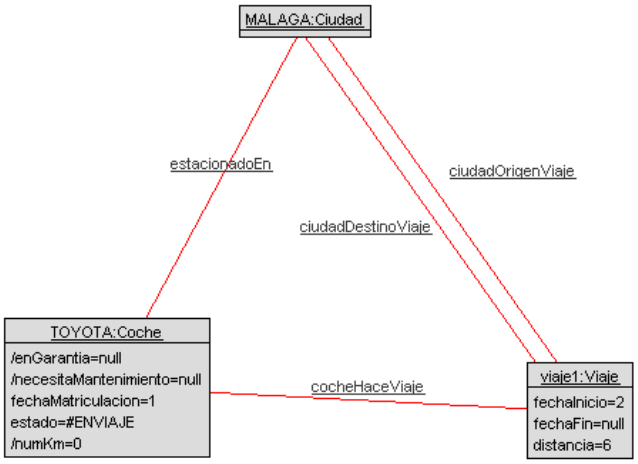
Efectivamente si colocamos una distancia inferior a 5 Km, se incumple la restricción.

Sin embargo, si la distancia es mayor de 5 Km, sí se cumple la restricción:



ciudadOrigenYDestinoDistintas

Se comprobará que el coche parte de una ciudad pero no puede tener como destino esa misma ciudad:

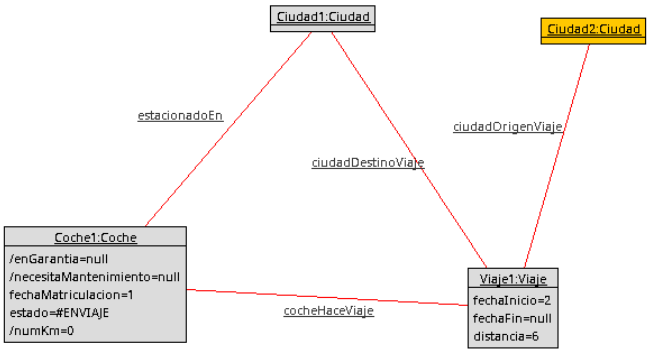


Class invariants

Invariant	Satisfied
Coche::cocheEnCiudadLlegadaUltimoViaje	true
Coche::cocheEnTallerMismaCiudad	true
Coche::cocheViajandoOEnCiudad	true
Coche::cuidadDeDestinoAnteriorCuidadDeOrigenSiguiente	true
Coche::estadoViajeImplicaViaje	true
Coche::fechaRevisionPosteriorAMatriculacion	true
Coche::revisionesDeCocheNoSePisan	true
Coche::viajesNoSeSolapan	true
Viaje::ciudadOrigenYDestinoDistintas	false
Viaje::distanciaMinimaEntreCiudades	true

1 cnstr. failed. Inherent cnstrs. OK. (5ms) 100%

Al asociar al objeto ‘viaje1’ la misma ciudad origen y ciudad destino del viaje (‘MALAGA’), se viola la restricción, luego la invariante funciona.



Class invariants

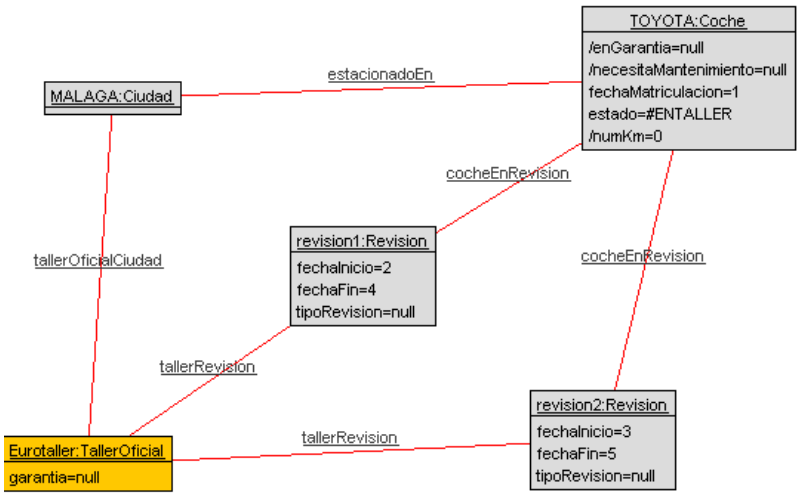
Invariant	Satisfied
Coche::cocheEnCiudadLlegadaUltimoViaje	true
Coche::cocheEnTallerMismaCiudad	true
Coche::cocheViajandoOEnCiudad	true
Coche::cuidadDeDestinoAnteriorCuidadDeOrigenSiguiente	true
Coche::estadoViajeImplicaViaje	true
Coche::fechaRevisionPosteriorAMatriculacion	true
Coche::revisionesDeCocheNoSePisan	true
Coche::viajesNoSeSolapan	true
Viaje::ciudadOrigenYDestinoDistintas	true
Viaje::distanciaMinimaEntreCiudades	true

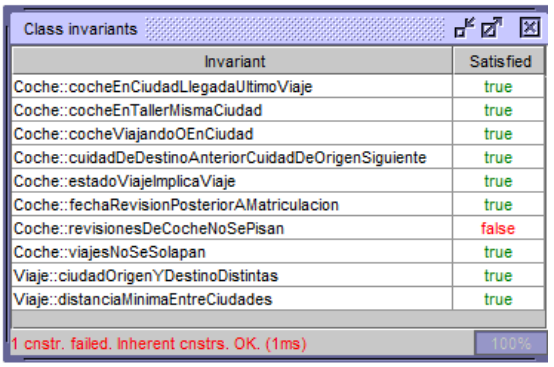
Cnstrs. OK. (2ms) 100 %

Sin embargo, si asociamos dos ciudades distintas, se cumple la invariante correctamente.

revisionesDeCocheNoSePisan

Esta invariante controla que solo exista una revisión para un coche en un mismo tiempo y no puedan coexistir 2 revisiones (independientemente del tipo que sea) a la vez.

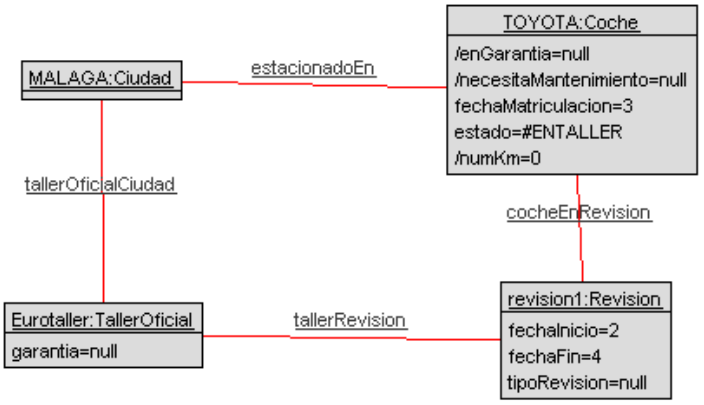


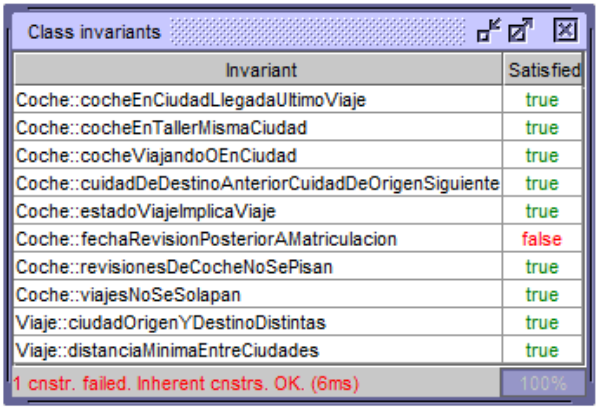


Como se observa en el diagrama, ‘revision1’ comienza el día 2 pero no acaba hasta el día 4. ‘revision2’ comienza el día 3, por lo que estarían realizándose 2 revisiones al mismo tiempo. Efectivamente, vemos como se detecta que no se cumple la invariante.

fechaRevisionPosteriorAMatriculacion

Debemos comprobar que un coche no puede someterse a una revisión hasta que no se haya matriculado. En la captura anterior comprobamos que, efectivamente, el coche se había matriculado el día 1 y la revisión no se realizaba hasta el día 2 por lo que no se violaba ninguna restricción. Pero, ¿y si realizamos la revisión antes de que el coche se matricule?

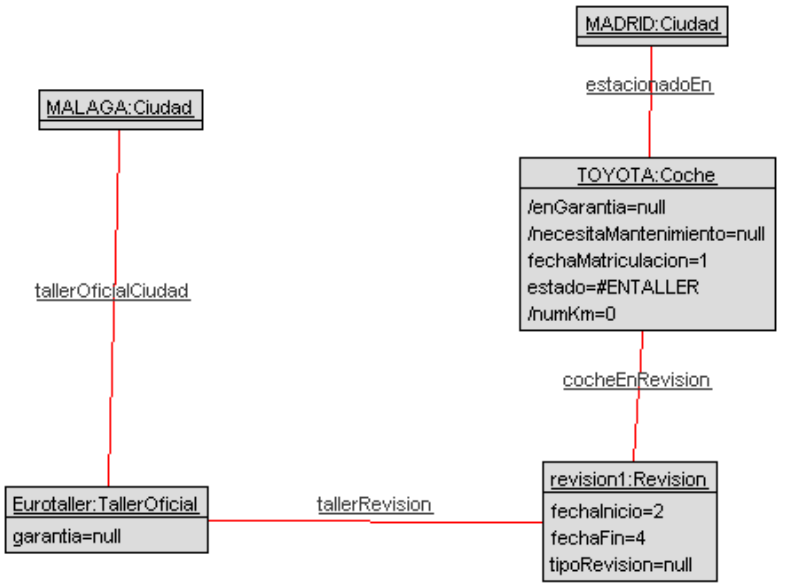


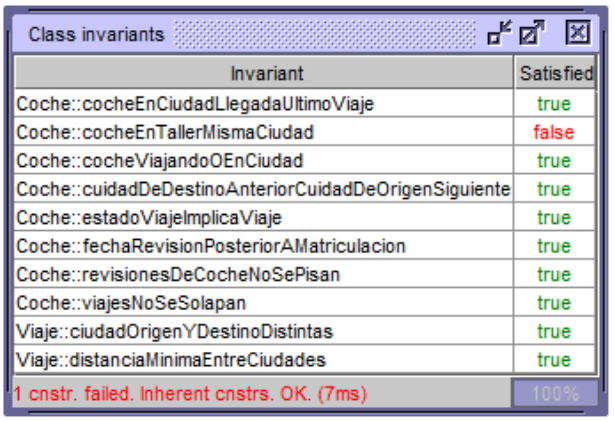


Al asociar una revisión que comienza antes de la fecha de matriculación de su coche asociado comprobamos que se viola la restricción, luego se cumple con la restricción diseñada.

CocheEnTallerMismaCiudad

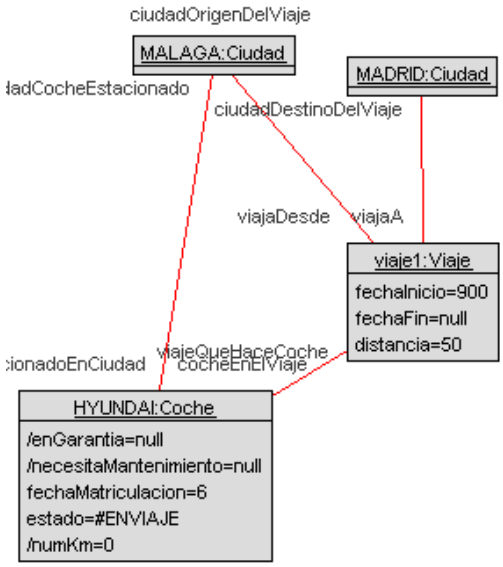
No podemos tener un coche ‘estacionadoEn’ una ciudad distinta en la que está siendo revisado:

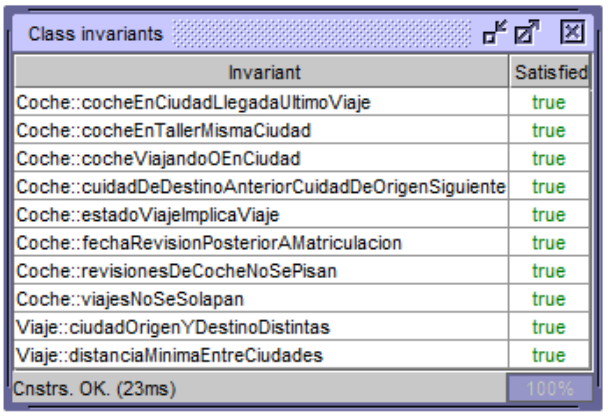




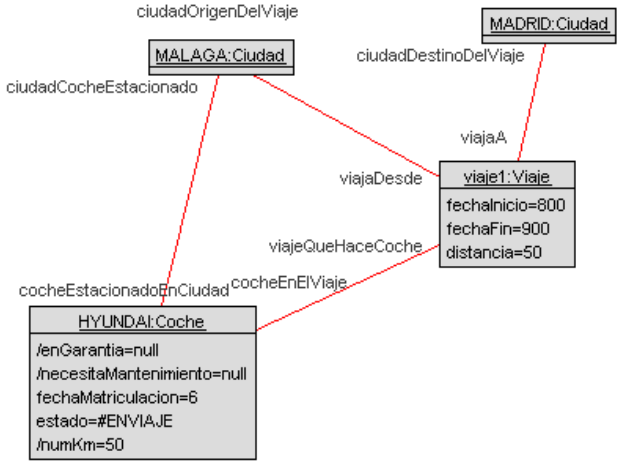
En este diagrama de objetos tenemos al coche ‘estacionadoEn’ la ciudad ‘MADRID’ pero está pasando la ‘revision1’ en un Taller Oficial ubicado en la ciudad ‘MALAGA’, por lo que efectivamente comprobamos que se viola la restricción.

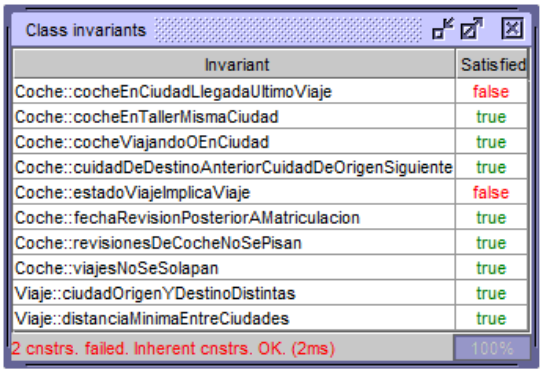
estadoViajeImplicaViaje



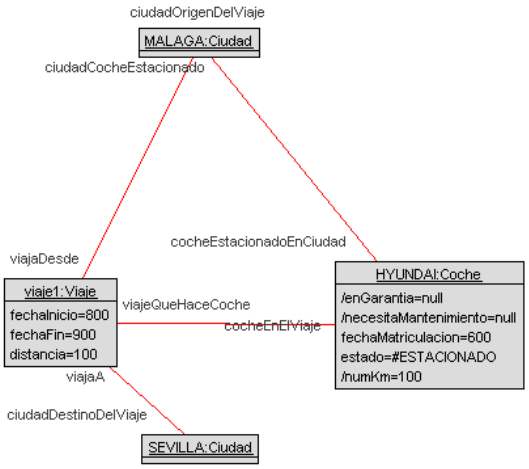


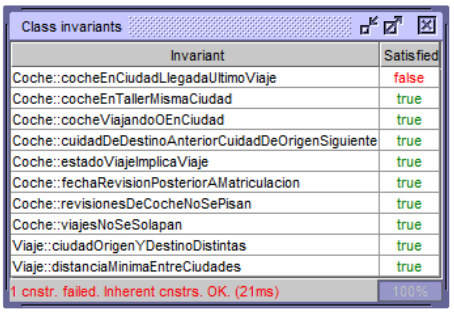
Este diagrama de objetos comprobamos que si el estado del coche es ENVIAJE, eso implica que es un viaje. Por lo que como el estado es ese, se cumple nuestra invariante.



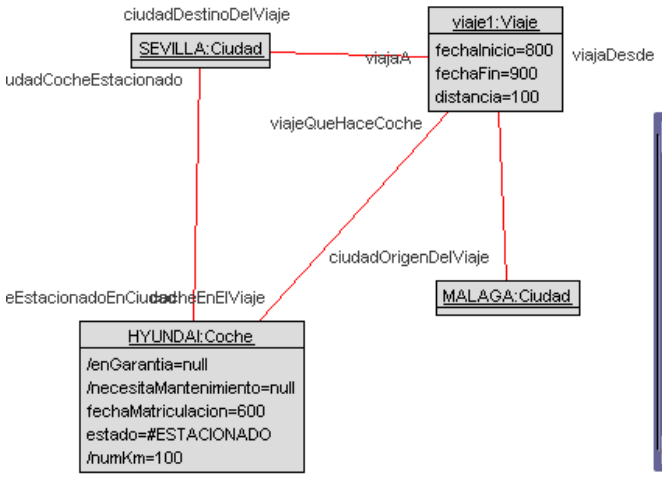


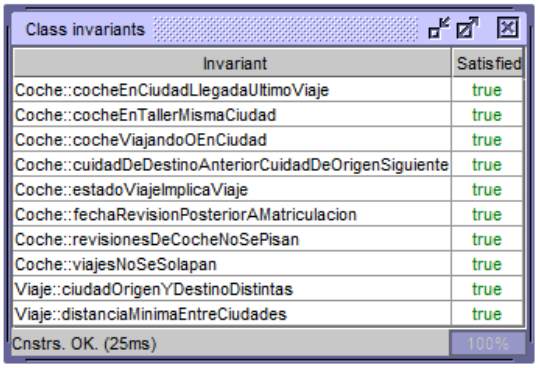
cocheEnCiudadLlegadaUltimoViaje





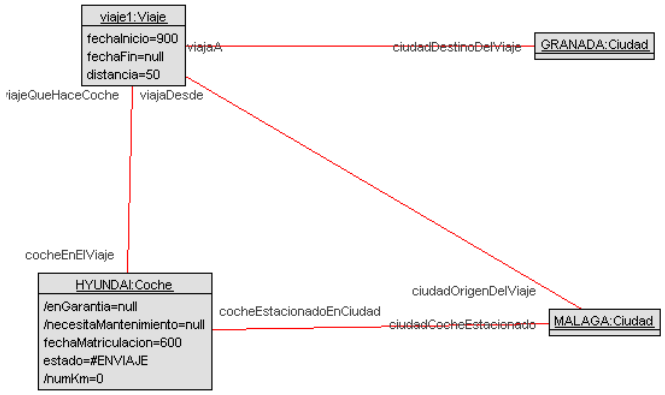
Este diagrama de objetos, muestra un objeto ‘Coche’ que ha realizado un viaje con origen MALAGA y destino SEVILLA. Tal y como modela la relación ‘EstacionadoEn’, el coche se supone estacionado en la ciudad orígen MALAGA, cuando debería estar en SEVILLA si ya ha realizado el viaje.

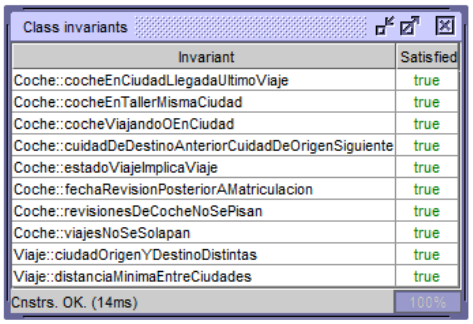




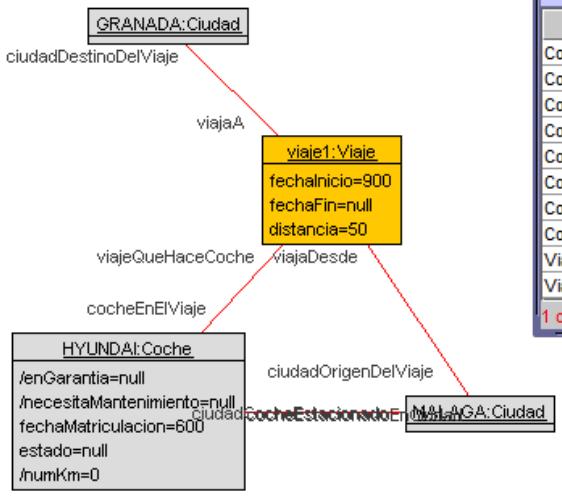
Ahora, como podemos observar el coche sí que se encuentra estacionado en SEVILLA, que es la ciudad destino del viaje que acaba de realizar. Efectivamente, la invariante controla esta restricción (ahora no nos salta a false cocheEnCiudadLlegadaUltimoViaje).

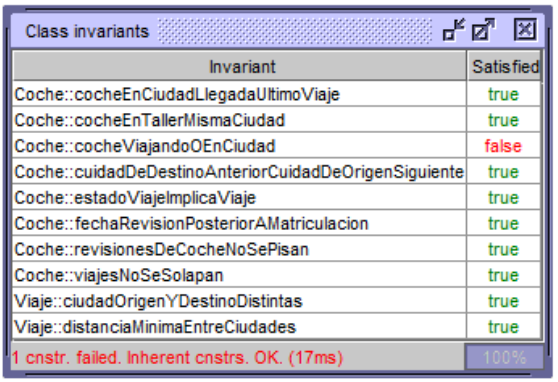
cocheViajandoOEnCiudad





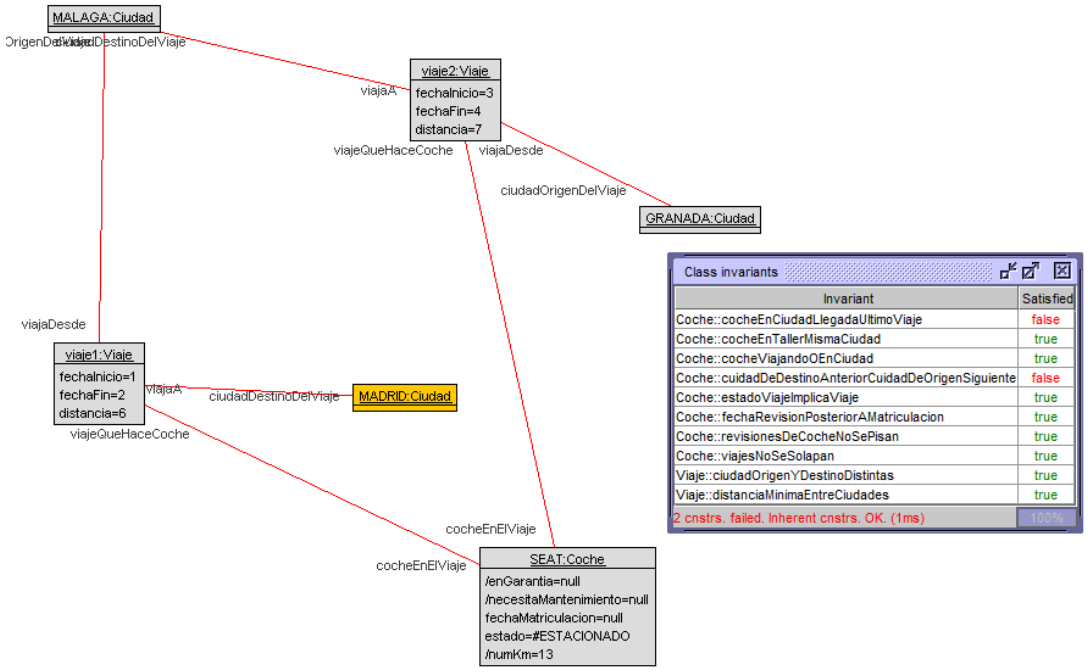
En este diagrama de objetos comprobamos que el coche se encuentre o en una ciudad o se encuentre viajando. En este caso, el coche está viajando, por lo que no nos da ningún error.



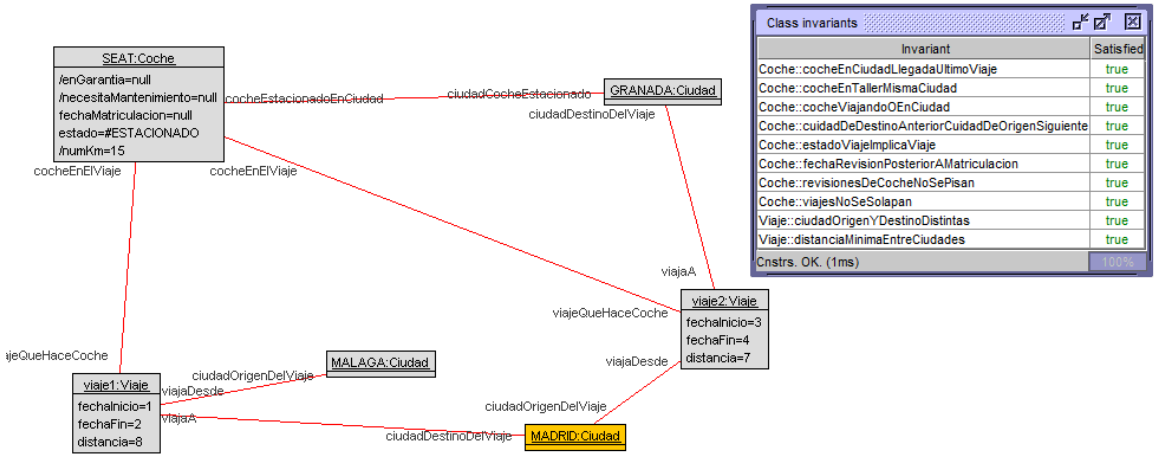


Ahora nos da error, ya que hemos puesto que el estado del coche sea null por lo que ahora nos daría un error debido a que el estado del coche debe ser o viajando o en una ciudad.

ciudadDeDestinoAnteriorCuidadDeOrigenSiguiente

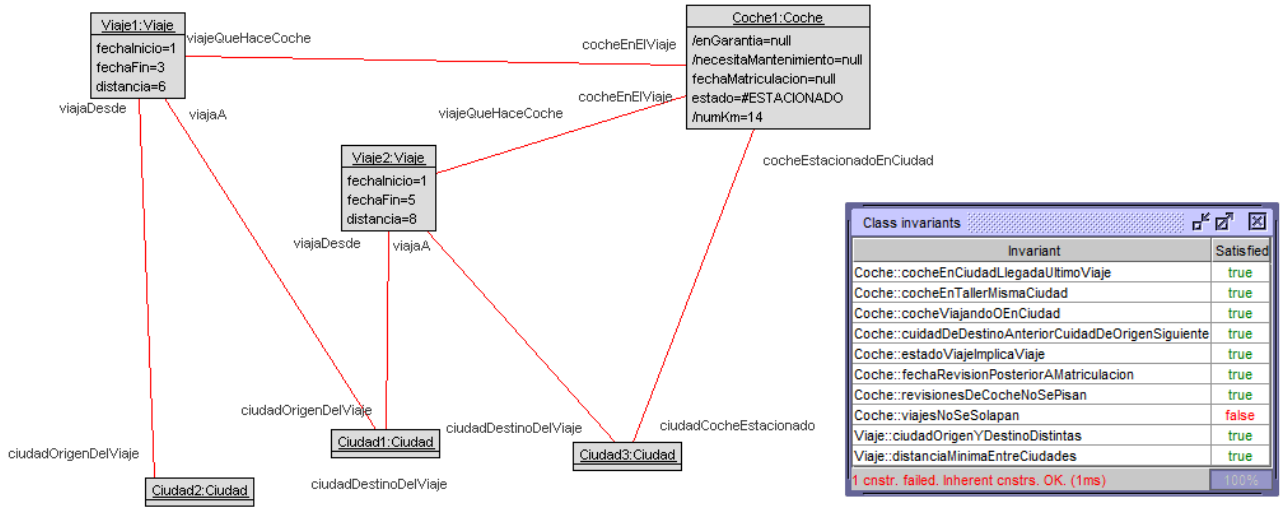


Como podemos observar, el viaje2 comienza en Granada, el cual no es el destino del viaje1 (sería Madrid). Por lo tanto, no se cumple la invariante que comprueba que el anterior destino sea el próximo origen. Además, tampoco se cumplirá aquella que comprueba que el coche que tenga al menos un viaje, debe encontrarse en la ciudad de destino de su anterior viaje.

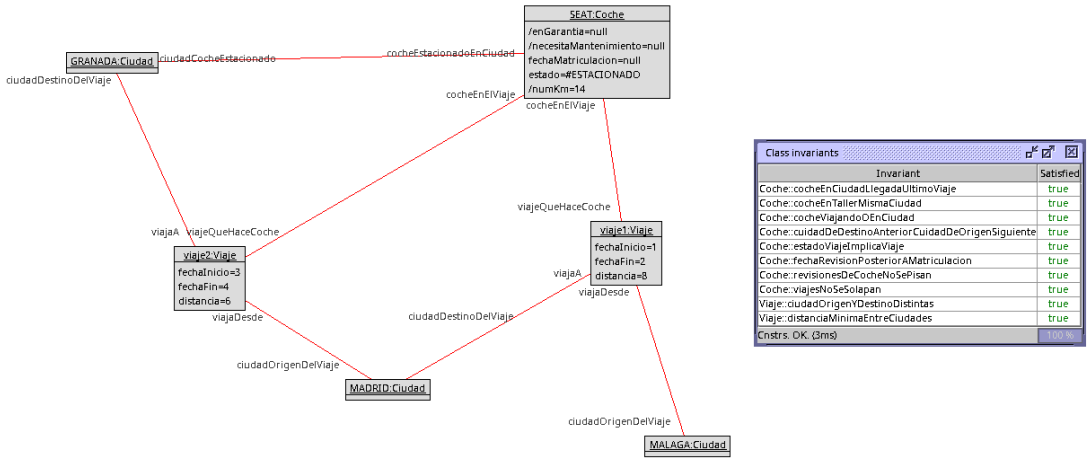


Aquí observamos que se cumple la invariante ya que la ciudad en la que empieza el viaje2 es la misma en la que terminó el viaje1.

viajesNoSeSolapan



Al ser las fechas de inicio de ambos viajes iguales, la invariante nos dará false, ya que estamos comprobando que no se solapen los viajes en el tiempo. Ambos viajes deberán empezar uno detrás del otro.



Sin embargo, en este caso el segundo viaje empieza un día después de la fecha de llegada del primer viaje, ya que, aunque no empiezan el mismo día, debe empezar como mínimo inmediatamente después del viaje anterior. Por lo tanto, la invariante nos da true.

CÓDIGO USE

* En este apartado, todo lo que esté en letra cursiva se tratará de código.

```
model Practica2

enum EstadoCoche {
   ESTACIONADO,
   ENVIAJE,
   ENTALLER
}

enum TipoRevision {
   MANTENIMIENTO,
   REPARACION
}


class Clock
attributes
   NOW : Integer init = 0 -- POSIX representation starting at 0
--       resolution: Integer init = 1
--    operations
--       tick()
--          begin
--             self.NOW := self.NOW + self.resolution;
--             for o in self.ao do
--                o.tick()
--          end;
--       end
--       post TimePasses: self.NOW = self.NOW@pre + self.resolution
--    run (n: Integer)
--       begin
--          for i in Sequence{1..n} do
--             self.tick()
--          end
end
--   constraints
--    inv PositiveResolution: self.resolution > 0



class Coche
```

```
attributes
enGarantia: Boolean
derive:
   let hoy:Integer = Clock.allInstances()->asSequence()->first().NOW in
   if hoy - self.fechaMatriculacion >= 400 then
                  let ultimaRevisionOficial:Revision= self.revisionDelCoche->select(r |
r.tallerQueRevisa.oclIsTypeOf(TallerOficial))->sortedBy(fechaFin)->last() in
                                  if hoy - (ultimaRevisionOficial.fechaFin +
ultimaRevisionOficial.tallerQueRevisa.oclAsType(TallerOficial).garantia) > 0 then
         false
      else
         true
      endif
   else
      true
   endif

necesitaMantenimiento: Boolean
derive:
   let hoy:Integer = Clock.allInstances()->asSequence()->first().NOW in
   if hoy - self.fechaMatriculacion >= 400 then
      if self.revisionDelCoche->size() = 0 then
         true
      else
         hoy - self.revisionDelCoche->sortedBy(fechaInicio)->last().fechaFin > 100
      endif
   else
      false
   endif
fechaMatriculacion: Integer

estado: EstadoCoche
numKm: Integer
derive:
            self.viajeQueHaceCoche->select(v|v.fechaFin <> null)->collect(viaje |
viaje.distancia)->sum()
end

class Ciudad
end

class Viaje
attributes
fechaInicio: Integer
```

fechaFin: Integer
distancia: Integer
end

abstract class Taller
end

class TallerOficial < Taller
attributes
garantia: Integer
end

class TallerNoOficial < Taller
end

class Revision
attributes
fechaInicio: Integer
fechaFin: Integer
tipoRevision: TipoRevision
end

----------Asociaciones----------

association estacionadoEn between
Ciudad[0..1] role ciudadCocheEstacionado
Coche[*] role cocheEstacionadoEnCiudad
end

association cocheHaceViaje between
Coche[1] role cocheEnElViaje
Viaje[*] role viajeQueHaceCoche
end

association contieneTallerNoOficial between
Ciudad[1] role ciudadConTallerNoOficial
TallerNoOficial[*] role tallerNoOficialEstaEnCiudad
end

association tallerOficialCiudad between
Ciudad[1] role ciudadConTallerOficial
TallerOficial[0..1] role tallerOficialEstaEnCiudad
end

```
association cocheEnRevision between
   Coche[1] role cocheHaciendoRevision
   Revision[*] role revisionDelCoche
end

association tallerRevision between
   Taller[1] role tallerQueRevisa
   Revision[*] role RevisionRealizadaPorTaller
end

association ciudadDestinoViaje between
   Ciudad[1] role ciudadDestinoDelViaje
   Viaje[*] role viajaA
end

association ciudadOrigenViaje between
   Ciudad[1] role ciudadOrigenDelViaje
   Viaje[*] role viajaDesde
end

--Invariantes

constraints

context Viaje
--cada ciudad debe estar al menos a 5 kilómetros de distancia de otra.
inv distanciaMinimaEntreCiudades:
   self.distancia >= 5

--la ciudad de origen y destino de un viaje deben ser distintas
inv ciudadOrigenYDestinoDistintas:
   self.ciudadOrigenDelViaje <> self.ciudadDestinoDelViaje


context Coche
--un coche no podra pasar dos revisiones a la vez
inv revisionesDeCocheNoSePisan:
        self.revisionDelCoche -> forAll (r1 | self.revisionDelCoche -> forAll (r2 | r1 <>
r2 implies (r1.fechaFin <= r2.fechaInicio or r1.fechaInicio >= r2.fechaFin)))

--fecha de revision del coche tiene que ser posterior a fecha de matriculacion
inv fechaRevisionPosteriorAMatriculacion:
        self.revisionDelCoche -> forAll(r | r.fechaInicio >= self.fechaMatriculacion)
```

```
--Si un coche está siendo sometido a revisión en un taller, el coche se debe encontrar
--en la ciudad de ese mismo taller
inv cocheEnTallerMismaCiudad:
self.revisionDelCoche->exists(r | r.fechaInicio <> null) implies(
        self.estado = #ENTALLER implies(
        let ultimoTaller: Taller = self.revisionDelCoche -> sortedBy(fechaFin) -> last().tallerQueRevisa in ultimoTaller.oclIsTypeOf(TallerOficial) and ultimoTaller.oclAsType(TallerOficial).ciudadConTallerOficial = self.ciudadCocheEstacionado or ultimoTaller.oclIsTypeOf(TallerNoOficial) and ultimoTaller.oclAsType(TallerNoOficial).ciudadConTallerNoOficial = self.ciudadCocheEstacionado
        )
)

inv estadoViajeImplicaViaje:
        self.estado = #ENVIAJE implies self.viajeQueHaceCoche -> exists(viaje | viaje.fechaFin = null)

--si un coche se encuentra en una ciudad, su próximo viaje debe empezar desde esa ciudad y
--no desde otra
inv cuidadDeDestinoAnteriorCuidadDeOrigenSiguiente:
  self.viajeQueHaceCoche -> forAll( v |
         let viajePrevio = self.viajeQueHaceCoche->select(v2 | v2.fechaFin <= v.fechaInicio)->sortedBy(fechaFin)->last() in
             viajePrevio.oclIsUndefined() or viajePrevio.ciudadDestinoDelViaje = v.ciudadOrigenDelViaje)

inv cocheViajandoOEnCiudad:
      (self.estado = EstadoCoche :: ENVIAJE) or (self.estado = EstadoCoche :: ESTACIONADO) or (self.estado = EstadoCoche :: ENTALLER)

inv cocheEnCiudadLlegadaUltimoViaje:
   self.viajeQueHaceCoche->exists(viaje | viaje.fechaFin <> null) implies
                        (self.viajeQueHaceCoche->select(v | v.fechaFin <> null)->sortedBy(fechaFin)->last().ciudadDestinoDelViaje = self.ciudadCocheEstacionado)

--2 viajes no se pueden solapar en el tiempo
inv viajesNoSeSolapan:
   self.viajeQueHaceCoche->forAll(v1, v2 |
```

v1 <> v2 and v1.fechaFin <>null and v2.fechaFin<>null implies (v1.fechaInicio >= v2.fechaFin or v1.fechaFin <= v2.fechaInicio))

APARTADO B

ACCIONES

- Un coche comienza un viaje desde la ciudad en la que se encuentra.
- Una operación avanzar que se ejecuta sobre los coches, y que no recibe ningún parámetro.
- Se debe modelar el paso del tiempo, de modo que un tic del reloj representa el paso de un día. Se debe tener en cuenta a la hora de que los coches puedan avanzar en el viaje que estén realizando.
- La clase ‘Clock’ ahorá tendrá las siguientes acciones:

    - Tick (): avanza un día
    - Run (n : Integer): avanza n días

RESTRICCIONES

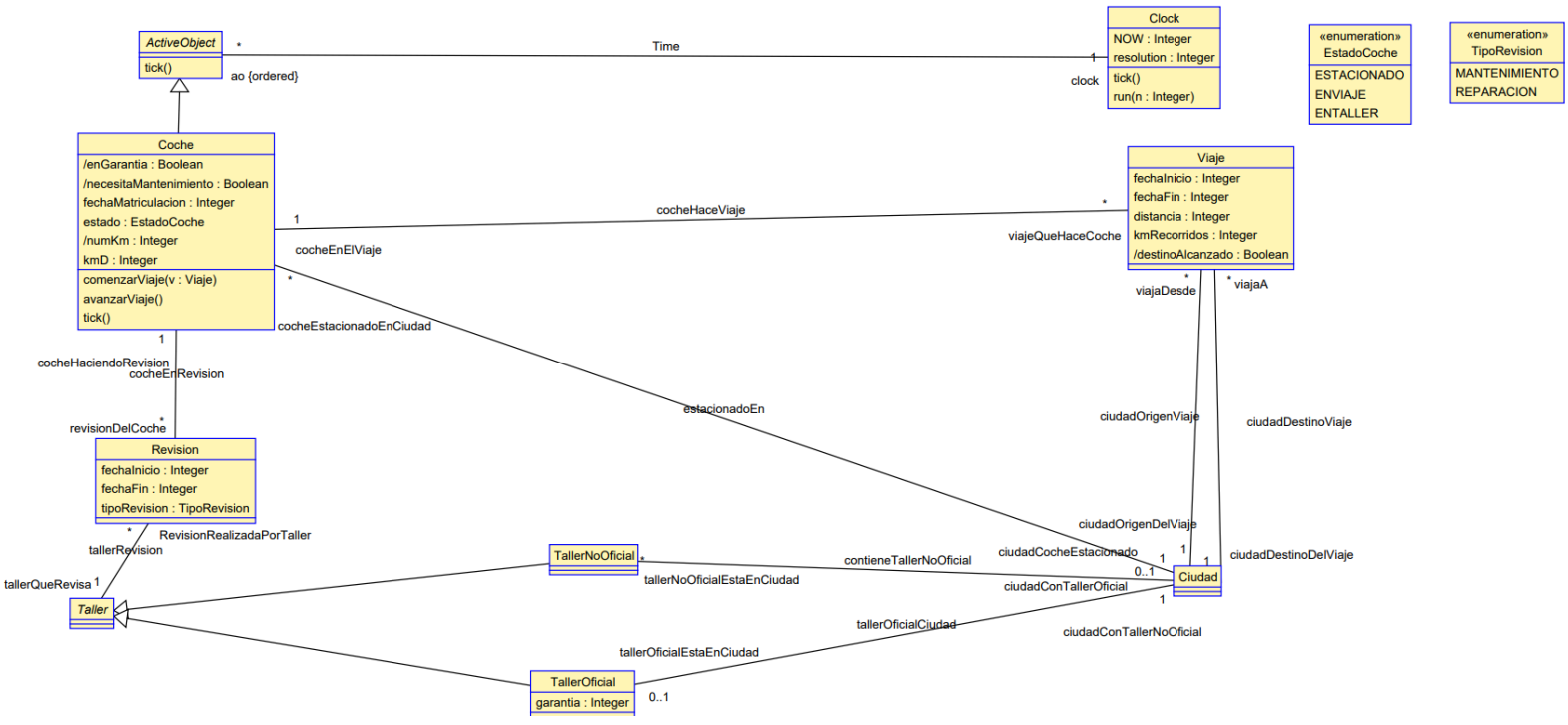
Los invariantes en este apartado serán los mismos que en el apartado A, pues en este apartado se introduce solo el paso del tiempo.

Tan solo se introduce un invariante nuevo en el contexto de Clock:

PositiveResolution

Este invariante controlará que el atributo NOW (que representa nuestra fecha actual) siempre se vea incrementado positivamente (el tiempo no puede ir hacia atrás).

DIAGRAMA DE CLASES (USE)

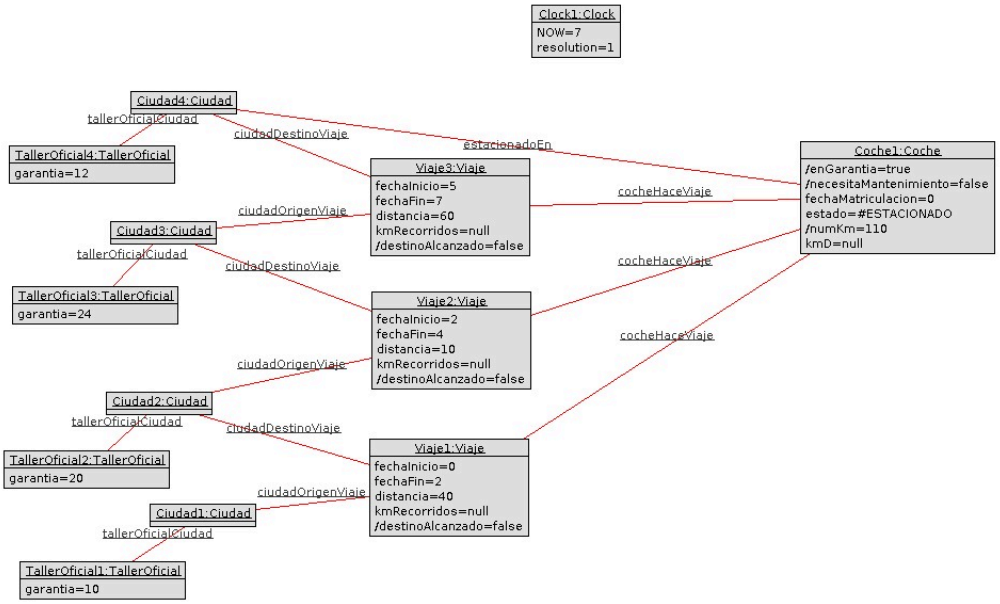


El diagrama de clases es prácticamente idéntico al del apartado anterior, exceptuando que en la clase coche ahora se implementan los métodos

comenzarViaje(Viaje v), avanzarViaje() y tick(), que modelarán el cómo el coche avanza de una ciudad a otra, actualizando sus kms recorridos y la ciudad en la que se encuentran.

DIAGRAMA DE OBJETOS (.SOIL)

A continuación, se muestra un diagrama de objetos completo con varios objetos de todas las clases que simulan el comportamiento de un coche que ha realizado 3 viajes y se ha actualizado correctamente el atributo numKm con los kilómetros en total que ha recorrido el coche:



CÓDIGO USE

* En este apartado, todo lo que esté en letra cursiva se tratará de código.

A continuación, mostramos el código nuevo que ha sido introducido, como el de la clase Clock, la clase activeObject y su respectiva relación.

También se ha modificado la clase 'Coche', para poder introducir las operaciones para que este pudiera realizar los viajes, con sus respectivas precondiciones y postcondiciones:

```
class Clock

   attributes
      NOW : Integer init = 0 -- POSIX representation starting at 0
      resolution: Integer init = 1
   operations
      tick()
         begin
            self.NOW := self.NOW + self.resolution;
            for o in self.ao do
               o.tick()
         end;
      end
      post TimePasses: self.NOW = self.NOW@pre + self.resolution
      run (n: Integer)
         begin
            for i in Sequence{1..n} do
               self.tick()
         end
      end
   constraints
      inv PositiveResolution: self.resolution > 0
end

  abstract class ActiveObject -- real-time objects
   operations
      tick() begin end
end

association Time between
```

```
    Clock [1]
    ActiveObject [*] role ao ordered
end
```

```
class Coche < ActiveObject
attributes
enGarantia: Boolean
derive:
    let hoy:Integer = Clock.allInstances()->asSequence()->first().NOW in
    if hoy - self.fechaMatriculacion >= 400 then
                    let   ultimaRevisionOficial:Revision=   self.revisionDelCoche->select(r  |
r.tallerQueRevisa.oclIsTypeOf(TallerOficial))->sortedBy(fechaFin)->last() in
                                        if      hoy      -      (ultimaRevisionOficial.fechaFin      +
ultimaRevisionOficial.tallerQueRevisa.oclAsType(TallerOficial).garantia) > 0 then
            false
        else
            true
        endif
    else
        true
    endif
necesitaMantenimiento: Boolean
derive:
    let hoy:Integer = Clock.allInstances()->asSequence()->first().NOW in
    if hoy - self.fechaMatriculacion >= 400 then
        if self.revisionDelCoche->size() = 0 then
            true
        else
            hoy - self.revisionDelCoche->sortedBy(fechaInicio)->last().fechaFin > 100
        endif
    else
        false
    endif
fechaMatriculacion: Integer
estado: EstadoCoche
numKm: Integer
derive:
```

```
        self.viajeQueHaceCoche->select(v|v.fechaFin <> null)->collect(viaje |
viaje.distancia)->sum()
kmD: Integer
  operations
    comenzarViaje(v : Viaje)
      begin
        declare dia : Integer;
        dia:=Clock.allInstances()->asSequence()->any(true).NOW;
        insert(self, v) into cocheHaceViaje;
        self.estado:=#ENVIAJE;
        v.fechaInicio:= dia;
        v.kmRecorridos:=0;
      end
                  pre CondicionCoche : (self.estado=#ESTACIONADO) and
(v.kmRecorridos=null)
      post EstadoActualizado : (self.estado = #ENVIAJE)
                                post FechaInicioEstablecida: (v.fechaInicio =
Clock.allInstances()->asSequence()->any(true).NOW)
      post ViajeAsignadoAlCoche: (self.viajeQueHaceCoche->includes(v))

    avanzarViaje() --Cada tick ejecuta un avanzarViaje
      begin
        declare v : Viaje, dia : Integer;
        dia:=Clock.allInstances()->asSequence()->any(true).NOW;
                                    v:=self.viajeQueHaceCoche->select(v | not
v.destinoAlcanzado)->any(true);
          if ((v.kmRecorridos + self.kmD) >= v.distancia) then
            v.kmRecorridos:=v.distancia;
            self.estado:=#ESTACIONADO;
            v.fechaFin:=dia;
          else
            v.kmRecorridos:=v.kmRecorridos + self.kmD
          end
      end
      pre YaComenzado: (self.estado = #ENVIAJE)
                  post CondicionAvanzadoSiDestinoAlcanzo: let v : Viaje =
self.viajeQueHaceCoche->select(v | not v.destinoAlcanzado)->any(true) in
(v.kmRecorridos = v.distancia) implies (self.estado = #ESTACIONADO)
                  post FechaFinDefinidaSiDestinoAlcanzado: let v : Viaje =
self.viajeQueHaceCoche->select(v | not v.destinoAlcanzado@pre)->any(true) in
(v.kmRecorridos = v.distancia) implies (v.fechaFin =
Clock.allInstances()->asSequence()->any(true).NOW)
                   post KilometrajeActualizadoSiNoDestino: let v : Viaje =
self.viajeQueHaceCoche->select(v | not v.destinoAlcanzado)->any(true) in
```

(v.kmRecorridos < v.distancia) implies (v.kmRecorridos = v.kmRecorridos@pre + self.kmD)

tick()

begin

if not self.viajeQueHaceCoche->select(v | not v.destinoAlcanzado)->any(true).destinoAlcanzado then

self.avanzarViaje()

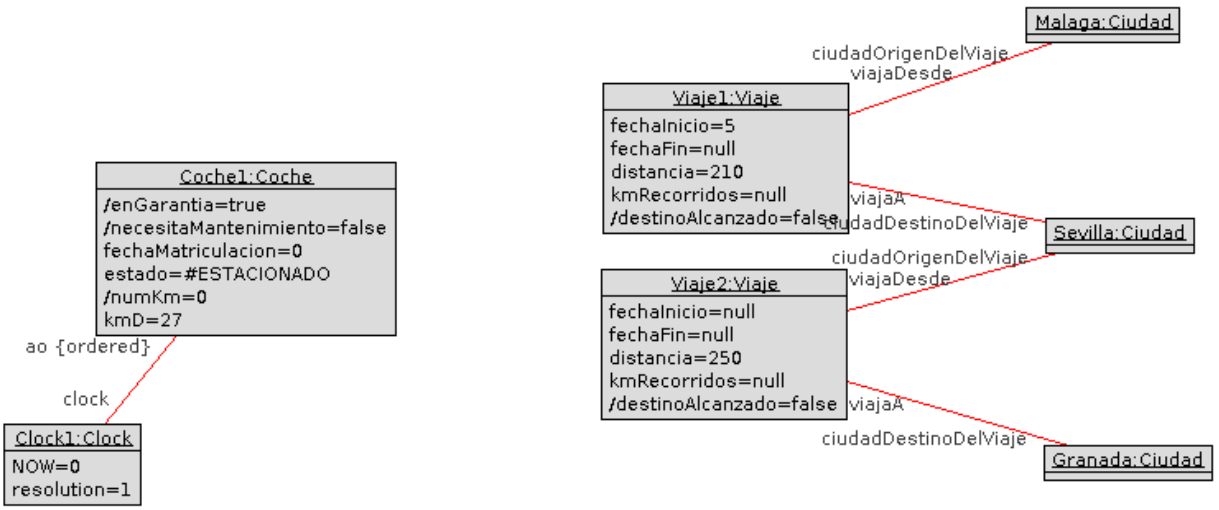
end

end

APARTADO C

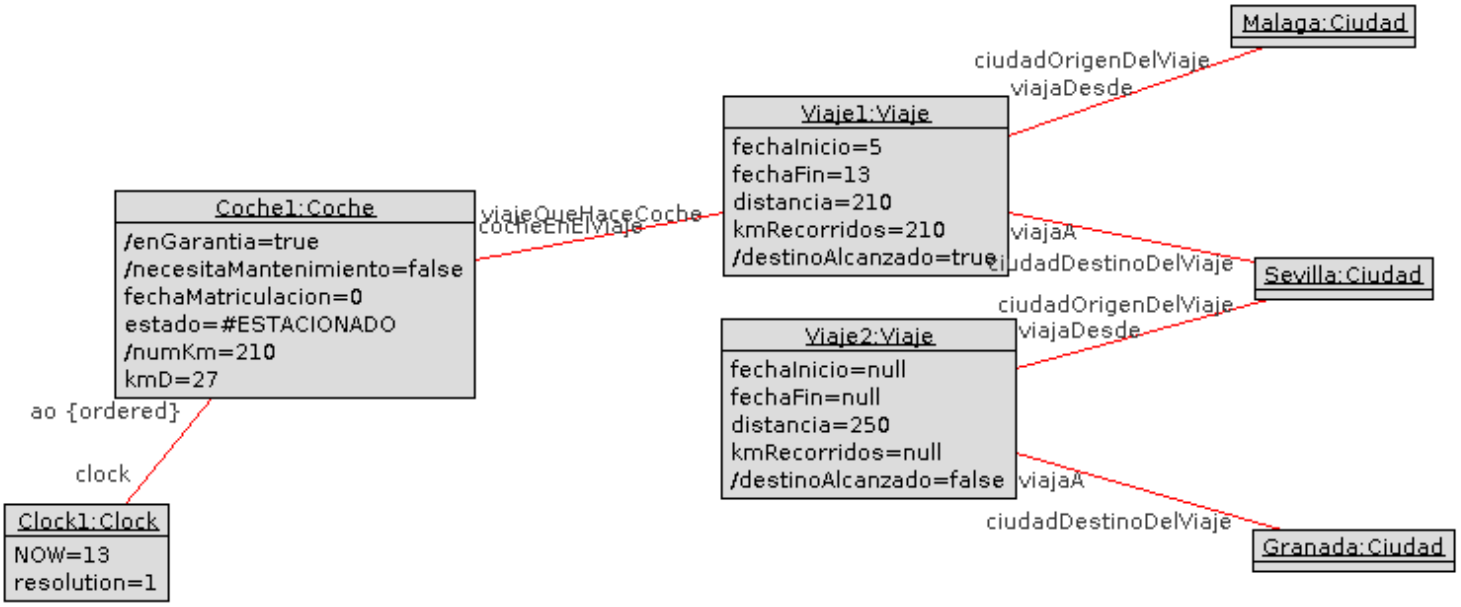
En este apartado hemos desarrollado el modelo de objetos que se pide y lo hemos simulado. A continuación se muestran las capturas de cada una de las instancias que se piden:

CAPTURA INSTANCIA 0



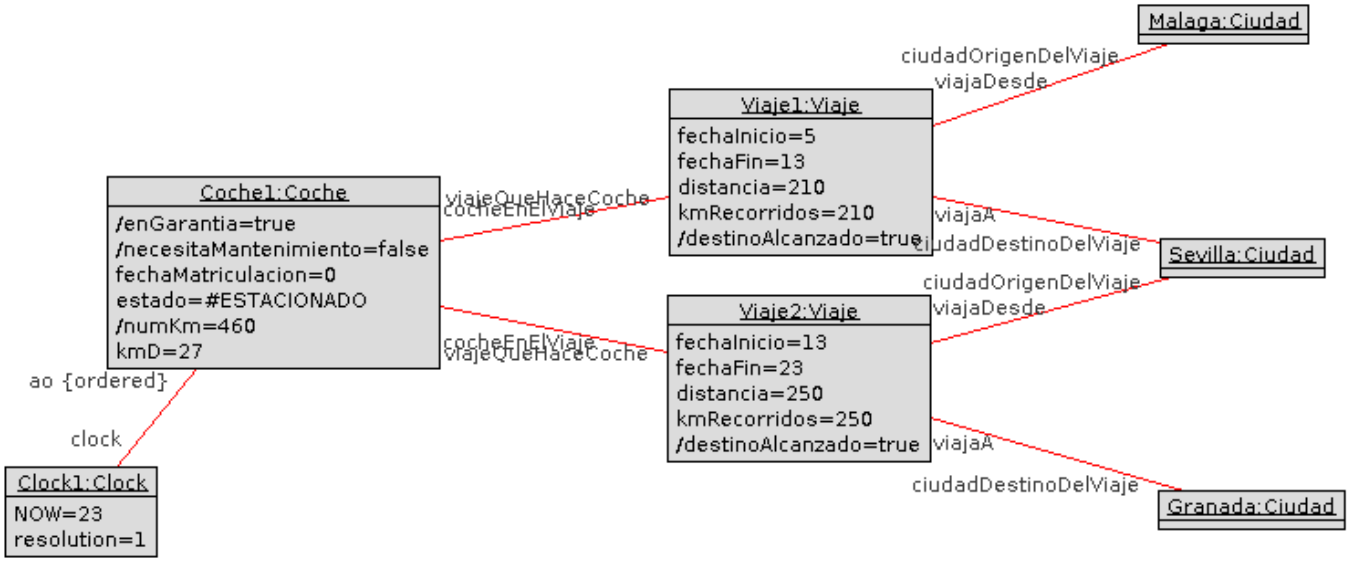
En la instancia 0 nos encontramos en el día 0 con un coche (Coche1) matriculado estacionado en Málaga y los viajes entre Málaga y Sevilla (210 km), y entre Sevilla y Granada (250 km) creados. El coche tiene 0 kilómetros recorridos y está en garantía.

CAPTURA INSTANCIA 1



En la instancia 1 estamos en el día 13 con el coche estacionado en Sevilla, después de hacer el viaje de Málaga a Sevilla. El coche tiene 210 kilómetros recorridos y podemos ver que el viaje (Viaje1) ha finalizado, ya que estamos en la fecha de fin del viaje (13) y la distancia es igual a la de kilómetros recorridos del viaje (210 km)

CAPTURA INSTANCIA 2



En la instancia 2, es el día 23 y el coche se encuentra estacionado en Granada, tras haber completado el viaje de Sevilla a Granada. El coche tiene 460 kilómetros

recorridos y podemos ver que el viaje (Viaje2) ha concluido, ya que estamos en la fecha de fin del viaje (23) y la distancia es igual a la de kilómetros recorridos del viaje (250 km).

CÓDIGO USE

* En este apartado, todo lo que esté en letra cursiva se tratará de código.

!new Ciudad('Malaga')
!new Ciudad('Sevilla')
!new Ciudad('Granada')
!new Coche
!Coche1.fechaMatriculacion := 0
!Coche1.estado := EstadoCoche::ESTACIONADO
!Coche1.kmD := 27
!new Viaje
!Viaje1.fechaInicio := 5
!Viaje1.distancia := 210
!new Viaje
!Viaje2.distancia := 250
!insert (Malaga,Viaje1) into ciudadOrigenViaje
!insert (Sevilla,Viaje1) into ciudadDestinoViaje
!insert (Sevilla,Viaje2) into ciudadOrigenViaje
!insert (Granada,Viaje2) into ciudadDestinoViaje
!new Clock

--Instante 0

!insert (Clock1,Coche1) into Time
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Coche1.comenzarViaje(Viaje1)
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()

!Clock1.tick()

--Instante 1

!Coche1.comenzarViaje(Viaje2)
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()
!Clock1.tick()

--Instante 2