Modelado y Diseño del Software

Memoria de la Práctica 2

Participantes:

Nombre	Apellidos	Correo
Pau/Pablo	Fernández Rueda	pauf@uma.es
Eduardo	García Rivas	eduardogarr@uma.es
Francisco Eloy	González Castillo	eloygonzalez@uma.es
María Paulina	Ordóñez Walkowiak	mpow@uma.es
Javier	Toledo Delgado	javier.toledo.delgado@uma.es
Daniela	Suárez Morales	danielasuarez@uma.es

Índice

Introducción

En el siguiente documento se muestran los diagramas de clases en las aplicaciones Visual Paradigm y USE. Además de mostrar las restricciones a las que está sujeta el modelo, las variables derivadas que contiene y algunas notas aclaratorias para facilitar la comprensión y entendimiento del mismo, sus entidades, relaciones y diagramas. En esta práctica, además ,se modelará el comportamiento del sistema, específicamente el de los viajes de los coches.

Al final del documento se recopilan los códigos de los diagramas y pruebas.

Parte 1: Modelo Conceptual del Sistema

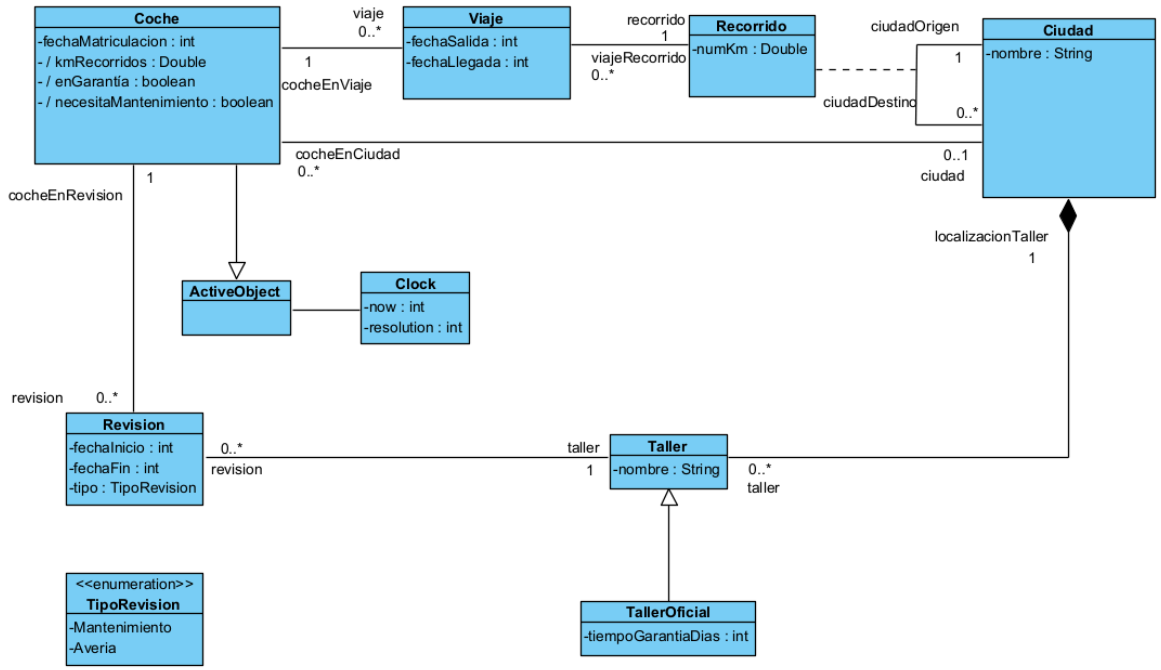
En este apartado se desarrollará en UML, utilizando las herramientas USE y Visual Paradigm, un modelo conceptual de la estructura del sistema del enunciado, con los correspondientes elementos, las relaciones entre ellos y todas las restricciones de integridad necesarias (esto último solamente en la herramienta USE).

En lo relacionado con el tiempo, en este apartado consideraremos que tenemos una clase Clock de la cual habrá una única instancia y estará asociada con el coche, con el fin de simplificar el acceso a sus atributos. Dicha clase tiene un atributo “now” que nos indica el día en que nos encontramos actualmente.

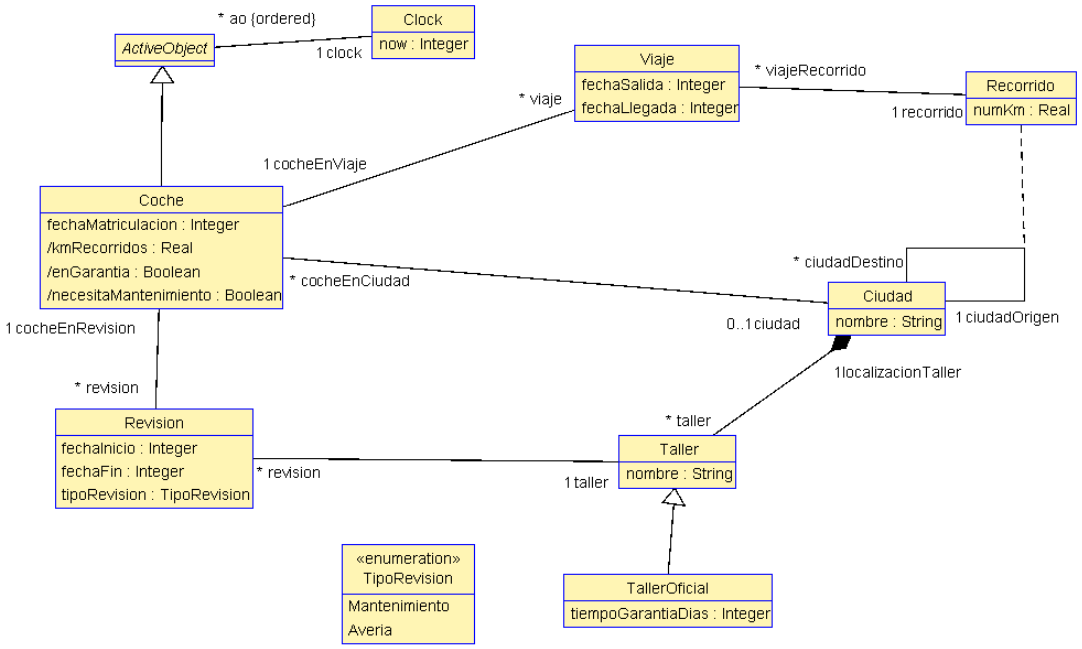
El código de esta parte se encuentra en el apartado Practica2.use (Apartado A).

Diagramas de clases (Apartado A)

- Visual Paradigm:



- USE:



Explicaciones adicionales

Herencias:

- "TallerOficial" hereda de "Taller": ya que "TallerOficial" es un tipo de "Taller" pero con la única diferencia de que ofrece garantía a los coches que hacen revisiones en el mismo.

Asociaciones:

- "Coche" y "Ciudad": un coche puede estar viajando o en una ciudad. Para especificar esto hemos optado por crear una relación de asociación (0..1) con "Coche" y "Ciudad", por tanto un coche no puede estar en una ciudad si está viajando.
- "Viaje" y "Recorrido": esta relación permite que un recorrido pueda estar asociado a muchos viajes y que un viaje solo contenga un único recorrido.
- "Coche" y "Revisión": esta relación sirve para indicar el número de revisiones que ha tenido el coche.
- "Coche" y "Viajes": esta relación sirve para indicar el número de viajes que ha tenido el coche.

Clases asociación:

- Hemos creado una clase asociación con "Ciudad" para registrar los recorridos entre las ciudades, pudiendo haber ciudades aisladas o conectadas a varias.

Composición:

- "Taller" y "Ciudad": los talleres están asociados a una ciudad, por lo que, en caso de que esta sea eliminada, todos los talleres asociados deberán ser igualmente suprimidos.

Clases:

- Finalmente hemos añadido la clase "Clock", aunque no se modelará el paso del tiempo en este apartado, la utilizaremos para comprobar las restricciones relacionadas con las fechas y posiciones de los coches.

Variables Derivadas

A continuación se realizarán las explicaciones referentes a los atributos derivados del modelo, además de comprobaciones.

Clase Coche:

1. **“kmRecorridos”** : indica la cantidad de kilómetros realizados por un coche.

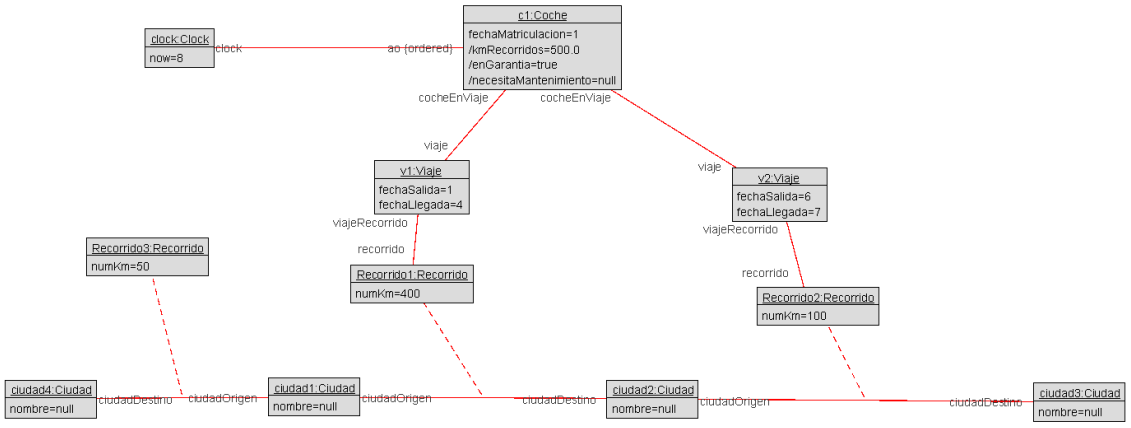
```
kmRecorridos : Real
        derive : self.viaje->select(v | v.fechaLlegada <=
self.clock.now).recorrido.numKm->sum()
```

Comprobación:

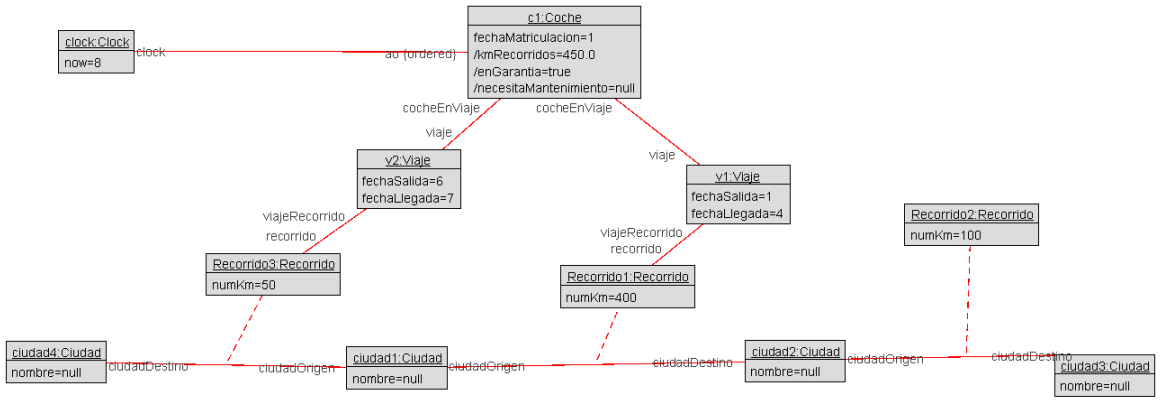
```
!new Clock ('clock')
!clock.now := 2
--!clock.now := 8


!new Coche ('c1')
!c1.fechaMatriculacion := 1

!insert(clock,c1) into Time


!new Ciudad ('ciudad1')

!new Ciudad ('ciudad2')

!new Ciudad ('ciudad3')

!new Ciudad ('ciudad4')



!insert (ciudad1,ciudad2) into Recorrido
!Recorrido1.numKm := 400


!insert (ciudad2,ciudad3) into Recorrido
!Recorrido2.numKm := 100


!insert (ciudad1,ciudad4) into Recorrido
```

```
!Recorrido3.numKm := 50


!new Viaje ('v1')
!v1.fechaSalida := 1
!v1.fechaLlegada := 4


!new Viaje ('v2')
!v2.fechaSalida := 6
!v2.fechaLlegada := 7


!insert (v1,Recorrido1) into ViajeRecorrido
--!insert (v2,Recorrido3) into ViajeRecorrido
!insert (v2,Recorrido2) into ViajeRecorrido


!insert (c1,v1) into ViajeCoche
!insert (c1,v2) into ViajeCoche
```

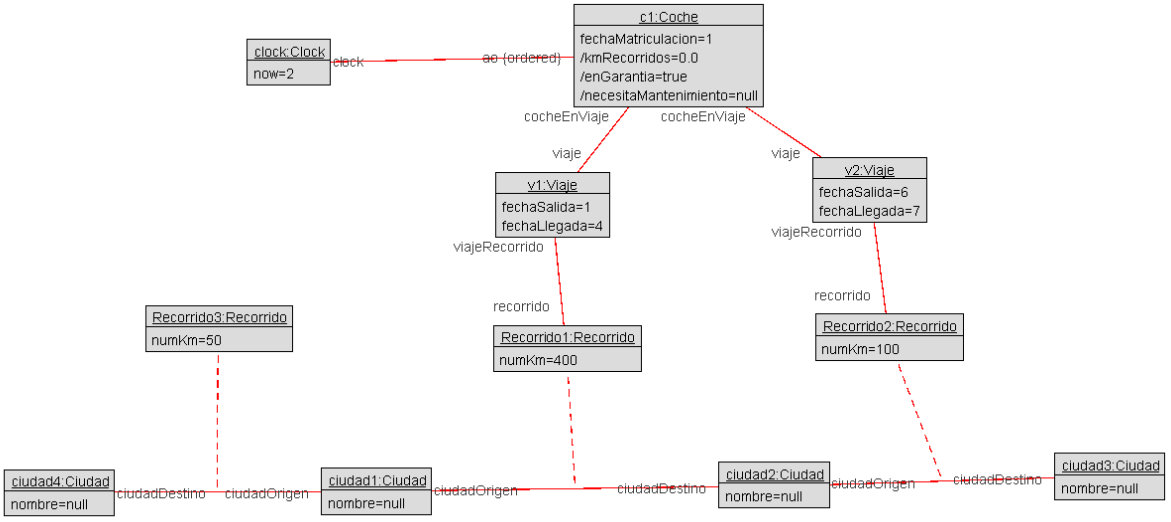
Explicación:



Como podemos observar en el viaje 1 ha recorrido 400 km y en el viaje 2 ha recorrido 100 km, y como los dos viajes tienen de fecha de llegada anterior al atributo ‘now’ del reloj los kilómetros recorridos son 500 km.



Como podemos observar, en el viaje 1 ha recorrido 400 km y en el viaje 2 ha recorrido 50 km; como los dos viajes tienen de fecha de llegada anterior al atributo 'now' del reloj los kilómetros recorridos son 450 km.



Como las fechas de llegada de los dos viajes son posteriores al atributo now del reloj, no cuentan los km de los viajes para kmRecorridos entonces es 0.

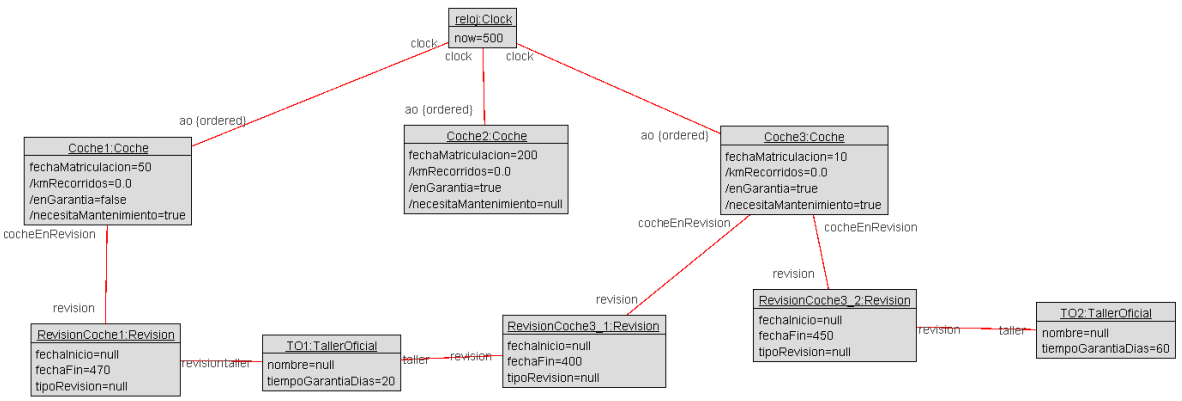
2. **“enGarantia”:** indica si el coche está en garantía o no

```
enGarantia : Boolean
        derive: (self.clock.now - self.fechaMatriculacion) < 400
or
            self.revision->exists(r |
r.taller.oclIsTypeOf(TallerOficial) and
                    (self.clock.now - r.fechaFin) <=

r.taller.oclAsType(TallerOficial).tiempoGarantiaDias)
```

Comprobación:

```
!new Clock('reloj')
!reloj.now := 500
-- para que el invariante no se cumpla hay que cambiar now por

!new TallerOficial('TO1')
!TO1.tiempoGarantiaDias := 20

--No está en garantía
!new Coche('Coche1')
!Coche1.fechaMatriculacion := 50
-- now - fechaMatriculacion = 430, no tiene garantia

!insert(reloj,Coche1) into Time

!new Revision('RevisionCoche1')
!RevisionCoche1.fechaFin := 470
-- fechaFin + tiempoGarantiaDias = 490, y como now < 490, tiene
garantia

!insert(Coche1, RevisionCoche1) into RevisionCoche
!insert(TO1, RevisionCoche1) into RevisionTaller

--Han pasado menos de 4 años desde la fecha de matriculación
--entonces está en garantía

!new Coche('Coche2')
!Coche2.fechaMatriculacion := 200
```

```
!insert(reloj,Coche2) into Time

!new TallerOficial('TO2')
!TO2.tiempoGarantiaDias := 60

--Han pasado más de 4 años desde la fecha de matriculación
--pero la revisión que tuvo el día 450 en un taller Oficial le
otorga
--60 días de garantía, entonces está en garantía hasta el día 510
!new Coche('Coche3')
!Coche3.fechaMatriculacion := 10

!insert(reloj,Coche3) into Time

!new Revision('RevisionCoche3_1')
!RevisionCoche3_1.fechaFin := 400

!new Revision('RevisionCoche3_2')
!RevisionCoche3_2.fechaFin := 450

!insert(Coche3, RevisionCoche3_1) into RevisionCoche
!insert(Coche3, RevisionCoche3_2) into RevisionCoche
!insert(TO1, RevisionCoche3_1) into RevisionTaller
!insert(TO2, RevisionCoche3_2) into RevisionTaller
```



Explicación:

- El coche 1 no está en garantía porque han pasado más de 4 años desde la fecha de matriculación y la revisión con el taller oficial que fue el día 470 le otorgó 20 días de garantía(estuvo hasta el día 490 en garantía), entonces como está en el día 500 no tiene garantía.
- El coche 2 está en garantía porque han pasado menos de 4 años desde la fecha de matriculación.

- El coche 3 está en garantía porque la última revisión que tuvo en un taller oficial fue el día 450 y le otorgó 60 días de garantía (estará en garantía hasta el día 510), entonces como esta en el día 500 tiene garantía.

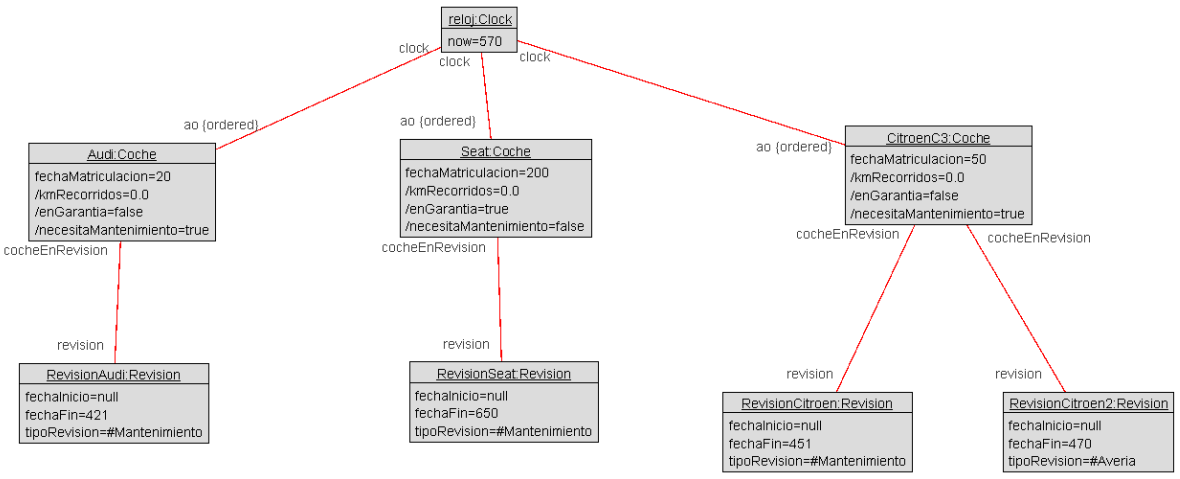
3. **“necesitaMantenimiento”:** Un coche necesita mantenimiento si ha pasado cuatro años desde su fecha de matriculación o si ha un año desde la última revisión de mantenimiento.

```
necesitaMantenimiento : Boolean
        derive : (self.clock.now - self.fechaMatriculacion) > 400
or
                (self.clock.now - self.revision
                            ->select(r | r.tipoRevision =
TipoRevision::Mantenimiento)

->sortedBy(fechaFin)->last().fechaFin) > 100
```

Comprobación:

```
!new Clock('reloj')
!reloj.now := 570

--El coche Audi ha pasado más de 4 años desde la fecha de
matriculación
--entonces necesita mantenimiento
!new Coche('Audi')
!Audi.fechaMatriculacion := 20

!insert(reloj, Audi) into Time

!new Revision('RevisionAudi')
!RevisionAudi.fechaFin := 421
!RevisionAudi.tipoRevision := TipoRevision::Mantenimiento

--El coche Seat no ha pasado 4 años desde la fecha de
matriculación
--ni un año desde la última revisión de mantenimiento entonces no
necesita mantenimiento
!new Coche('Seat')
```

```
!Seat.fechaMatriculacion := 200

!insert(reloj, Seat) into Time

!new Revision('RevisionSeat')
!RevisionSeat.fechaFin := 650
!RevisionSeat.tipoRevision := TipoRevision::Mantenimiento

--El coche CitroenC3 ha pasado más de un año desde la última
revisión de mantenimiento
--porque la revision 2 es de tipo Averia por lo que no cuenta
--entonces necesita mantenimiento
!new Coche('CitroenC3')
!CitroenC3.fechaMatriculacion := 50

!insert(reloj, CitroenC3) into Time

!new Revision('RevisionCitroen')
!RevisionCitroen.fechaFin := 451
!RevisionCitroen.tipoRevision := TipoRevision::Mantenimiento

!new Revision('RevisionCitroen2')
!RevisionCitroen2.fechaFin := 470
!RevisionCitroen2.tipoRevision := TipoRevision::Averia

!insert(CitroenC3, RevisionCitroen) into RevisionCoche
!insert(Audi, RevisionAudi) into RevisionCoche
!insert(Seat, RevisionSeat) into RevisionCoche
!insert(CitroenC3, RevisionCitroen2) into RevisionCoche
```



Explicación:

- Now vale 570
- El coche Audi ha pasado más de 4 años (20+400+1, añado el +1 porque es se comprueba que haya pasado > 400, es decir, es estrictamente positiva) desde la fecha de matriculación y desde la última revisión de mantenimiento han pasado mas de 100 días (421+100) entonces **necesita mantenimiento**
- El coche Seat no ha pasado 4 años (200 + 370) desde la fecha de matriculación ni un año desde la última revisión de mantenimiento entonces **no necesita mantenimiento.**
- El coche Citroen ha pasado más de 4 años (50+400+1) desde la fecha de matriculación y más de 100 días (451+100) desde la última revisión de mantenimiento y tiene una revision pero es de avería por lo que no nos dan 100 días para la próxima revisión por lo que **sigue necesitando mantenimiento.**

Restricciones y Comprobaciones

En el siguiente apartado del documento se representan todas las invariantes/restricciones del modelo. En muchos casos los valores numéricos no coinciden con los que se piden, esto es con el objetivo de poder hacer las pruebas, pero en el código final esto estará cambiado:

1. **"DistanciaMinima"**: las distancias entre ciudades deben de ser de más de 5 kilómetros.

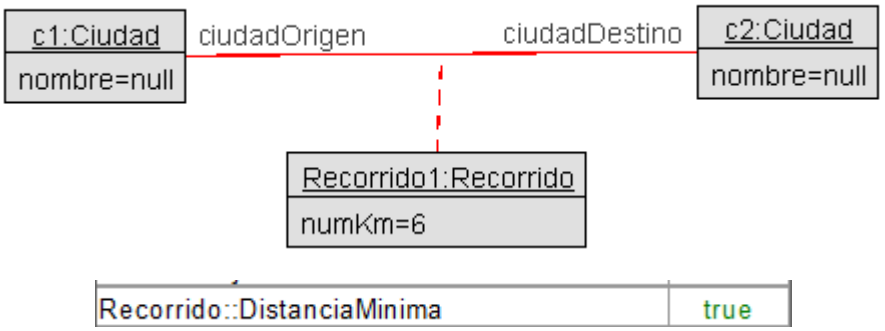
- Restricción:

```
inv DistanciaMinima:
        self.numKm > 5
```

- Comprobación:

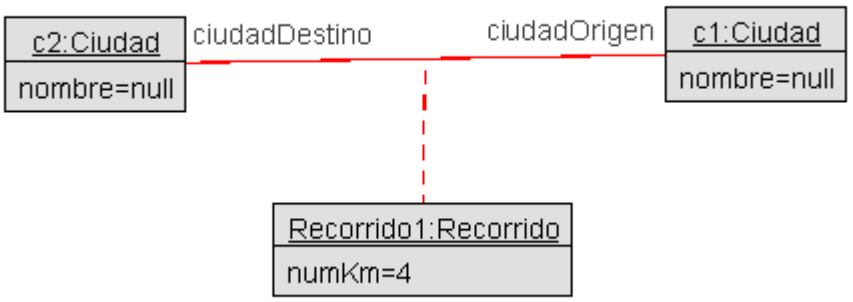
```
!new Ciudad('c1')
!new Ciudad('c2')

!insert (c1,c2) into Recorrido

!Recorrido1.numKm := 6
```

- Explicación Caso 1 (se cumple el invariante):



En este caso, se puede observar numKm mayor que 5 por lo que se cumple el invariante.

- Explicación Caso 2 (se incumple el invariante):



Recorrido::DistanciaMinima	false

En este caso, se puede observar que numKm menor que 5, por lo que no se cumple el invariante.

2. **“fechaMatriculacionValida”**: la fecha de las matriculaciones debe de tener un formato adecuado (positivo y no nulo).

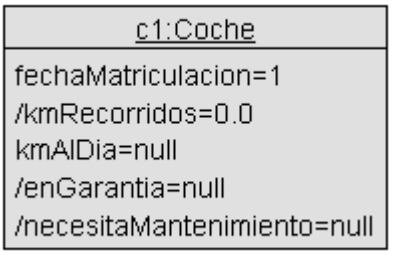
- Restricción:

```
      inv fechaMatriculacionValida:
            self.fechaMatriculacion >= 0 and self.fechaMatriculacion <>
null
```

- Comprobación:

```
!new Coche('c1')
!c1.fechaMatriculacion := 1
-- para que el invariante no se cumpla, cambiamos la fecha de
matriculación a un valor negativo o comentamos la línea
```

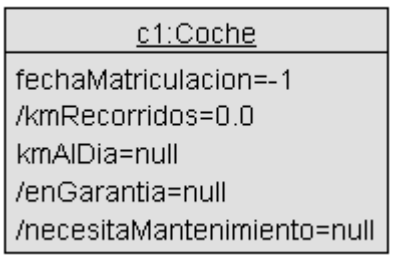
- Explicación Caso 1 (se cumple el invariante):

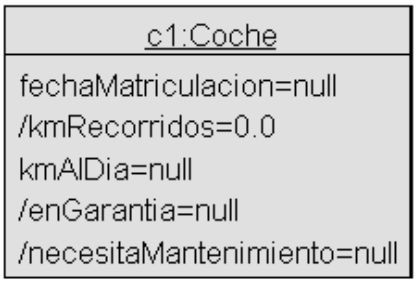


Coche::fechaMatriculacionValida	true

En este caso, se puede observar que al ser fechaMatriculacion un número no negativo y no nulo, se cumple el invariante.

- Explicación Caso 2 (se incumple el invariante):





Coche::fechaMatriculacionValida	false

En este caso, se puede observar que al ser fechaMatriculacion un número negativo o nulo , se incumple el invariante.

3. **"fechasRevisionValidas"**: la fecha de las revisiones debe de tener un formato adecuado (positivo y no nulo).

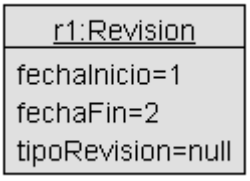
- Restricción:

```
inv fechasRevisionValidas:
        ((self.fechaInicio <> null and self.fechaInicio>=0) and
        (self.fechaFin <> null and self.fechaFin>=0)) and
        (self.fechaInicio <= self.fechaFin)
```

- Comprobación:

```
!new Revision('r1')
!r1.fechaInicio := 1
!r1.fechaFin := 2
```

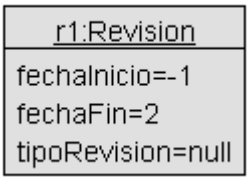
- Explicación Caso 1 (se cumple el invariante):



Revision::fechasRevisionValidas	true

En este caso, se puede observar que fechaInicio y fechaFin son números no negativos y fechaInicio es menor que fechaFin por lo que se cumple el invariante

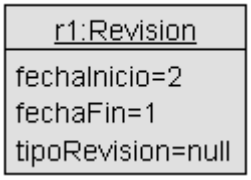
- Explicación Caso 2 (se incumple el invariante):



Revision::fechasRevisionValidas	false

En este caso, se puede observar que fechaInicio es un número negativo por lo que no se cumple el invariante, esto pasaría cuando fechaFin es negativa o ambos lo son.

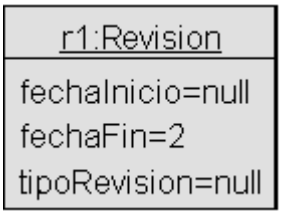
- Explicación Caso 3 (se incumple el invariante):

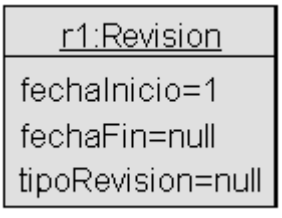


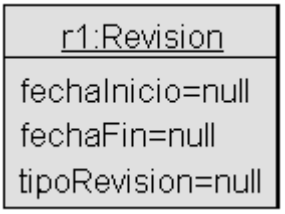
Revision::fechasRevisionValidas	false

En este caso, se puede observar que fechaInicio es mayor que fechaFin por lo que no se cumple el invariante.

- Explicación Caso 4 (se incumple el invariante):







Revision::fechasRevisionValidas	false

En este caso, se puede observar que fechaInicio es nula y fechaFin es nula, entonces se incumple el invariante.

4. **“fechaRevisionPosteriorCoche”**: las revisiones deben realizarse después de la matriculación del coche.

- Restricción:

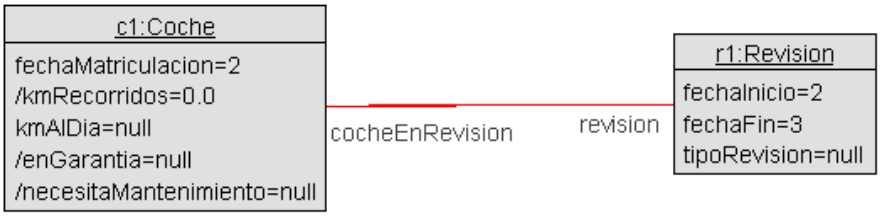
```
inv fechaRevisionPosteriorCoche:
        self.cocheEnRevision.fechaMatriculacion <= self.fechaInicio
```

- Comprobación:

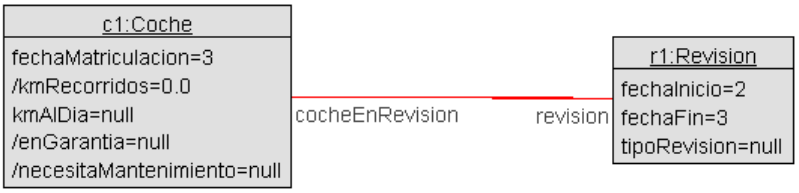
```
!new Coche('c1')
```

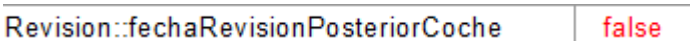
```
!c1.fechaMatriculacion := 2
-- para que el invariante no se cumpla se pone fechaMatriculacion
-- mayor que la fechaInicio de la revision

!new Revision('r1')
!r1.fechaInicio := 2
!r1.fechaFin := 3


!insert (c1,r1) into RevisionCoche
```

- Explicación Caso 1 (se cumple el invariante):



- Explicación Caso 2 (se incumple el invariante):





5. **“garantiaValida”**: el atributo tiene un formato válido(positivo y no nulo).

- Restricción:

```
   inv garantiaValida:
        self.tiempoGarantiaDias <> null and self.tiempoGarantiaDias > 0
```

- Comprobación:

```
!new TallerOficial ('to1')
!to1.tiempoGarantiaDias := 10

!new TallerOficial ('to2')
!to2.tiempoGarantiaDias := -1
```

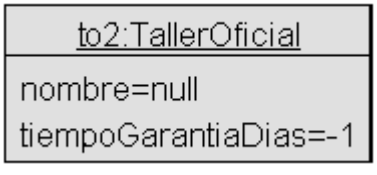
```
!new TallerOficial ('to3')
```

- Explicación Caso 1 (se cumple el invariante):

TallerOficial::garantiaPositiva	true

Como podemos observar al colocar un valor positivo, el invariante se cumple.

- Explicación Caso 2 (se incumple el invariante):



TallerOficial::garantiaPositiva	false

Al colocar el valor negativo en el campo de tiempo de garantía, comprobamos que el invariante se incumple. Asimismo, cuando no añadimos ningún valor al campo el invariante se incumple.

6. **"tallerMismaCiudad"**: si el coche se encuentra en una revisión en el momento actual, esto significa que el taller y el coche se encuentran en la misma ciudad. Si el coche no se encuentra actualmente en una revisión el invariante dará true.

- Restricción:

```
    inv tallerMismaCiudad:
                let revisiones: Sequence(Revision) = self.revision->
sortedBy(fechaFin) -> asSequence() in
        let hoy : Integer = self.clock.now in
        revisiones->forAll(r |
            (r.fechaInicio <= hoy and hoy <= r.fechaFin) implies
            r.taller.localizacionTaller = self.ciudad
        )
```

- Comprobación:

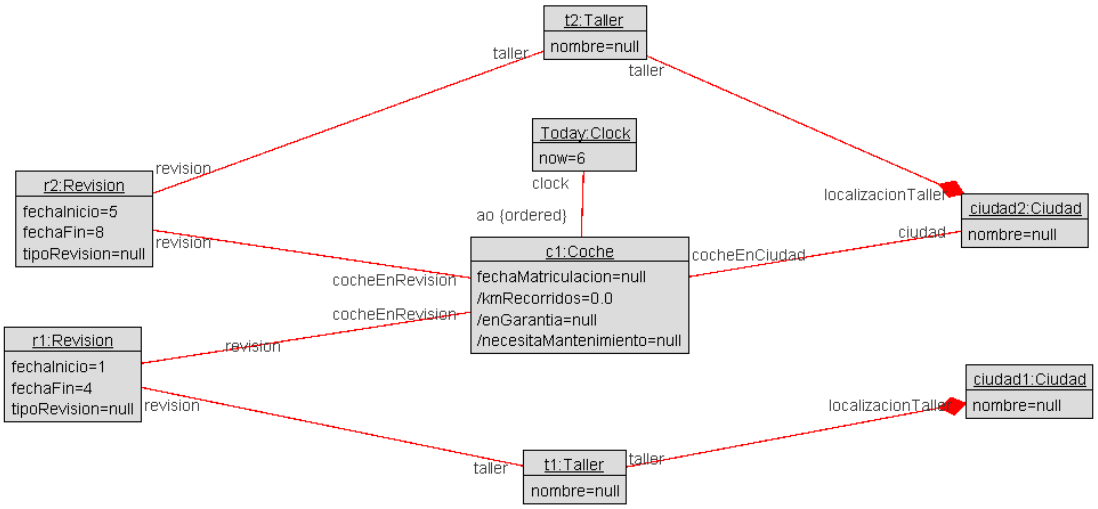
```
!new Clock ('Today')
!Today.now := 6

!new Coche ('c1')

!new Ciudad('ciudad1')
```

```
!new Ciudad('ciudad2')

!new Taller ('t1')

!new Revision ('r1')
!r1.fechaInicio:= 1
!r1.fechaFin := 4

!new Revision('r2')
!r2.fechaInicio:= 5
!r2.fechaFin := 8


!insert (c1,r2) into RevisionCoche
!insert (c1,r1) into RevisionCoche


!insert (t1,r1) into RevisionTaller
!insert (t1,r2) into RevisionTaller


!insert (c1,ciudad1) into CocheCiudad


!insert (ciudad1,t1) into CiudadTaller
-- para que no se cumpla el invariante cambiar ciudad1 por ciudad2
```

Como podemos observar, el coche “c1” tiene dos revisiones siendo “r1” la última revisión que realizó. Además de que en el tiempo del sistema,este coche se encuentra realizando dicha revisión en el taller t1.

Para la comprobación de dicho invariante procederemos a comentar una de las dos últimas líneas del código.

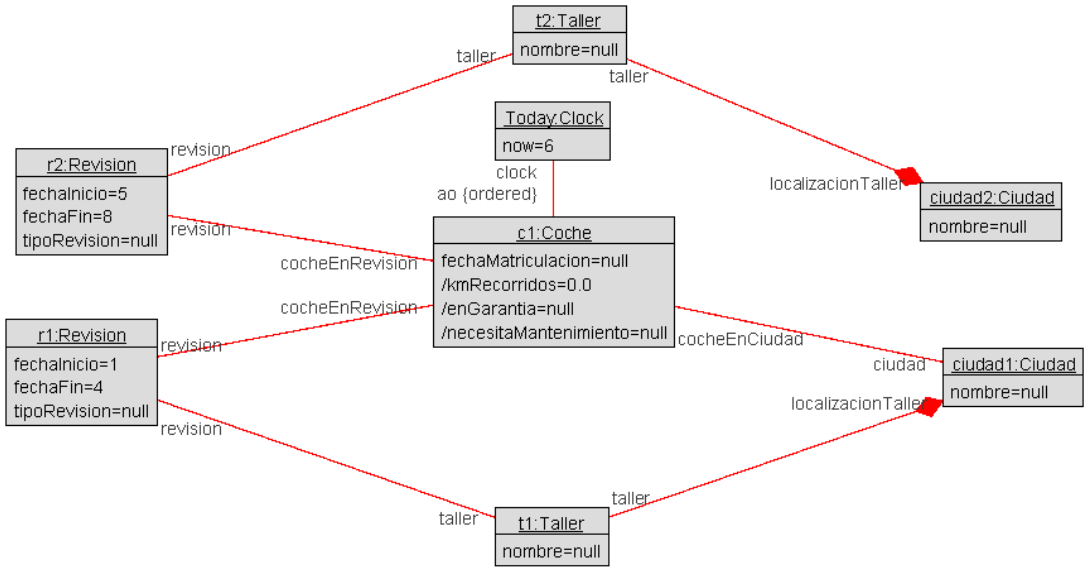
- Explicación Caso 1 (se cumple el invariante):



Coche::tallerMismaCiudad	true

En este caso, el coche tiene dos revisiones en diferentes ciudades, como nos encontramos en el día 6 y se está llevando a cabo la revisión 2 en la ciudad 2, el coche tiene que estar en la ciudad 2, por lo que se cumple el invariante.

Explicación Caso 2 (se incumple el invariante):



Coche::tallerMismaCiudad	false

En este caso, nos encontramos en el día 6 en el que el coche debería estar en una revisión en la ciudad 2 pero coche está en la ciudad 1 por lo que no se cumple el invariante.

7. **“unicoClock”**: en el apartado A del documento nos indica que hay una única instancia de clock para ello lo aseguraremos con la siguiente invariante.

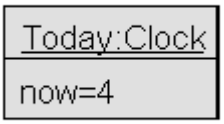
- Restricción:

```
inv unicoClock:
        Clock.allInstances()->size() = 1
```

- Comprobación:

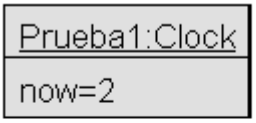
```
!new Clock ('Today')
!Today.now := 4

-- !new Clock ('Prueba1')
-- !Prueba1.now := 2
```

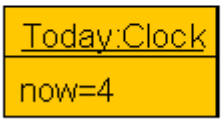
- Explicación Caso 1 (se cumple el invariante):



En este caso, solo creamos un único clock en el diagrama de objetos. Como resultado, el invariante se cumple.

- Explicación Caso 2 (se incumple el invariante):





En este caso, creamos dos instancias de clock. Como resultado, el invariante se incumple.

8. **“ciudadDestinoEsCiudadOrigenSiguiente”**: este invariante comprueba que la ciudad destino de un viaje de un coche es la ciudad origen del siguiente viaje de ese coche ordenando los viajes según su fecha de salida.

- Restricción:

```
inv ciudadDestinoEsCiudadOrigenSiguiente:
```

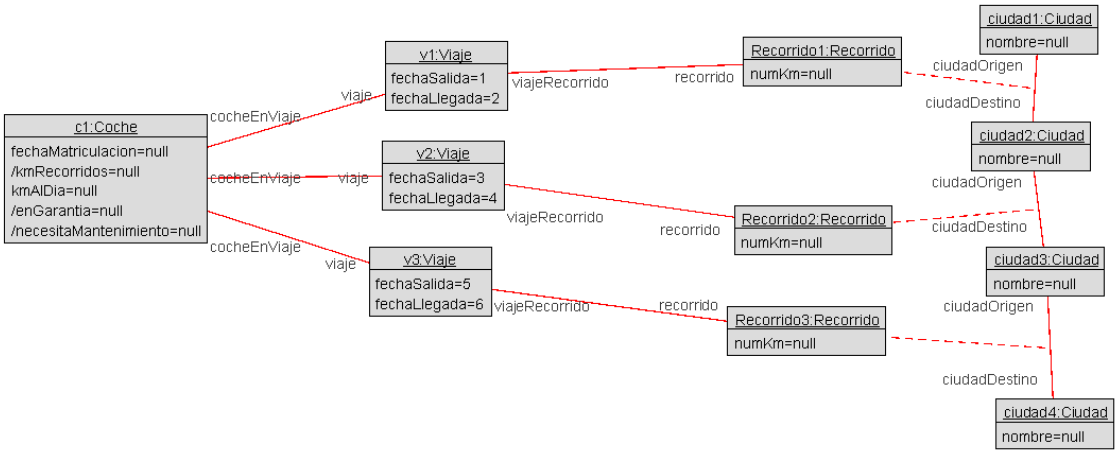
```
        let viajesOrdenados : Sequence(Viaje) = self.viaje
                ->sortedBy(fechaSalida)->asSequence() in
        viajesOrdenados->forAll(v |
                  viajesOrdenados->indexOf(v)  <  viajesOrdenados->size()
implies
            v.recorrido.ciudadDestino = viajesOrdenados
                                        ->at(viajesOrdenados->indexOf(v)  +
1).recorrido.ciudadOrigen
        )
```

- Comprobación:

```
!new Coche('c1')

!new Ciudad('ciudad1')
!new Ciudad('ciudad2')
!new Ciudad('ciudad3')
!new Ciudad('ciudad4')

!new Viaje('v1')
!v1.fechaSalida := 1
!v1.fechaLlegada := 2

!insert(c1,v1) into ViajeCoche

!insert(ciudad1,ciudad2) into Recorrido

!insert(v1,Recorrido1) into ViajeRecorrido



!new Viaje('v2')
!v2.fechaSalida := 3
!v2.fechaLlegada := 4

!insert(c1,v2) into ViajeCoche
!insert (ciudad2,ciudad3) into Recorrido

!insert(v2,Recorrido2) into ViajeRecorrido

!new Viaje('v3')
!v3.fechaSalida := 5
!v3.fechaLlegada := 6
```

```
!insert(c1,v3) into ViajeCoche

!insert (ciudad3,ciudad4) into Recorrido

!insert(v3,Recorrido3) into ViajeRecorrido
```

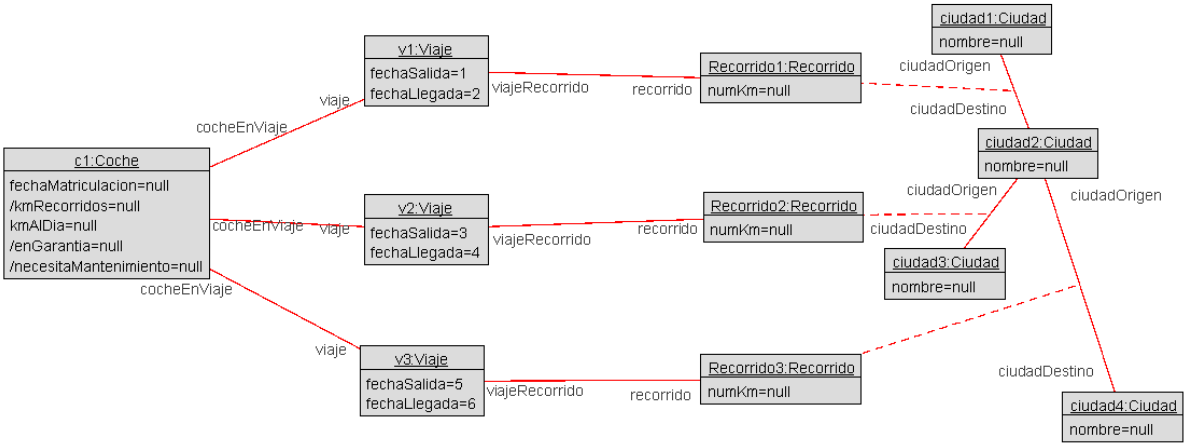
- Explicación Caso 1 (se cumple el invariante):



Coche::ciudadDestinoEsCiudadOrigenSiguiente	true

En este caso, el recorrido 1 tiene de ciudad origen ‘ciudad1’ y de ciudad destino ‘ciudad2’ y el recorrido 2 tiene de ciudad origen ‘ciudad2’ y de ciudad destino ‘ciudad3’ por lo tanto el invariante se cumple.

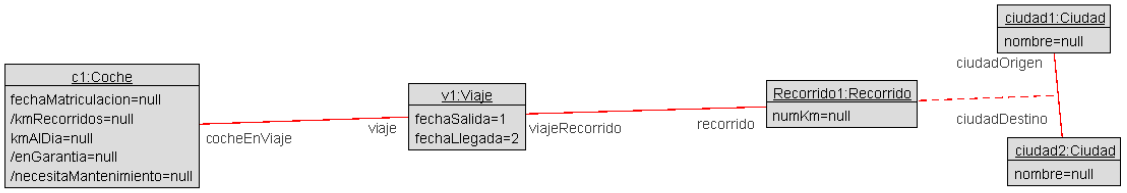
- Explicación Caso 2 (se incumple el invariante):



Coche::ciudadDestinoEsCiudadOrigenSiguiente	false

En este caso, he cambiado la ciudad origen del Recorrido3 de ciudad3 a ciudad2 lo que hace que la restricción no se cumpla.

- Explicación Caso 3 (solo hay un viaje):



Coche::ciudadDestinoEsCiudadOrigenSiguiente	true

En este caso también también se cumple el invariante dado la lógica del código

9. **"viajeDetrasDeOtro"**: este invariante comprueba que la fecha de salida de un viaje de un coche sea mayor o igual que la fecha de llegada de su viaje anterior

- Restricción:

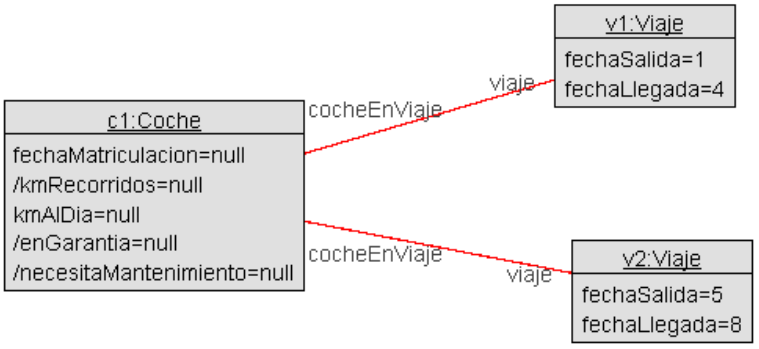
```
inv viajeDetrasDeOtro:
        let viajesOrdenados : Sequence(Viaje) = self.viaje
            ->sortedBy(fechaSalida)->asSequence() in
        viajesOrdenados->forAll(v |
                viajesOrdenados->indexOf(v)  <  viajesOrdenados->size()
implies
            v.fechaLlegada <= viajesOrdenados
                    ->at(viajesOrdenados->indexOf(v) + 1).fechaSalida
        )
```

- Comprobación:

```
!new Coche('c1')

!new Viaje('v1')
!v1.fechaSalida := 1
!v1.fechaLlegada := 4

!insert(c1,v1) into ViajeCoche



!new Viaje('v2')
!v2.fechaSalida := 5
!v2.fechaLlegada := 8

-- para que no se cumpla el invariante, se cambia cualquier
-- fecha de v2 por uno que este dentro del rango [v1.fechaSalida,
v1.fechaLlegada]
```

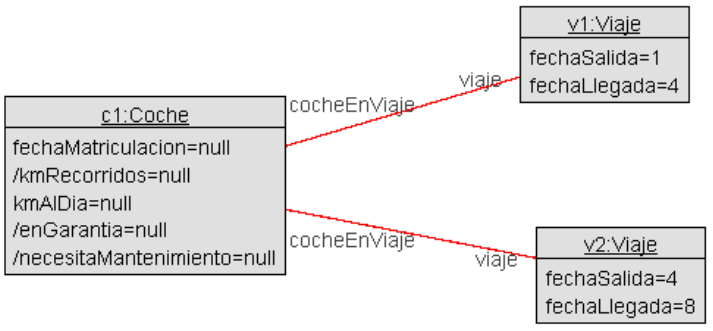
```
!insert(c1,v2) into ViajeCoche

check
```

- Explicación Caso 1 (se cumple el invariante):



Coche::viajeDetrasDeOtro	true

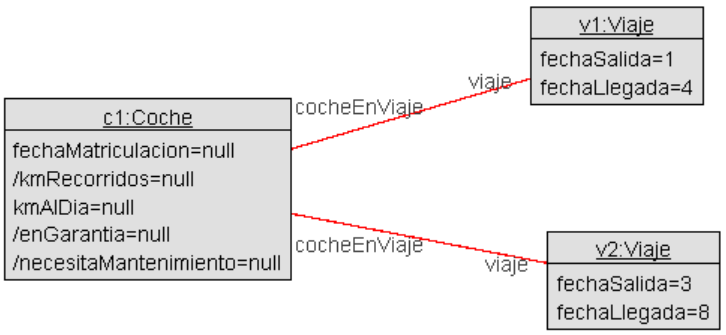
En este caso, el viaje 2 tiene una fecha de salida mayor que la fecha de llegada del viaje 1, por lo que el invariante se cumple.

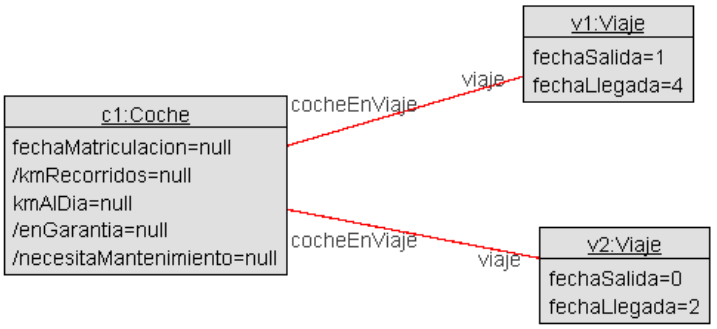


Coche::viajeDetrasDeOtro	true

En este caso, la fecha de salida del viaje 2 es igual que la fecha de llegada del viaje 1, por lo que el invariante se cumple.

- Explicación Caso 3 (se incumple el invariante):





Coche::viajeDetrasDeOtro	false

Podemos comprobar que cuando la fecha de llegada de un viaje es mayor a la fecha de salida del siguiente, entonces el invariante no se cumple.

10. **"solo1TallerOficial"**: una ciudad solo puede tener un taller oficial.

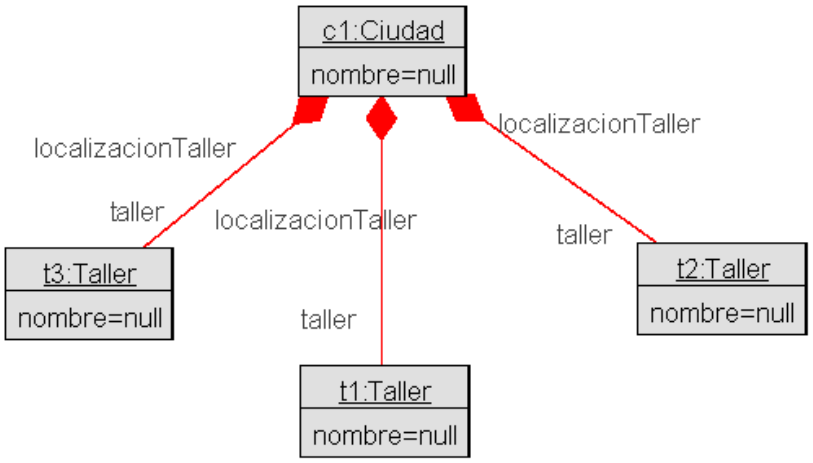
- Restricción:

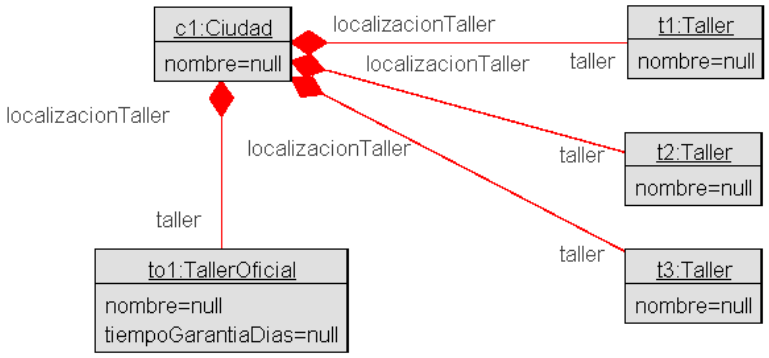
```
inv solo1TallerOficial:
        self.taller->select(t|t.oclIsTypeOf(TallerOficial))->size()<=1
```

- Comprobación:

```
-- añadimos dos talleres oficiales a una ciudad

!new TallerOficial('to1')
!new TallerOficial('to2')

!new Taller('t1')
!new Taller('t2')
!new Taller('t3')

!new Ciudad('c1')

--!insert (c1, to1) into CiudadTaller
--!insert (c1, to2) into CiudadTaller
-- si comentamos alguna de estas dos líneas devuelve true

!insert (c1, t1) into CiudadTaller
!insert (c1, t2) into CiudadTaller
!insert (c1, t3) into CiudadTaller
-- los talleres normales no deberían afectar

check
```

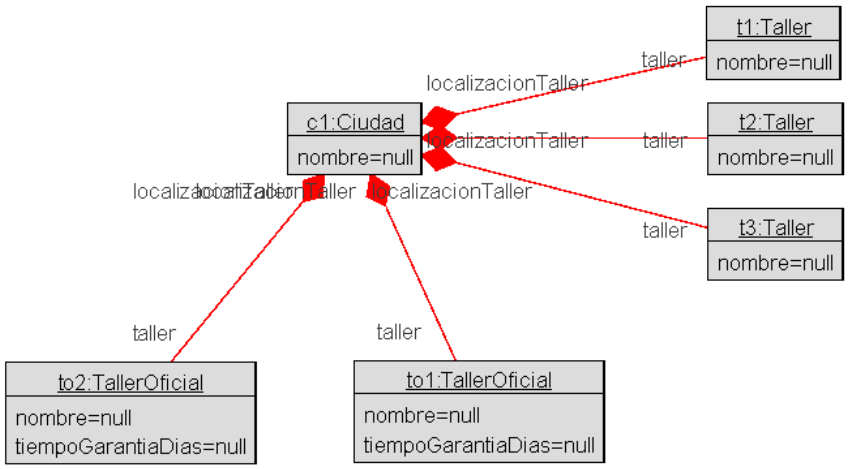
- Explicación Caso 1 (se cumple el invariante):





Comprobamos que al no asociar talleres oficiales o asociar un taller oficial el invariante se cumple.

- Explicación Caso 2 (se incumple el invariante):



Al añadir más de un taller oficial a una ciudad comprobamos que el invariante se incumple.

11. **“cocheViajandoOEnCiudad”**: un coche si se encuentra viajando no puede estar en ninguna ciudad y viceversa y también un coche debe de encontrarse en la ciudad de

destino del último viaje o en la ciudad inicial si no ha realizado ningún viaje.

- Restricción:

```
inv cocheViajandoOEnCiudad:
                                let    viajes    :    Sequence(Viaje)    =
self.viaje->sortedBy(fechaSalida)->asSequence() in
        let hoy : Integer = self.clock.now in
        if self.viaje->isEmpty() then
                    -- si el coche no se encuentra en ningun viaje y no
tiene ningun viaje pendiente
                -- la ciudad del coche tiene que estar definida
                not self.ciudad.oclIsUndefined()
        else
                if viajes->exists(v | v.fechaSalida <= hoy and hoy <=
v.fechaLlegada) then
                    -- Si el coche está en un viaje, su ciudad debe ser
indefinida
                self.ciudad.oclIsUndefined()
            else
                            if   not  viajes->select(v  |  v.fechaSalida  >=
hoy)->isEmpty() then
                        -- Si el coche no tiene viajes realizados pero
tiene viajes pendientes,
                    -- su ciudad es la ciudad origen del primer viaje
                                    viajes->select(v  |  v.fechaSalida  >=
hoy)->first().recorrido.ciudadOrigen = self.ciudad
                else
                        -- Si tiene viajes realizados, su ciudad debe
coincidir con el destino del último viaje finalizado
                                   viajes->select(v  |  v.fechaLlegada  <=
hoy)->last().recorrido.ciudadDestino = self.ciudad

                endif
            endif
        endif
```

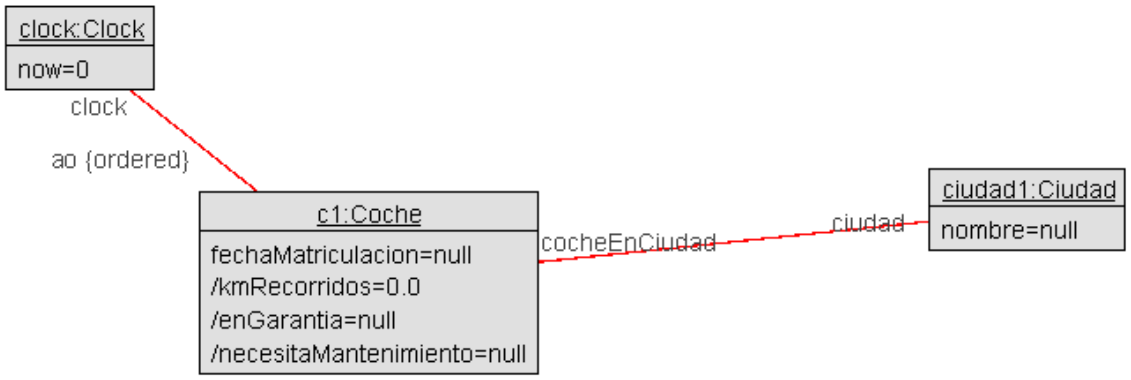
- Comprobación:

```
--En este archivo se comprobará el invariante cocheViajandoOEnCiudad


--Creamos un reloj para controlar el tiempo
```

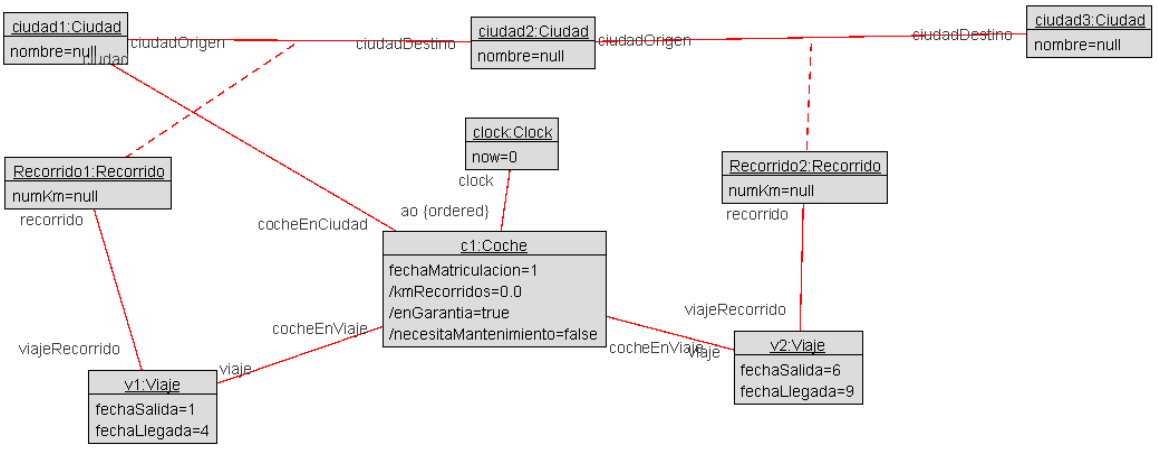
```
!new Clock ('clock')
!clock.now := 0
--Para que se incumpla el invariante modificamos el dia actual a uno
dentro del intervalo
--del viaje v1


!new Coche ('c1')

!insert (clock,c1) into Time


!new Ciudad ('ciudad1')

!new Ciudad ('ciudad2')

!new Ciudad ('ciudad3')



!insert (ciudad1,ciudad2) into Recorrido

!insert (ciudad2,ciudad3) into Recorrido


!new Viaje ('v1')
!v1.fechaSalida := 1
!v1.fechaLlegada := 4

!new Viaje ('v2')
!v2.fechaSalida := 6
!v2.fechaLlegada := 9

!insert (v1,Recorrido1) into ViajeRecorrido

!insert (v2,Recorrido2) into ViajeRecorrido


!insert (c1,v1) into ViajeCoche
!insert (c1,v2) into ViajeCoche
```

```
!insert (c1,ciudad1) into CocheCiudad
```

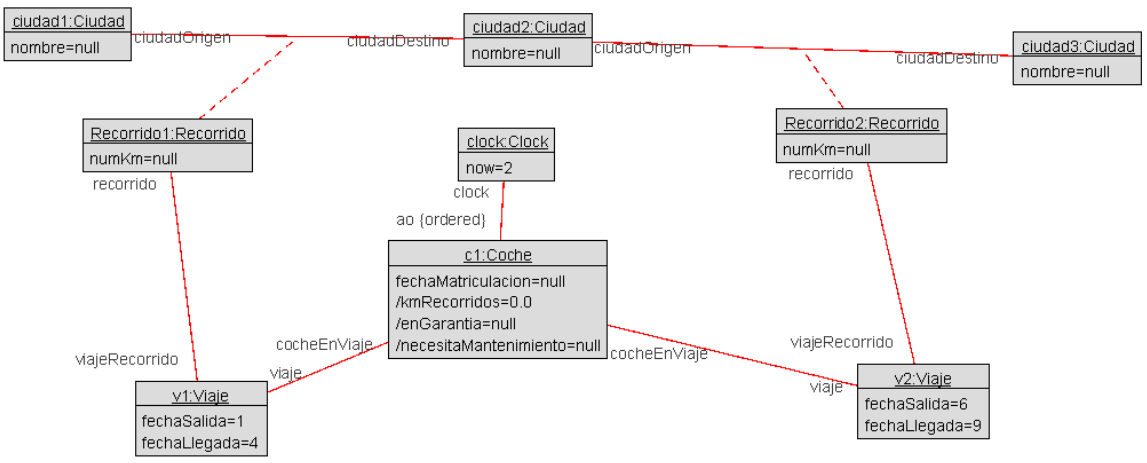
- Explicación Caso 1 (se cumple el invariante):



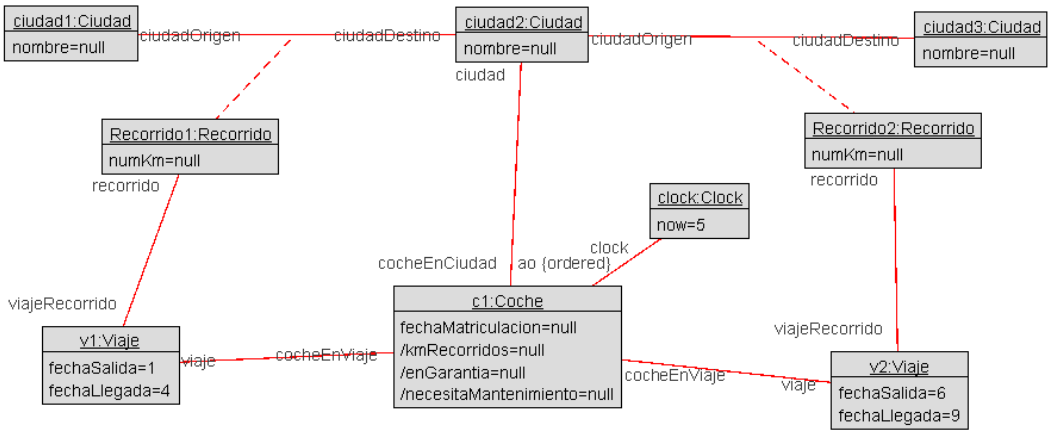
En este caso, el coche no tiene ningún viaje por lo que debería estar en una ciudad, entonces se cumple el invariante.



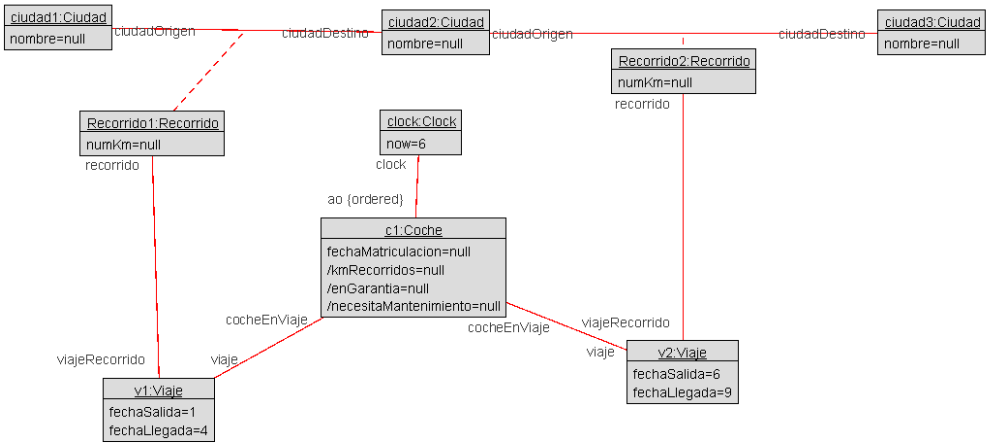
En este caso, now vale 0, v1 empieza el día 1, entonces el coche debe estar en la ciudad origen del v1, por lo que se cumple el invariante.



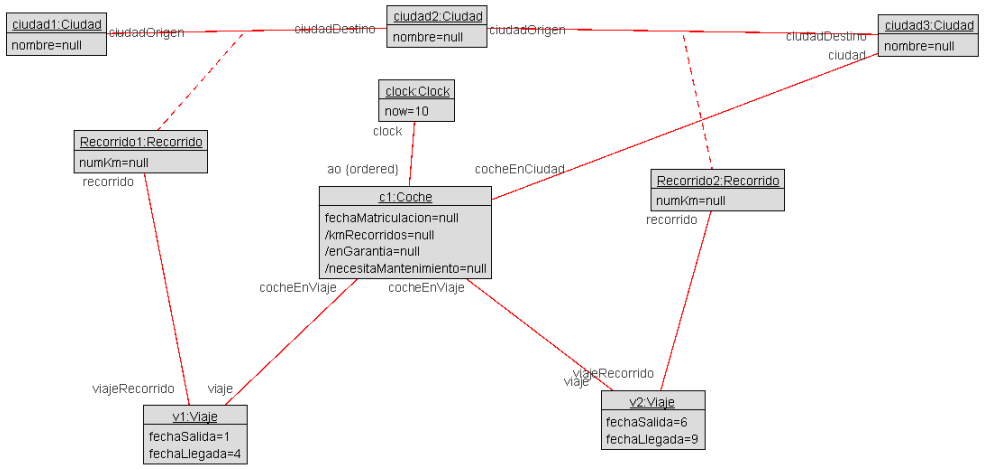
En este caso, now vale 2, por lo que se encuentra viajando en el v1 por lo que coche no está en ninguna ciudad por lo que se cumple el invariante.



En este caso, now vale 5, por lo que se encuentra entre v1 y v2 y como v1 tiene fecha de llegada 4, el coche debe encontrarse en la ciudad destino del v1, por lo que el invariante se cumple.



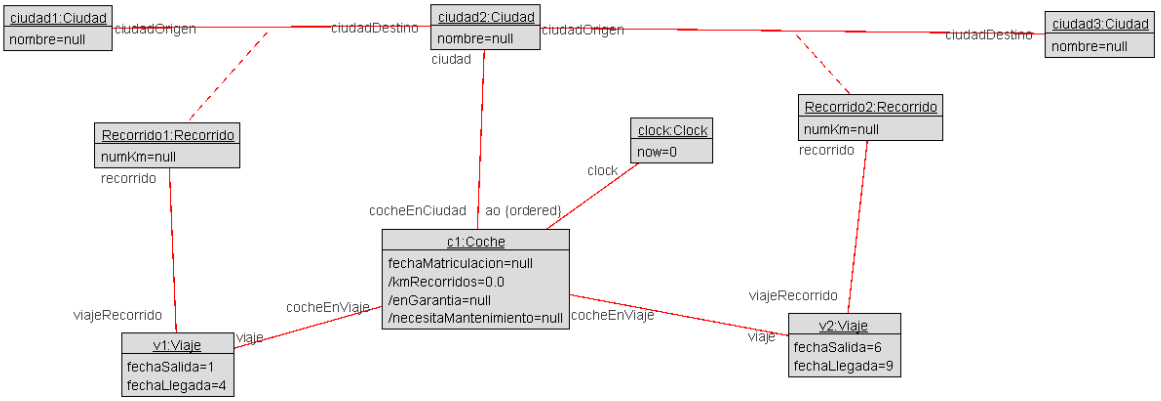
En este caso, now vale 6 y coche se encuentra viajando en el v2, por lo que coche no está en ninguna ciudad, entonces se cumple el invariante.



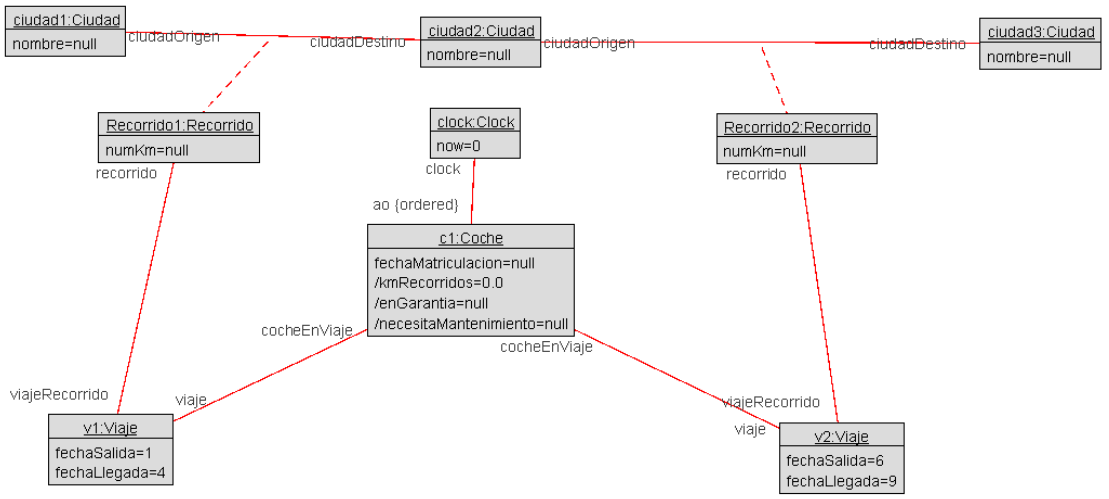
En este caso, now vale 10 y como la fecha de llegada del v2 es 9 entonces el coche se encuentra en la ciudad destino del v2, por lo que el invariante se cumple.

Coche::cocheViajandoOEnCiudad	true

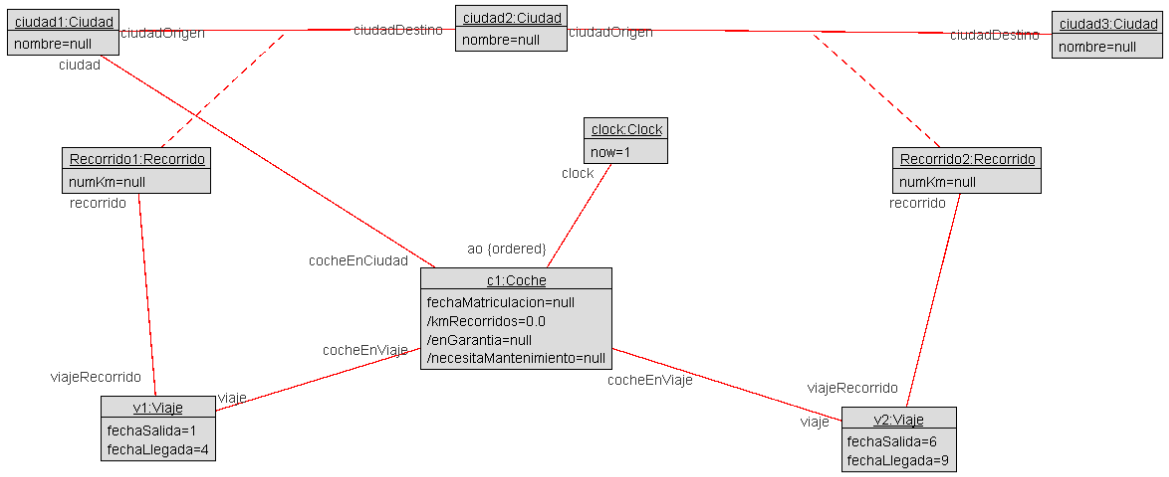
- Explicación Caso 2 (se incumple el invariante):



En este caso, now vale 0, v1 empieza el dia 1, entonces el coche debe estar en la ciudad origen del v1, pero se encuentra en la ciudad 2 por lo que no se cumple el invariante



En este caso, now vale 0, v1 empieza el dia 1, entonces el coche debe estar en la ciudad origen del v1, pero no se encuentra en ninguna ciudad, por lo que no se cumple el invariante



En este caso, el valor de "now" es 1, lo que indica que se encuentra viajando en el v1. Por lo tanto, el coche no debería estar en ninguna ciudad, sin embargo, se encuentra en ciudad1, lo que significa que el invariante no se cumple.

Coche::cocheViajandoOEnCiudad	false

Como podemos observar en el diagrama de objetos, si el momento actual coincide con el intervalo de un viaje, el invariante se incumple ya que el coche tiene una asociación con ciudad, indicando que se encuentra en la misma cuando no debería.

12. **“unaRevisionALaVez”**: no pueden coincidir las fechas de las revisiones del coche.

- Restricción:

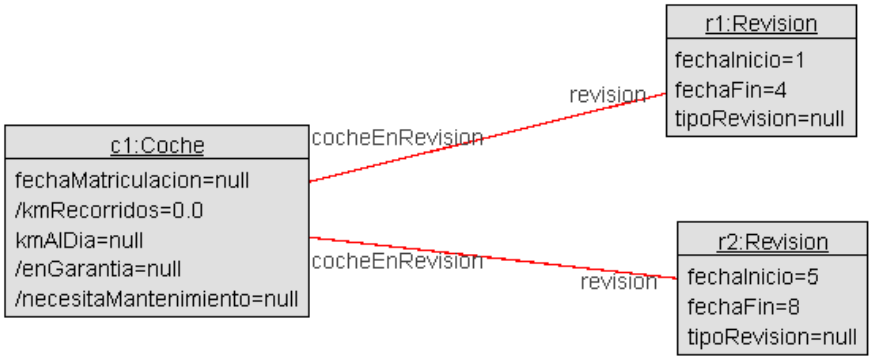
```
inv unaRevisionALaVez:
        self.revision->forAll(r1,r2 | r1 <> r2 implies
                (r1.fechaInicio <> r2.fechaInicio and r1.fechaFin <>
r2.fechaFin) and
```

```
            not (r1.fechaInicio <= r2.fechaInicio and r2.fechaInicio <=
r1.fechaFin) and
                not (r1.fechaInicio <= r2.fechaFin and r2.fechaFin <=
r1.fechaFin)
        )
```

- Comprobación:

```
!new Revision ('r1')
!r1.fechaInicio := 1
!r1.fechaFin := 4

!new Revision ('r2')
!r2.fechaInicio := 2
!r2.fechaFin := 3

!new Coche('c1')

!insert (c1, r1) into RevisionCoche

!insert (c1, r2) into RevisionCoche
```

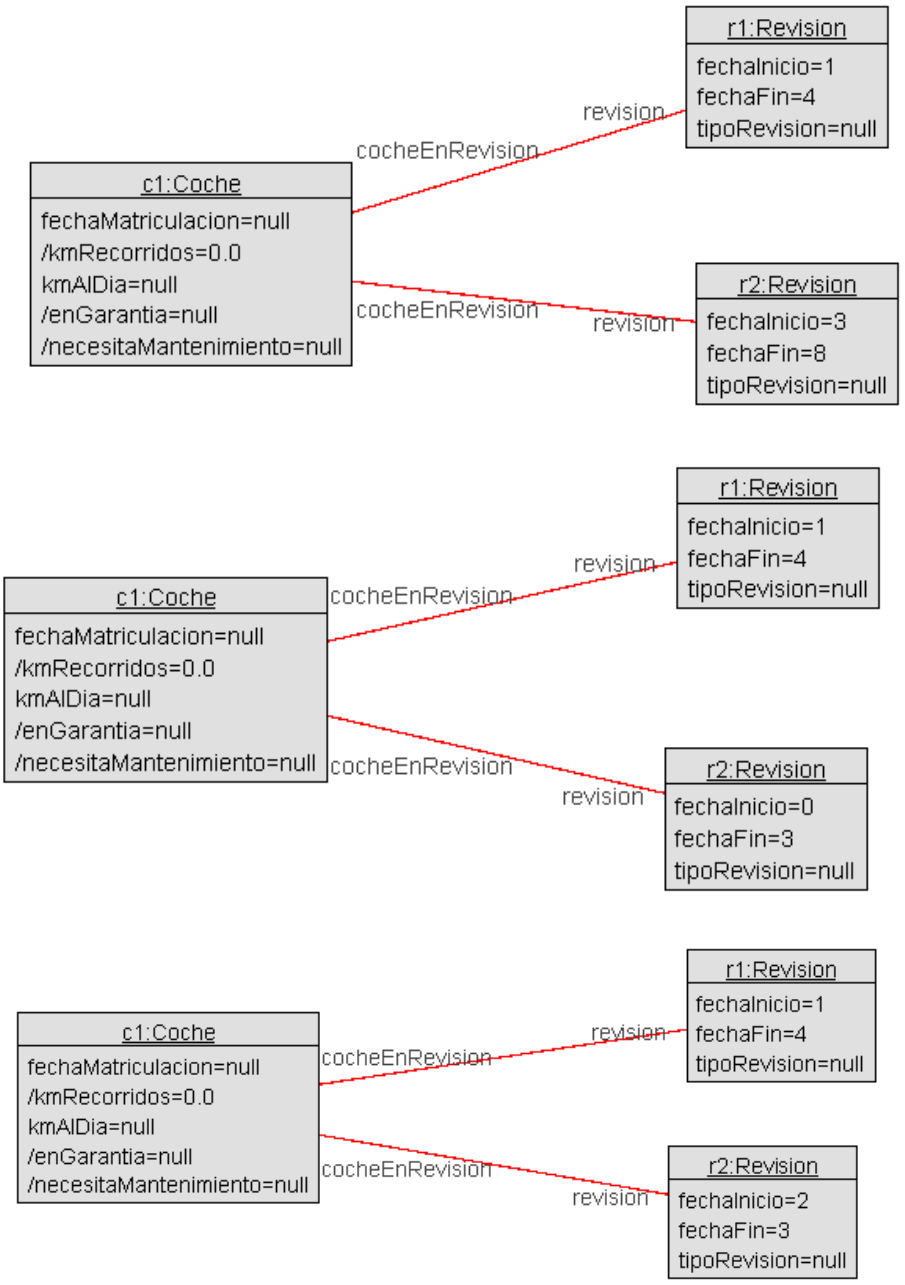
- Explicación Caso 1 (se cumple el invariante):



Coche::unaRevisionALaVez	true

Como podemos observar, fechaInicio y fechaFin de r2 no coinciden con el intervalo de fechaInicio y fechaFin de r1.

- Explicación Caso 2 (se incumple el invariante):



En todos estos casos en los que la fechaInicio o la fechaFin ambas de r2 coinciden con el intervalo de fechaInicio y fechaFin de r1 el invariante no se cumple.

13. **“fechasViajeValidas”**: la fecha de los viajes debe de tener un formato adecuado (positivo y no nulo) y la fecha de salida debe ser anterior a la de llegada.

- Restricción:

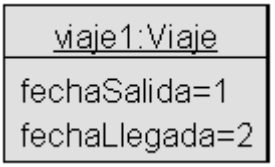
```
context Viaje
    inv fechasViajeValidas:
        ((self.fechaSalida <> null and self.fechaSalida>=0) and
        (self.fechaLlegada <> null and self.fechaLlegada>=0)) and
        (self.fechaSalida <= self.fechaLlegada)
```

- Comprobación:

```
reset
!new Viaje('viaje1')
!viaje1.fechaSalida := 1
!viaje1.fechaLlegada := 2
check
```

Se modifica la fecha de salida para comprobar los diferentes casos.

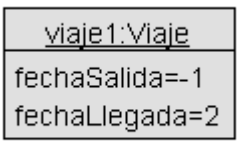
- Explicación Caso 1 (se cumple el invariante):



Viaje::fechasViajeValidas | true

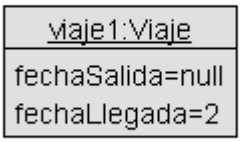
Como podemos comprobar, la fecha de salida y llegada están asignadas y son positivas. Además, la fecha de salida es anterior a la de llegada, por tanto, se cumple el invariante.

- Explicación Caso 2 (se incumple el invariante):



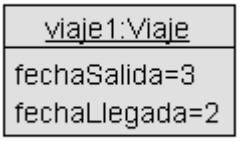
Viaje::fechasViajeValidas | false

La fecha de salida es negativa por eso se incumple el invariante.



Viaje::fechasViajeValidas	false

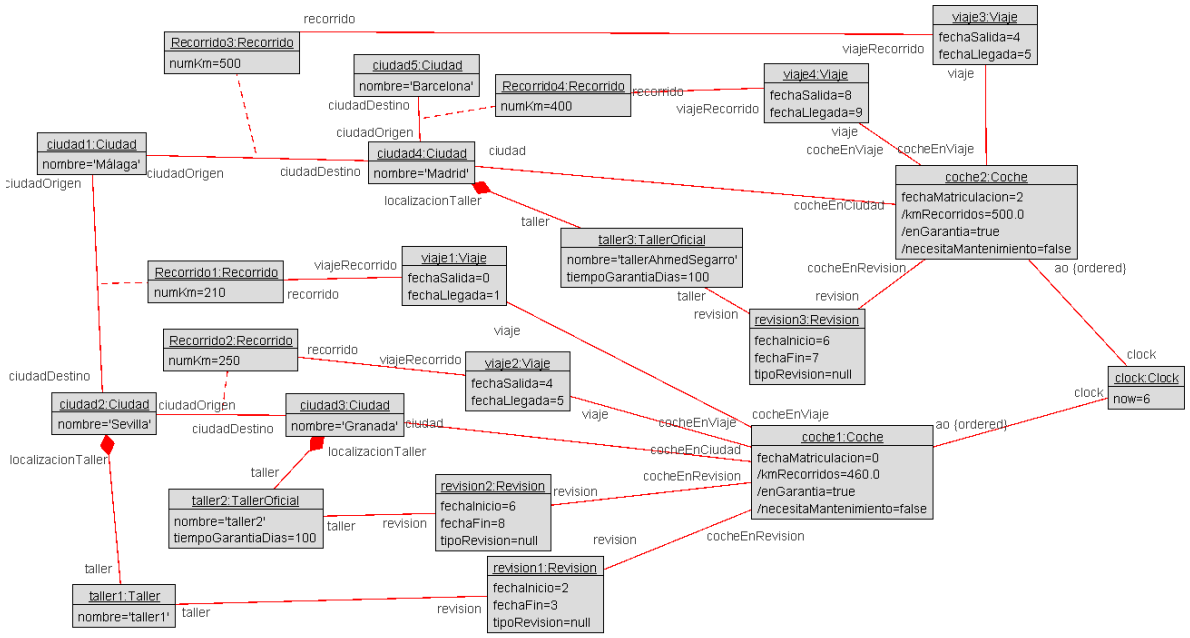
La fecha de salida no ha sido asignada y es nula, por tanto, se incumple el invariante.



Viaje::fechasViajeValidas	false

La fecha de salida es posterior a la de llegada, por tanto, se incumple el invariante.

Modelo conceptual general del apartado A



Código .soil:

```
--El sistema parte del instante 0
!new Clock ('clock')
!clock.now := 6

!new Coche('coche1')
!coche1.fechaMatriculacion := 0

!new Coche('coche2')
!coche2.fechaMatriculacion := 2

!insert (clock, coche1) into Time
!insert (clock, coche2) into Time

!new Ciudad ('ciudad1')
!ciudad1.nombre := 'Málaga'

!new Ciudad ('ciudad2')
!ciudad2.nombre := 'Sevilla'

!new Ciudad ('ciudad3')
!ciudad3.nombre := 'Granada'

!new Ciudad('ciudad4')
```

```
!ciudad4.nombre := 'Madrid'

!new Ciudad('ciudad5')
!ciudad5.nombre := 'Barcelona'

--El coche está inicialmente en Málaga
!insert (coche1,ciudad3) into CocheCiudad
!insert (coche2,ciudad4) into CocheCiudad

--Recorrido entre malaga - sevilla 210 km y sevilla - granada 250 km
!insert(ciudad1, ciudad2) into Recorrido
!Recorrido1.numKm := 210

!insert(ciudad2, ciudad3) into Recorrido
!Recorrido2.numKm := 250

!insert(ciudad1, ciudad4) into Recorrido
!Recorrido3.numKm := 500

!insert(ciudad4, ciudad5) into Recorrido
!Recorrido4.numKm := 400

--viaje 1
!new Viaje ('viaje1')
!viaje1.fechaSalida := 0
!viaje1.fechaLlegada := 1

!new Viaje ('viaje2')
!viaje2.fechaSalida := 4
!viaje2.fechaLlegada := 5

!new Viaje ('viaje3')
!viaje3.fechaSalida := 4
!viaje3.fechaLlegada := 5

!new Viaje ('viaje4')
!viaje4.fechaSalida := 8
!viaje4.fechaLlegada := 9


!insert (coche1, viaje1) into ViajeCoche
!insert (coche1, viaje2) into ViajeCoche
!insert (coche2, viaje3) into ViajeCoche
```

```
!insert (coche2, viaje4) into ViajeCoche

!insert (viaje1, Recorrido1) into ViajeRecorrido
!insert (viaje2, Recorrido2) into ViajeRecorrido
!insert (viaje3, Recorrido3) into ViajeRecorrido
!insert (viaje4, Recorrido4) into ViajeRecorrido

!new Taller ('taller1')
!taller1.nombre := 'taller1'

!new TallerOficial ('taller2')
!taller2.nombre := 'taller2'
!taller2.tiempoGarantiaDias := 100

!new TallerOficial ('taller3')
!taller3.nombre := 'tallerAhmedSegarro'
!taller3.tiempoGarantiaDias := 100

!new Revision ('revision1')
!revision1.fechaInicio := 2
!revision1.fechaFin := 3

!new Revision ('revision2')
!revision2.fechaInicio := 6
!revision2.fechaFin := 8

!new Revision ('revision3')
!revision3.fechaInicio := 6
!revision3.fechaFin := 7

!insert (taller1, revision1) into RevisionTaller
!insert (taller2, revision2) into RevisionTaller
!insert (taller3, revision3) into RevisionTaller


!insert (coche1, revision1) into RevisionCoche
!insert (coche1, revision2) into RevisionCoche
!insert (coche2, revision3) into RevisionCoche

!insert (ciudad2, taller1) into CiudadTaller
!insert(ciudad3, taller2) into CiudadTaller
!insert(ciudad4, taller3) into CiudadTaller
```

ACLARACIÓN:

Se ha comprobado que al hacer check en la consola para todos los .soil no se viola ninguna multiplicidad ni invariante.

Parte 2: Modelado del Comportamiento del Sistema

En esta sección del documento se modelará el sistema en función del paso del tiempo. En particular, se representará exclusivamente el comportamiento de los coches durante sus trayectos.

Explicaciones de las Operaciones

En el siguiente apartado, se procederá a explicar las operaciones añadidas al código USE.

- **tick() (Clock, ActiveObject y Coche)**: esta operación diseña el paso del tiempo del sistema, donde aumentará el atributo de la clase "Clock", "now", con el valor que tenga registrado en el atributo "resolution".

```
    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
```

  TimePasses: esta postcondición comprueba que el tiempo haya pasado como se espera

- **run(n:Integer) (Clock)**: esta operación ejecuta la función anterior un número n de veces.

```
    run (n:Integer)
      begin
        for i in Sequence{1..n} do
          self.tick()
        end
      end
```

- **nuevoViaje(recorrido: Recorrido) (Coche)** : esta operación crea un nuevo viaje y lo asocia al conjunto de viajes del coche. Además de asociar un recorrido al propio viaje.

```
    nuevoViaje(recorrido : Recorrido)
        begin
            declare v:Viaje;

            v:= new Viaje;
            v.fechaSalida:=self.clock.now;
            v.kilometrosRecorridos:= 0;
            insert(self, v) into ViajeCoche;
            insert(v, recorrido) into ViajeRecorrido;
            delete (self, self.ciudad) from CocheCiudad;

        end
        --Añadir precondición en el que now >= fecha matriculación
        pre OrigenRecorridoCorrecto
:(self.fechaMatriculacion<=clock.now and not
self.ciudad.oclIsUndefined() and
```

```
Recorrido.allInstances()->exists(r | r = recorrido) and
self.ciudad = recorrido.ciudadOrigen)

        post CreadoViajeConFechaSalidaNow : (self.viaje->exists(v
| v.recorrido = recorrido and v.fechaSalida = self.clock.now))
```

OrigenRecorridoCorrecto: esta precondición comprueba que para iniciar un viaje el coche se encuentre en una ciudad, que exista el recorrido que se pasa por parámetro, que la ciudad origen del recorrido sea la ciudad en la que se encuentra el coche y que la fecha de matriculación sea anterior a now.

CreadoViajeConFechaSalidaNow: se comprueba que se ha creado el viaje que comienza en el momento en el que se ejecuta la función y cuyo recorrido es el pasado por parámetro.

- **avanzar() (Coche)** : esta operación diseña el movimiento del coche, si este se encuentra realizando un viaje en dicho momento. En esta, los atributos de kilómetros recorridos por el coche y los kilómetros recorridos en el viaje, se actualizan (llamada a la función "**sumarKilometrosRecorridos**").
  Si los kilómetros que se avanzan hacen que se complete el total de recorridos del viaje la operación llamará a la función "**asignarFechaLlegada**".

```
    avanzar()
        begin
            declare kmQueRecorreAlAvanzar : Integer, viajeActual :
Viaje, kmRestantesViajeActual : Integer;
            viajeActual := new Viaje;

            if self.viaje -> exists(v |
v.fechaLlegada.oclIsUndefined()) then --Hay  un invariante que
comprueba que no haya más viajes en curso

                viajeActual := self.viaje->sortedBy(v |
v.fechaSalida)->last();
                kmRestantesViajeActual :=
viajeActual.recorrido.numKm - viajeActual.kilometrosRecorridos;

                --Como máximo recorre kmAlDia (↓)
                if kmRestantesViajeActual > self.kmAlDia then
                    kmQueRecorreAlAvanzar := self.kmAlDia;
                else
                    kmQueRecorreAlAvanzar :=
kmRestantesViajeActual;

viajeActual.asignarFechaLlegada(self.clock.now);
```

```
                    insert(self,
viajeActual.recorrido.ciudadDestino) into CocheCiudad;
                end;


viajeActual.sumarKilometrosRecorridos(kmQueRecorreAlAvanzar);
                self.kmRecorridos := self.kmRecorridos +
kmQueRecorreAlAvanzar;

            end;

        end

        post kilometrosActualizados: self.kmRecorridos =
(self.kmRecorridos@pre + self.kmAlDia)
        or (self.kmRecorridos = self.kmRecorridos@pre +
((self.viaje->sortedBy(v | v.fechaSalida)->last().recorrido.numKm
- (self.viaje->sortedBy(v |
v.fechaSalida)->last().kilometrosRecorridos))))
constraints
    inv kmAlDiaPositivo: self.kmAlDia >= 0
end
```

kilometrosActualizados: esta postcondición verifica que, al avanzar un coche en un viaje, los kilómetros recorridos coincidan con la cantidad que debería recorrer en un día completo. Si el viaje concluye antes de alcanzar la distancia diaria total, también se asegura de que los kilómetros recorridos sean los correctos.

- **sumarKilometrosRecorridos(km : Integer) (Viaje)**: esta operación suma un número de kilómetros "km" y los añade al atributo de la clase "Viaje".

```
    sumarKilometrosRecorridos(km : Integer)
        begin
            self.kilometrosRecorridos:= self.kilometrosRecorridos
+ km;
        end
        pre kmRecorridosNoNulo: km <> null
        post kmRecorridosActualizados: self.kilometrosRecorridos =
self.kilometrosRecorridos@pre + km
```

kmRecorridoNoNulo: comprueba que el valor pasado como parámetro no es nulo.

kmRecorridosActualizados: comprueba que el número de kilómetros recorridos del viaje se ha actualizado correctamente.

- **asignarFechaLlegada(llegada : Integer) (Viaje)**: esta operación asigna una fecha de llegada, pasada a la función, al objeto donde se realiza la operación.

```
asignarFechaLlegada(llegada : Integer)
    begin
        self.fechaLlegada := llegada;
    end
    pre fechaLlegadaNoNula: llegada <> null
    post fechaLlegadaActualizada: self.fechaLlegada = llegada
```

fechaLlegadaNoNula: comprueba que la fecha de llegada pasada como parámetro no sea nula.

fechaLlegadaActualizada: comprueba que la fecha de llegada que se asigna a un viaje una vez terminado es la indicada.

Explicaciones de Modificaciones Del Código

En el siguiente apartado se explicarán las modificaciones realizadas al código original para implementar el modelo dinámico.

Atributos nuevos:

- **resolution (Clock):** Añadimos esta variable para controlar de forma dinámica cúanto tiempo pasa cada tick del Clock. Antes de tenerla, el tiempo sólo podía avanzar de 1 en 1

```
    resolution : Integer init = 1
```

- **kmAlDia (Coche):** Esta nueva variable es necesaria porque nos indica la velocidad del coche por dia

```
    kmAlDia : Integer
```

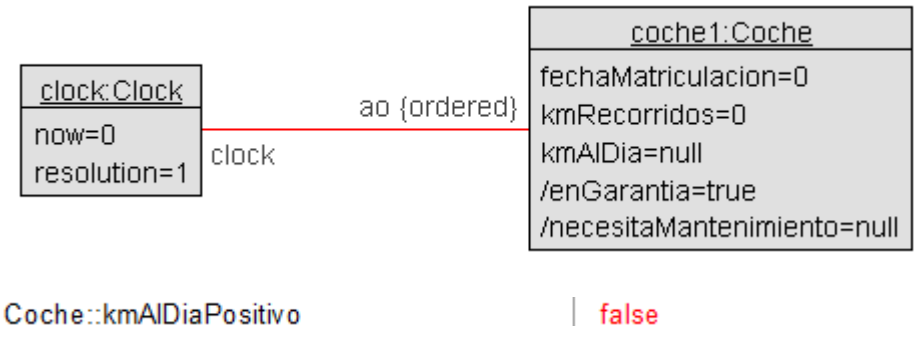
- **kilometrosRecorridos (Viaje):** Añadimos una nueva variable para registrar los kilómetros recorridos del viaje. Esto con el objetivo de poder saber si un viaje ha terminado o no.

```
    kilometrosRecorridos : Integer
```

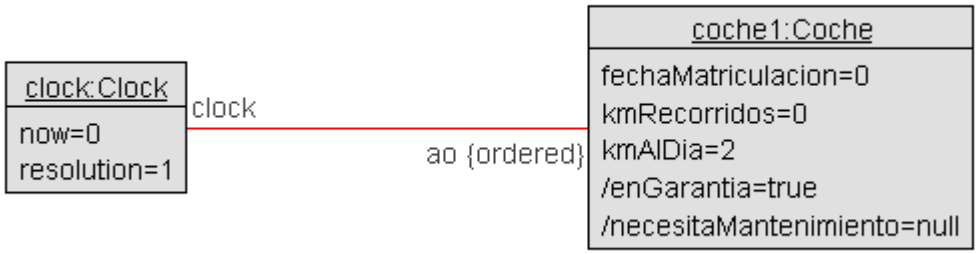
Invariantes nuevas:

- **kmAlDiaPositivo (Coche):** Escribimos esta restricción después de añadir el atributo kmAlDia para comprobar que es positivo y que no haya errores en el sistema

```
    inv kmAlDiaPositivo: self.kmAlDia >= 0
```



En este caso como la variable kmAlDia es null la invariante no se cumple, debe ser un número mayor que 0.



Coche::kmAlDiaPositivo	true

Aquí la variable kmAlDia es positiva, por tanto se cumple la invariante.

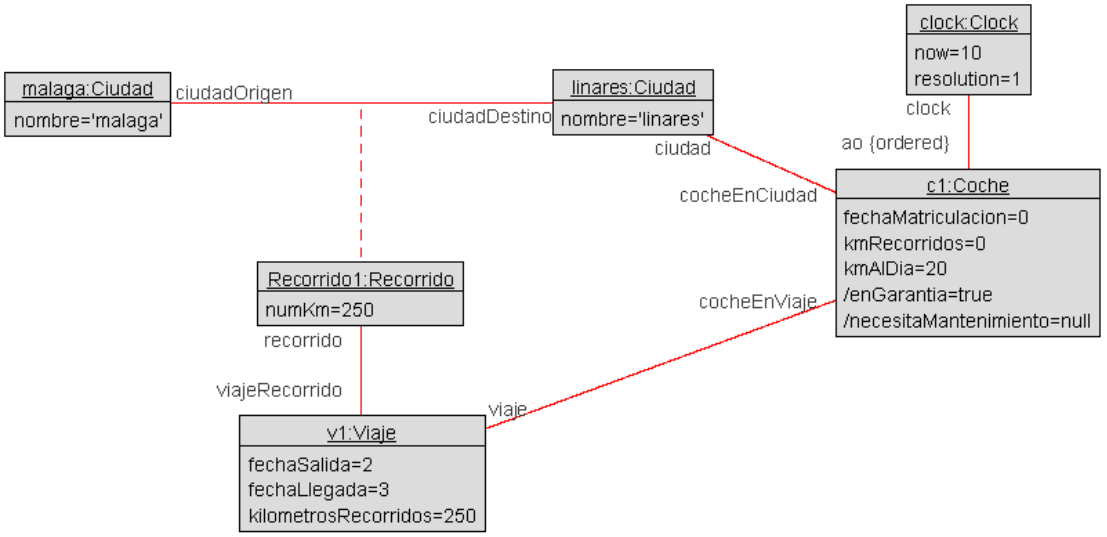
- **kmRecorridosViajeCorrecto**: debido a que hemos añadido el atributo kilometrosRecorridos en viaje, debemos comprobar que para todos los viajes terminados se haya registrado correctamente el número de kilómetros recorridos y coincidan con los del recorrido en cuestión

```
    inv kmRecorridosViajeCorrecto:
        self.fechaLlegada <> null implies
self.kilometrosRecorridos = self.recorrido.numKm
```

Código soil para comprobar:

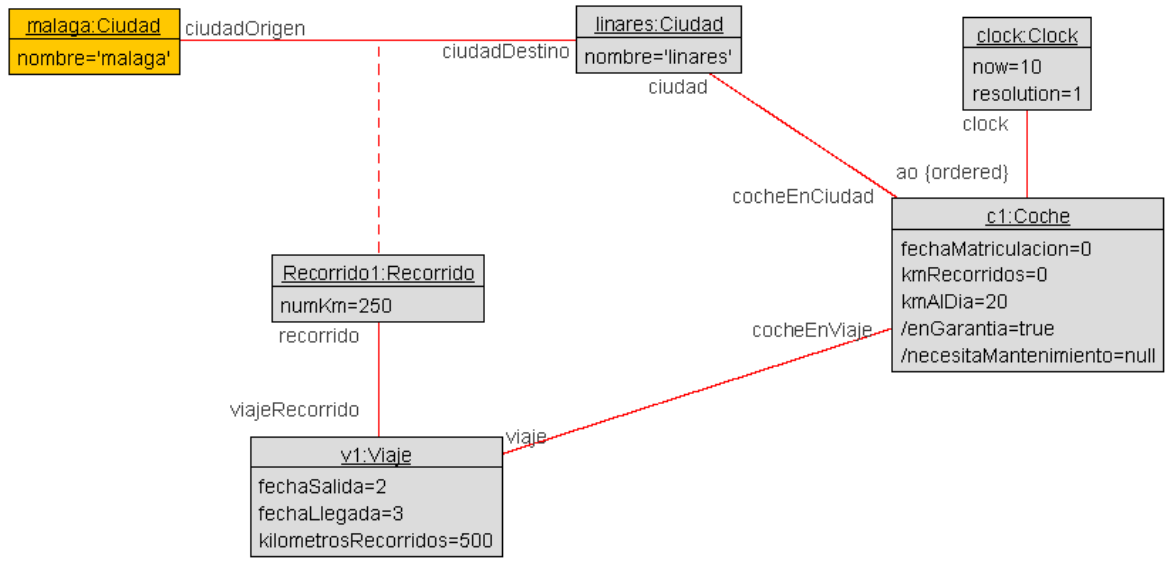
```
!new Clock('clock')
!clock.now := 10

!new Coche('c1')
!c1.fechaMatriculacion := 0

!c1.kmAlDia := 20

!insert(clock, c1) into Time

!new Ciudad('malaga')
!malaga.nombre := 'malaga'

!new Ciudad('linares')
!linares.nombre := 'linares'

!insert (c1, linares) into CocheCiudad

!insert (malaga, linares) into Recorrido
!Recorrido1.numKm := 250
```

```
!new Viaje('v1')
!v1.kilometrosRecorridos := 250
-- si cambiamos este valor o numKm del recorrido debería fallar la
invariante
!v1.fechaSalida := 2
!v1.fechaLlegada := 3

!insert (v1, Recorrido1) into ViajeRecorrido
!insert (c1, v1) into ViajeCoche
```



Viaje::kmRecorridosViajeCorrecto	true

Como podemos observar tenemos registrado un viaje que se realizó de Málaga a Linares, como el número de kilómetros que se recorrieron en ese viaje coincide con los del recorrido que se hizo, se cumple la invariante.



Viaje::kmRecorridosViajeCorrecto	false

Como vemos, el número de kilómetros registrados en el viaje no es consistente con los del recorrido que se hizo, por tanto la invariante no se cumple.

Invariantes cambiadas:

- **cocheViajandoOEnCiudad (Coche):** Hemos simplificado el código porque ahora el coche está viajando si su último viaje no tiene fecha de llegada, por tanto si existe ese viaje está viajando, si no, o no ha realizado ningún viaje o está en la ciudad del último viaje que realizó.

```
    inv cocheViajandoOEnCiudad:
        let viajes : Sequence(Viaje) =
self.viaje->sortedBy(fechaSalida)->asSequence() in
        let hoy : Integer = self.clock.now in
        --Si está viajando no tiene ciudad
        if viajes->last().fechaSalida <> null and
viajes->last().fechaLlegada = null then
            self.ciudad.oclIsUndefined()
        else
        -- si no esta viajando y tiene viajes anteriores,su última
ciudad es la de destino del último viaje terminado
            if viajes->last().fechaSalida <> null and
viajes->last().fechaLlegada <> null then
                self.ciudad =
viajes->last().recorrido.ciudadDestino
            else
            --si no hay viajes anteriores el coche está en alguna
ciudad
```

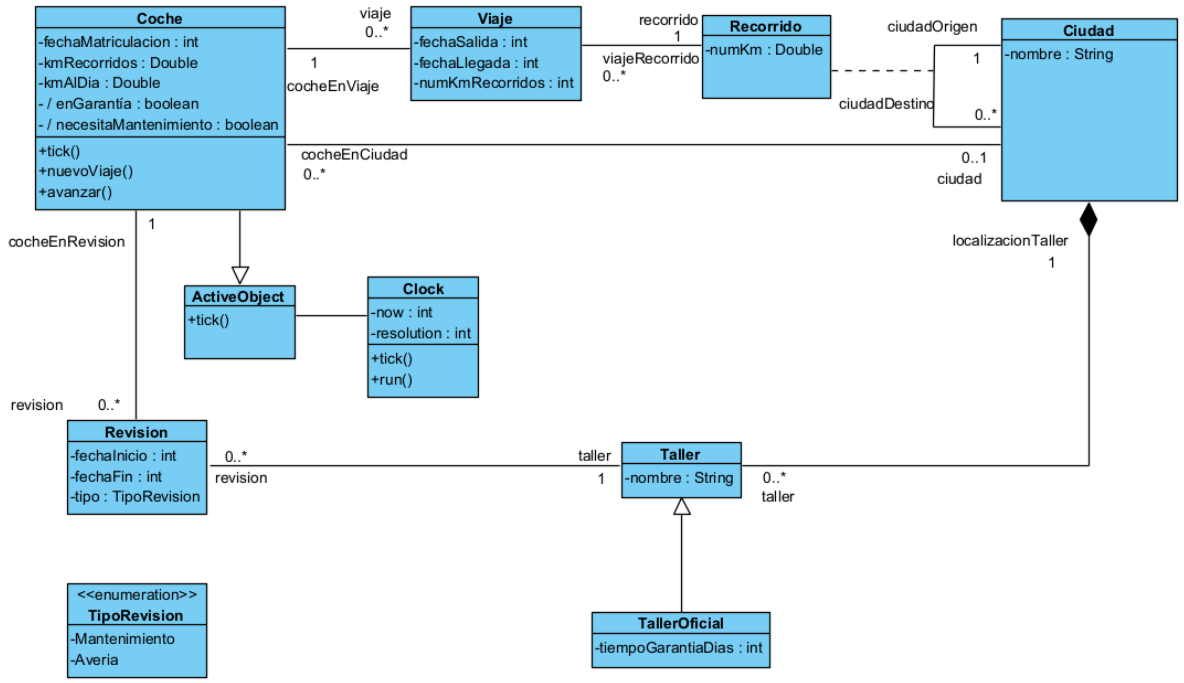
```
                not self.ciudad.oclIsUndefined()
            endif
        endif
```

- **fechaViajeValidas (Viaje):** Hemos cambiado esta invariante porque en el Apartado A los viajes tienen fecha de salida y llegada programadas, siendo estos 2 valores distintos de null, mientras que en el Apartado B la fecha de llegada al crear un viaje es null

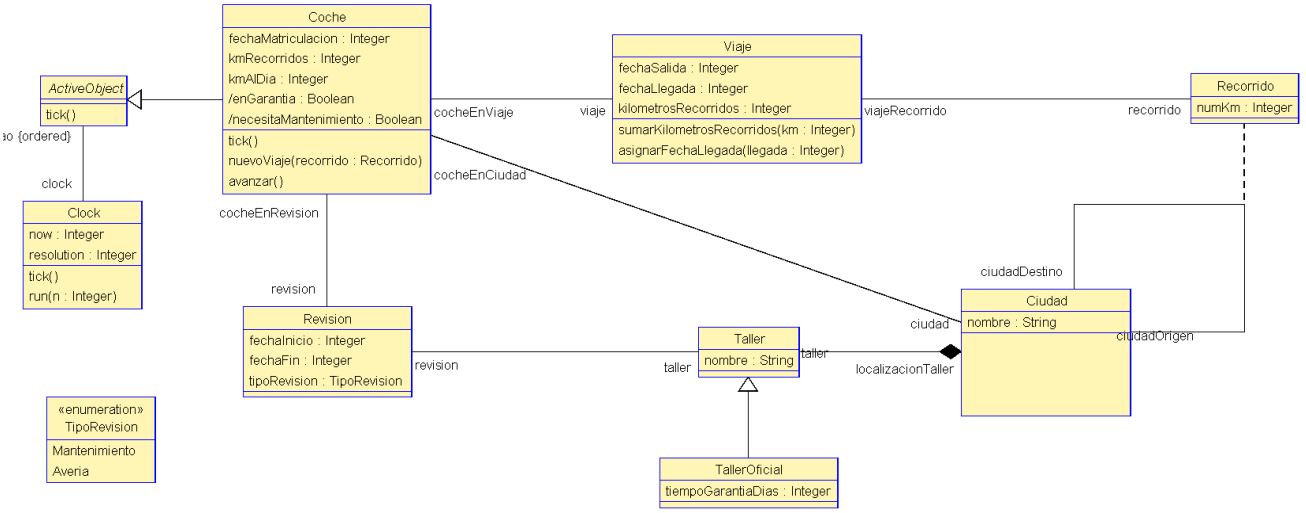
```
context Viaje
    inv fechasViajeValidas:
        ((self.fechaSalida <> null and self.fechaSalida>=0 ) and
        (self.fechaLlegada = null or self.fechaLlegada>=0)) and
        (self.fechaLlegada <> null implies (self.fechaSalida <=
self.fechaLlegada))
```

Diagramas de clases (Apartado B)

- Visual Paradigm:



- USE:



Simulacro del Sistema

En este apartado del documento se desarrollará y simulará el modelo con las siguientes características:

- ➔ Vamos a considerar tres ciudades: **Málaga, Sevilla y Granada**.

- ➔ Tendremos **dos recorridos**:
  - ◆ Málaga y Sevilla con 210 kilómetros
  - ◆ Sevilla y Granada con 250 kilómetros.

Supondremos un coche **matriculado en el instante 0 (día 0)** y que viaja a una **velocidad de 27**. El coche **comienza en Málaga** y continúa en Málaga **hasta el día 5**, día en que comienza un viaje haciendo **el recorrido de Málaga a Sevilla**. Los días van pasando y el coche va avanzando hasta que llega a Sevilla. El **mismo día que llega a Sevilla**, el coche comienza otro viaje haciendo **el recorrido entre Sevilla y Granada**. Los días van pasando y el coche continúa realizando el viaje. **Una vez llega a Granada, la simulación termina**.

El código de la simulación se encuentra en el apartado [GeneraApartadoC.soil]

Estado: Origen

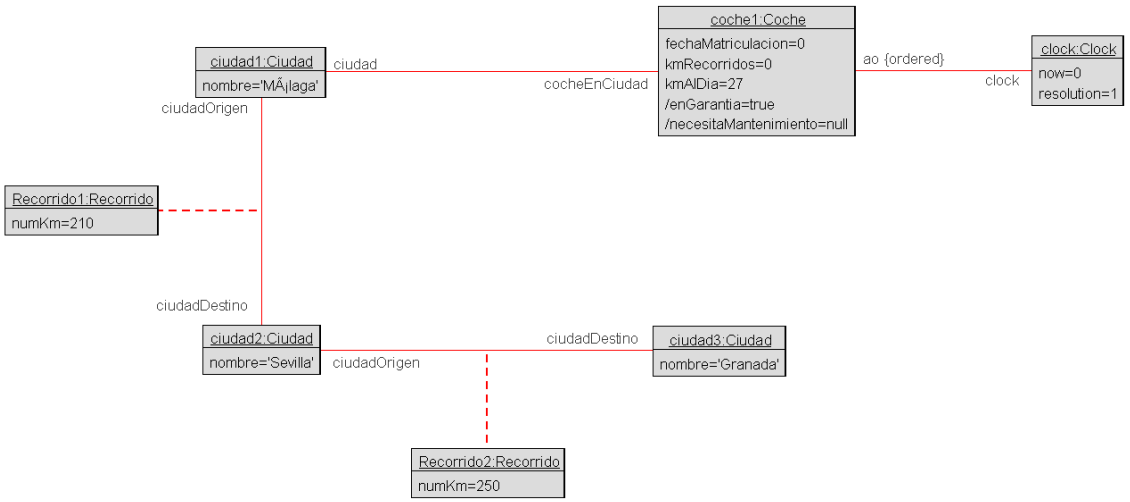


Diagrama de objetos inicial

El valor del atributo “now” se actualiza a medida que llamamos a la función tick de la clase reloj, hasta el instante 5.

Estado: Llegada a Sevilla

Creamos un objeto de tipo Viaje, cuya fecha de salida es el momento que indica el reloj (5) y está asociado al Recorrido 1. Es posible observar que la relación que tiene el coche con ciudad1 desaparece en el momento en el que se crea un viaje.

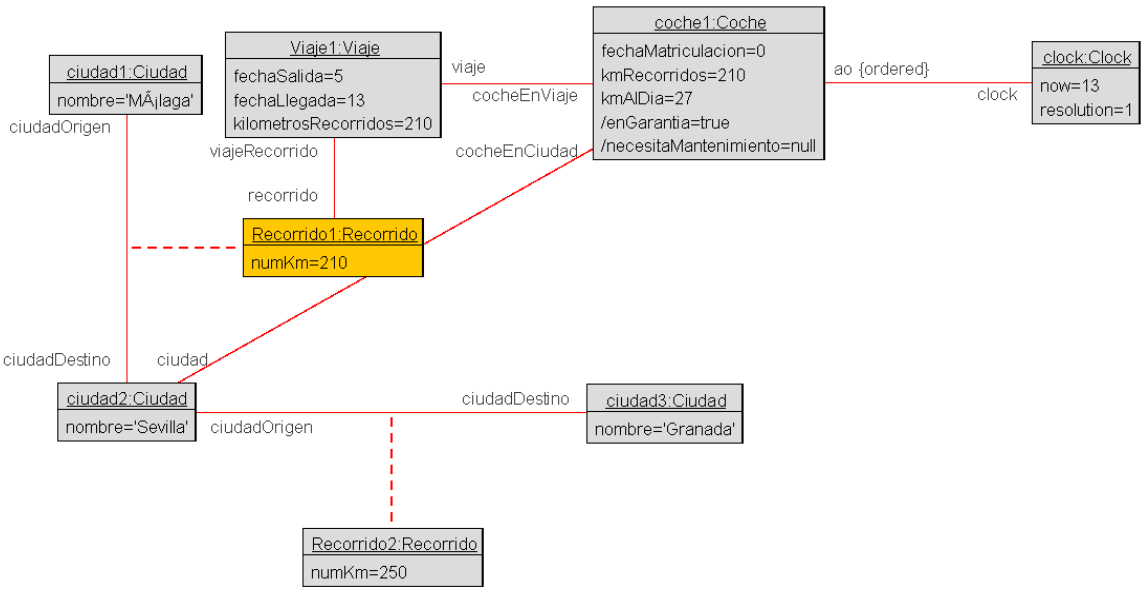


Diagrama de objetos al final del viaje

Después de 8 ticks, el viaje concluye, con la variable de kilómetros recorridos del viaje indica que este ha viajado por el recorrido completo. La variable que indica cuántos kilómetros ha recorrido un coche también se ha actualizado. Dado que el viaje ha concluido, coche se relaciona ahora con la ciudad destino, ciudad2.

En el mismo tick de reloj que termina el viaje 1, creamos el viaje 2, asignado al recorrido 2. Vuelve a desaparecer la relación entre coche y ciudad2.

Estado: Llegada a Granada

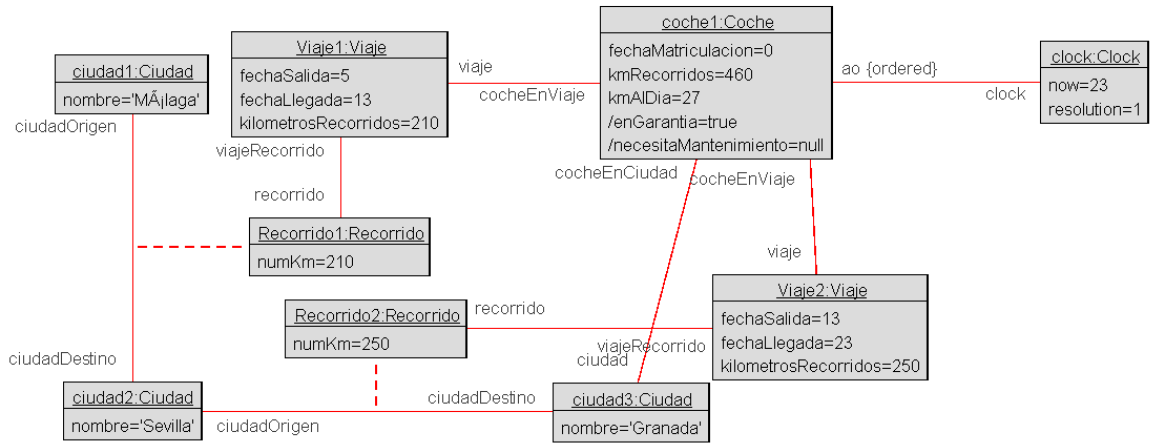
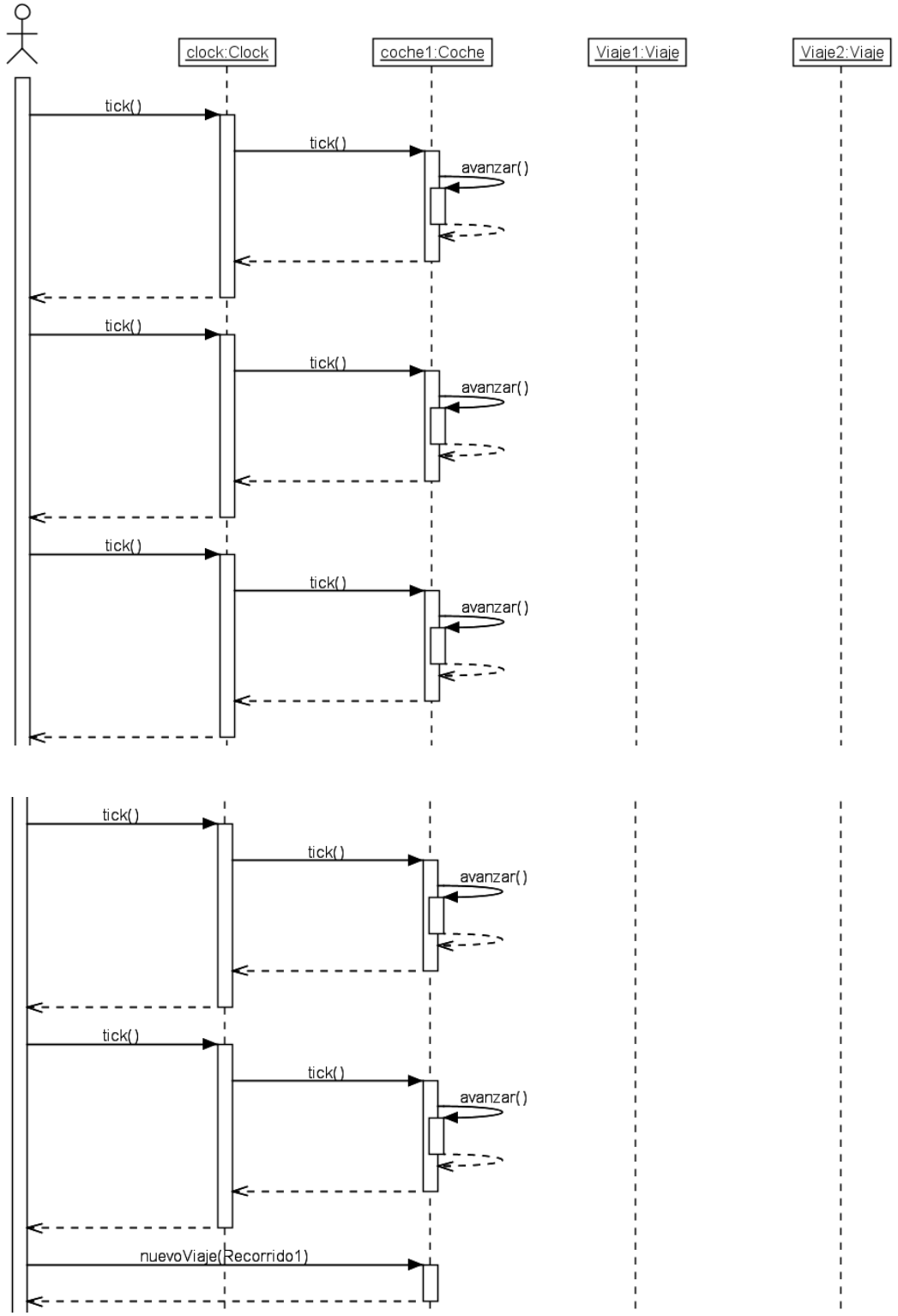


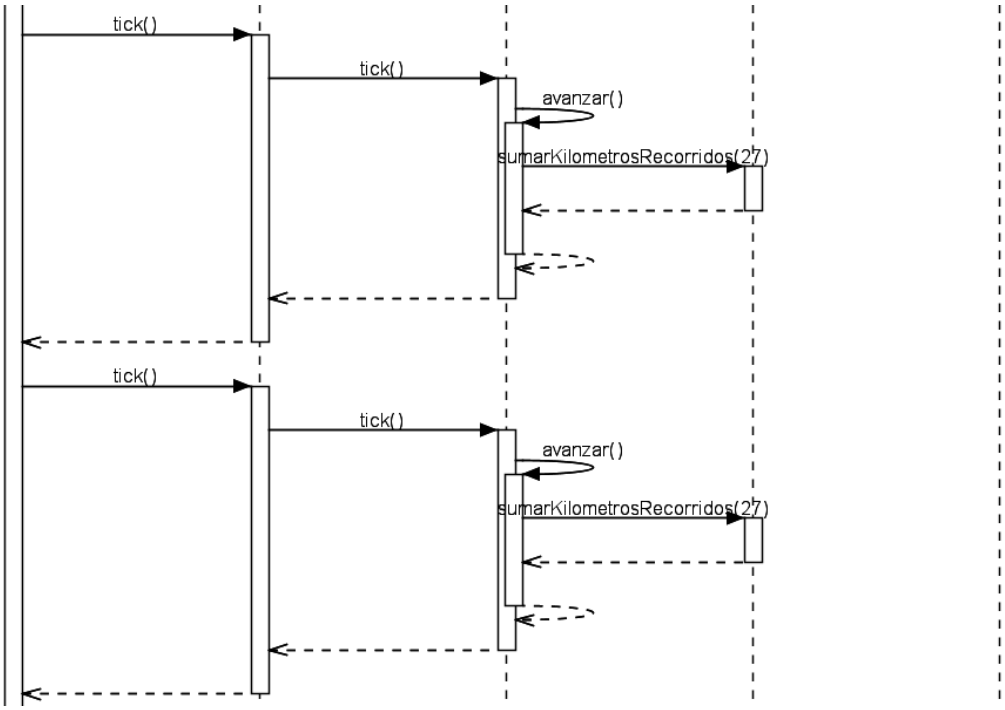
Diagrama de objetos final de la secuencia

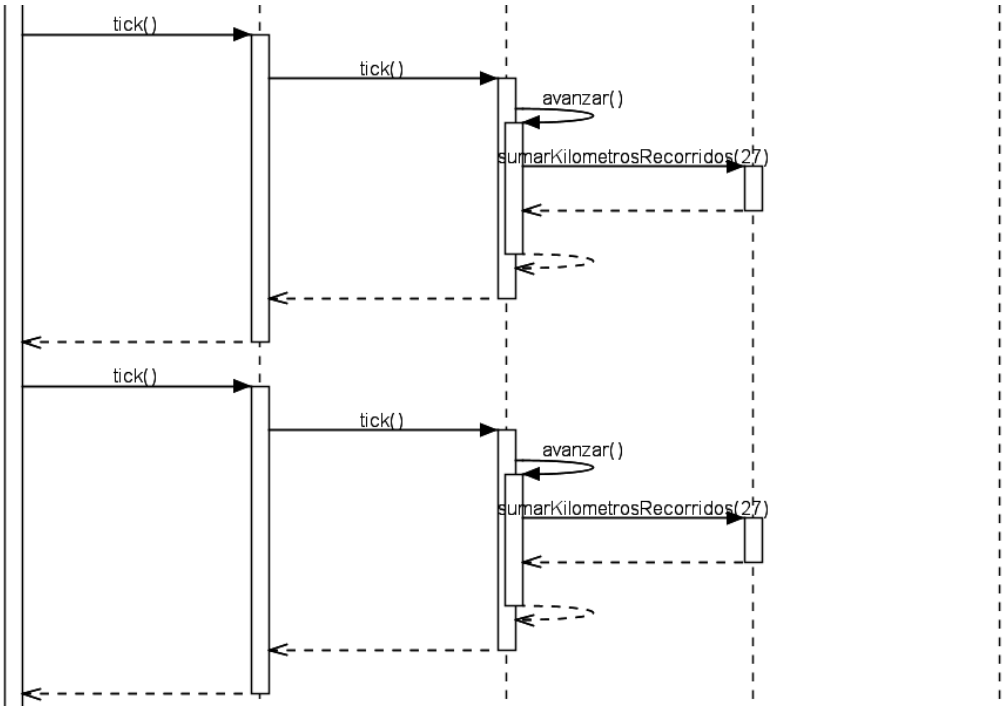
Como podemos observar, el coche mantiene los viajes almacenados, indicando la fecha de salida y de llegada (debido a que los ha terminado), asimismo, tiene una relación con ciudad3 que simboliza la posición actual del coche.

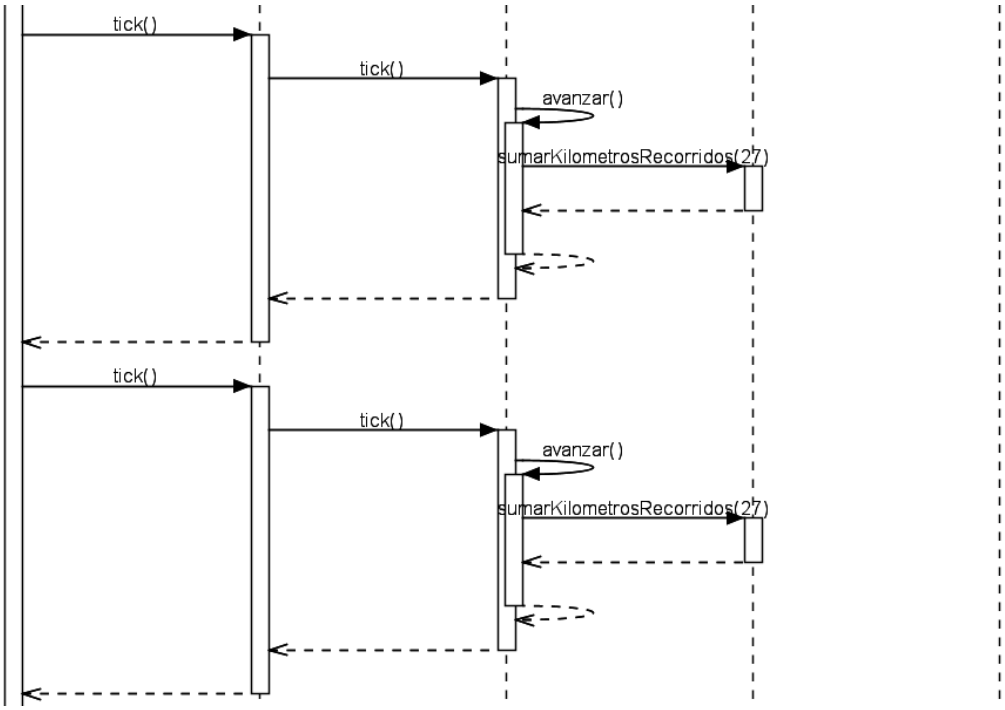
Aclaración: se ha comprobado que en todos los pasos de la simulación no se viole ninguna invariante.

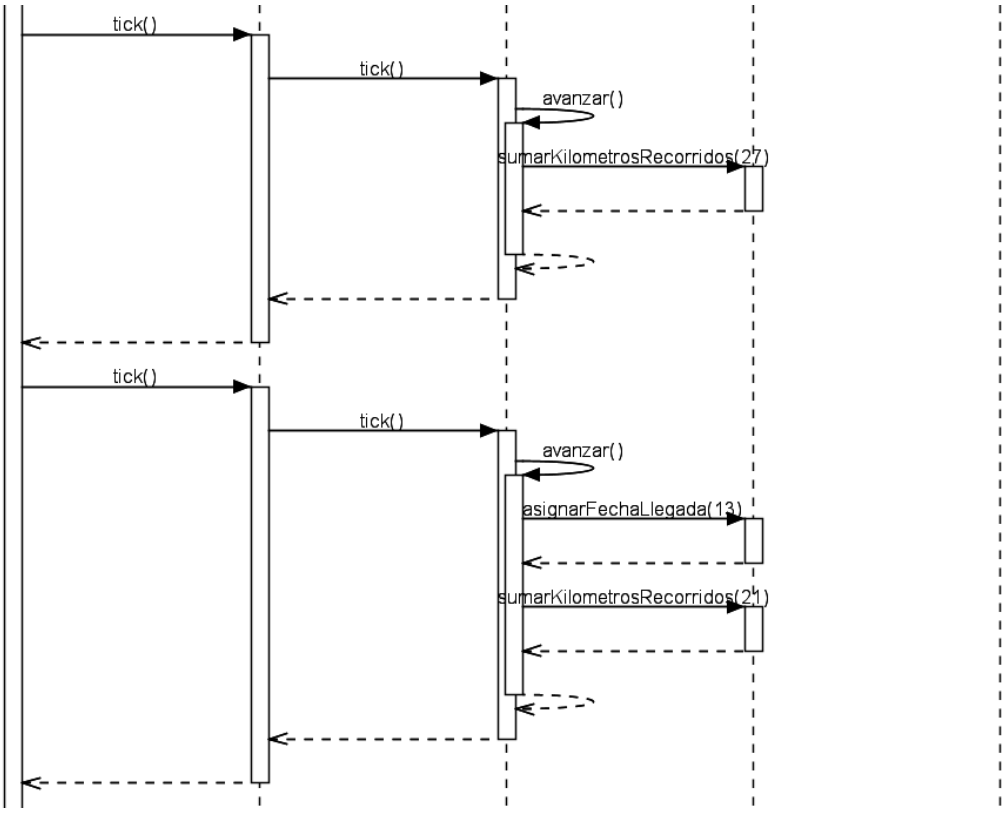
DIAGRAMA DE SECUENCIA

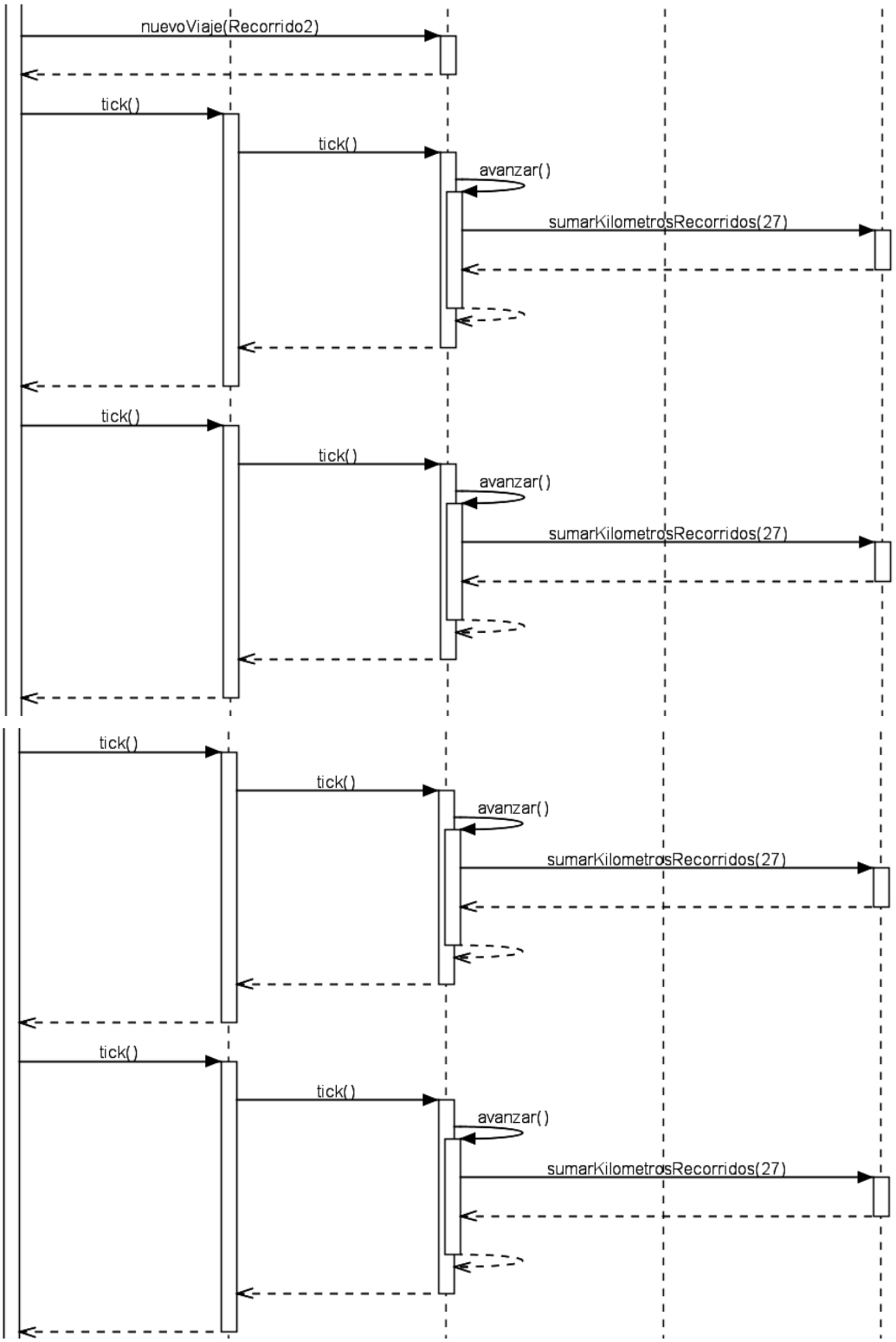


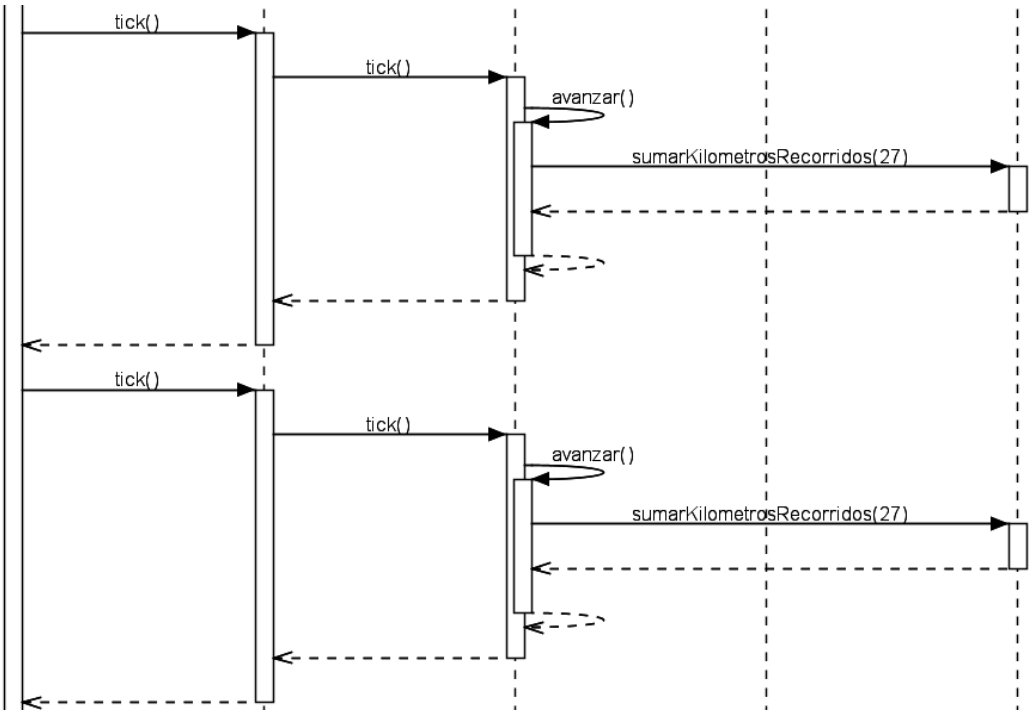

tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)

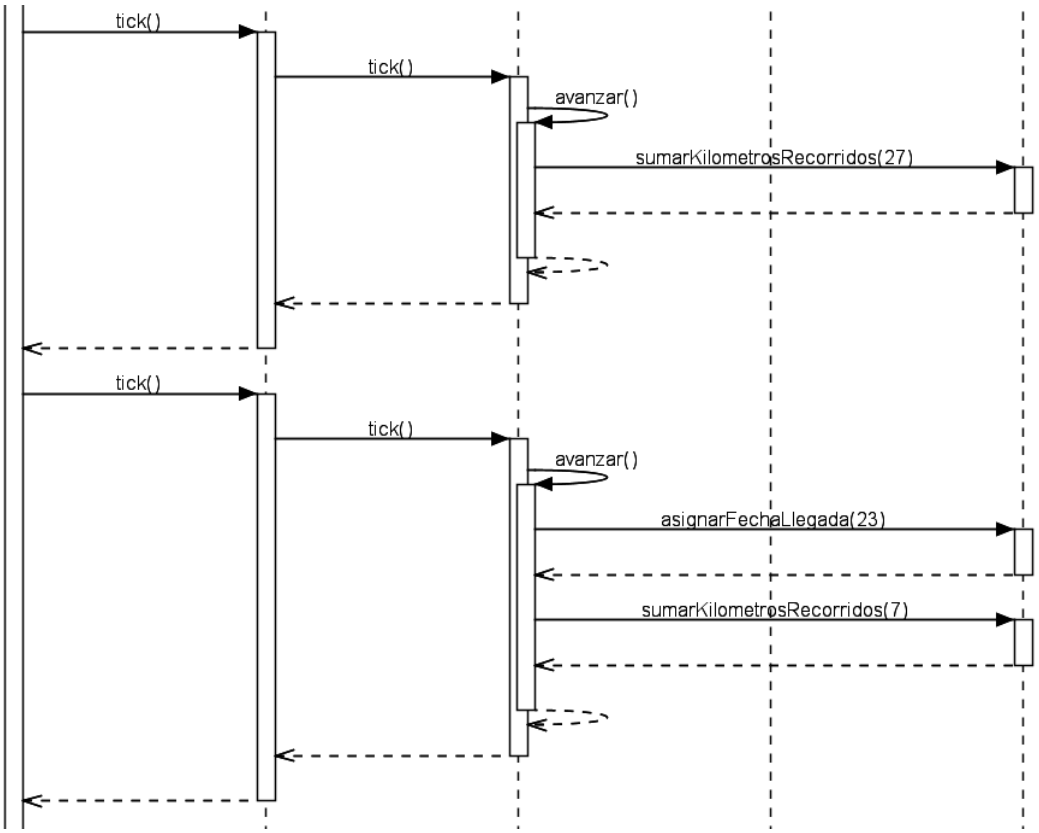

tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)


tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)


tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
asignarFechaLlegada(13)
sumarKilometrosRecorridos(21)


nuevoViaje(Recorrido2)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)


tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)


tick()
tick()
avanzar()
sumarKilometrosRecorridos(27)
tick()
tick()
avanzar()
asignarFechaLlegada(23)
sumarKilometrosRecorridos(7)

Códigos

Practica2.use (Apartado A)

```
model Practica2

----------Clases-----------
class Clock
attributes
    now : Integer
end

class Coche < ActiveObject
attributes
    fechaMatriculacion : Integer
    kmRecorridos : Real
        derive : self.viaje->select(v | v.fechaLlegada <=
self.clock.now).recorrido.numKm->sum()
    --Este hay que actualizarlo en el apartado B
    enGarantia : Boolean
        derive: (self.clock.now - self.fechaMatriculacion) < 400 or
            self.revision->exists(r |
r.taller.oclIsTypeOf(TallerOficial) and
                    (self.clock.now - r.fechaFin) <=

r.taller.oclAsType(TallerOficial).tiempoGarantiaDias)
    --Este hay que actualizarlo en el apartado B
    -- probar que pasa cuando se hace un mantenimiento antes de los 400
dias
    necesitaMantenimiento : Boolean
        derive : (self.clock.now - self.fechaMatriculacion) > 400 and
                (self.clock.now - self.revision
                            ->select(r | r.tipoRevision =
TipoRevision::Mantenimiento)

->sortedBy(fechaFin)->last().fechaFin) > 100
        -- derive: if self.revision->select(r | r.tipoRevision =
TipoRevision::Mantenimiento)->notEmpty()
        --             and (self.clock.now - self.fechaMatriculacion) >
400 then
        --     (self.clock.now - self.revision->select(r |
r.tipoRevision = TipoRevision::Mantenimiento)
```

```
        --
->sortedBy(fechaFin)->last().fechaFin) > 100
        -- else
        --     (self.clock.now - self.fechaMatriculacion) > 400
        -- endif
end



class Viaje
attributes
    fechaSalida : Integer
    fechaLlegada : Integer
end

class Ciudad
attributes
    nombre : String
end

associationclass Recorrido  between
    Ciudad[1] role ciudadOrigen
    Ciudad[0..*] role ciudadDestino
    attributes
        numKm : Real
end

class Taller
attributes
    nombre : String
end

class TallerOficial < Taller
attributes
    tiempoGarantiaDias : Integer
end

enum TipoRevision {Mantenimiento, Averia}

class Revision
attributes
    fechaInicio : Integer
    fechaFin : Integer
```

```
    tipoRevision : TipoRevision
end

abstract class ActiveObject
end

----------Fin de clases-----------

----------Asociaciones------------
association ViajeCoche between
    Coche[1] role cocheEnViaje
    Viaje[0..*] role viaje
end

association RevisionCoche between
    Coche[1] role cocheEnRevision
    Revision[0..*] role revision
end

association RevisionTaller between
    Taller[1] role taller
    Revision[0..*] role revision
end

association ViajeRecorrido between
    Viaje[0..*] role viajeRecorrido
    Recorrido[1] role recorrido
end

association CocheCiudad between
    Coche[0..*] role cocheEnCiudad
    Ciudad[0..1] role ciudad
end

association Time between
    Clock[1] role clock
    ActiveObject[*] role ao ordered
end

----------Composiciones-----------

composition CiudadTaller between
    Ciudad[1] role localizacionTaller
```

```
    Taller[0..*] role taller
end

----------Invariantes------------
constraints
--En el apartado A solo tendremos una única insatncia del objeto clock
--Para indicar esto crearemos una nueva invariante

context Clock
    inv unicoClock:
        Clock.allInstances()->size() = 1

context Recorrido
    inv DistanciaMinima:
        self.numKm > 5

context Coche
    inv fechaMatriculacionValida:
        self.fechaMatriculacion >= 0 and self.fechaMatriculacion <>
null

    inv tallerMismaCiudad:
        let revisiones: Sequence(Revision) = self.revision->
sortedBy(fechaFin) -> asSequence() in
        let hoy : Integer = self.clock.now in
        revisiones->forAll(r |
            (r.fechaInicio <= hoy and hoy <= r.fechaFin) implies
            r.taller.localizacionTaller = self.ciudad
        )

    inv ciudadDestinoEsCiudadOrigenSiguiente:
        let viajesOrdenados : Sequence(Viaje) = self.viaje
                ->sortedBy(fechaSalida)->asSequence() in
        viajesOrdenados->forAll(v |
            viajesOrdenados->indexOf(v) < viajesOrdenados->size()
implies
                v.recorrido.ciudadDestino = viajesOrdenados
                    ->at(viajesOrdenados->indexOf(v) +
1).recorrido.ciudadOrigen
        )
        -- ordenamos los viajes registrados de un coche por fecha de
salida
```

```
        -- y comprobamos para todos los viajes que la siguiente ciudad
de destino actual
        -- es la ciudad de origen de la siguiente

    inv viajeDetrasDeOtro:
        let viajesOrdenados : Sequence(Viaje) = self.viaje
            ->sortedBy(fechaSalida)->asSequence() in
        viajesOrdenados->forAll(v |
            viajesOrdenados->indexOf(v) < viajesOrdenados->size()
implies
            v.fechaLlegada <= viajesOrdenados
                    ->at(viajesOrdenados->indexOf(v) + 1).fechaSalida
        )
    -- Esta invariante comprueba que un si un viaje empieza el 1 y
acaba el 2,
    -- el siguiente viaje tiene que empezar el 3 y asi sucesivamente


    inv cocheViajandoOEnCiudad:
        let viajes : Sequence(Viaje) =
self.viaje->sortedBy(fechaSalida)->asSequence() in
        let hoy : Integer = self.clock.now in
        if self.viaje->isEmpty() then
                -- si el coche no se encuentra en ningun viaje y no
tiene ningun viaje pendiente
                -- la ciudad del coche tiene que estar definida
                not self.ciudad.oclIsUndefined()
        else
            if viajes->exists(v | v.fechaSalida <= hoy and hoy <=
v.fechaLlegada) then
                -- Si el coche está en un viaje, su ciudad debe ser
indefinida
                self.ciudad.oclIsUndefined()
            else
                if  not viajes->select(v | v.fechaSalida >=
hoy)->isEmpty() then
                    -- Si el coche no tiene viajes realizados pero
tiene viajes pendientes,
                    -- su ciudad es la ciudad origen del primer viaje
                    viajes->select(v | v.fechaSalida >=
hoy)->first().recorrido.ciudadOrigen = self.ciudad
                else
```

```
                    -- Si tiene viajes realizados, su ciudad debe
coincidir con el destino del último viaje finalizado
                    viajes->select(v | v.fechaLlegada <=
hoy)->last().recorrido.ciudadDestino = self.ciudad

                endif
            endif
        endif

    inv unaRevisionALaVez:
        self.revision->forAll(r1,r2 | r1 <> r2 implies
            (r1.fechaInicio <> r2.fechaInicio and r1.fechaFin <>
r2.fechaFin) and
            not (r1.fechaInicio <= r2.fechaInicio and r2.fechaInicio <=
r1.fechaFin) and
            not (r1.fechaInicio <= r2.fechaFin and r2.fechaFin <=
r1.fechaFin)
        )


context Revision
    inv fechasRevisionValidas:
        ((self.fechaInicio <> null and self.fechaInicio>=0) and
        (self.fechaFin <> null and self.fechaFin>=0)) and
        (self.fechaInicio <= self.fechaFin)

    inv fechaRevisionPosteriorCoche:
        self.cocheEnRevision.fechaMatriculacion <= self.fechaInicio

context TallerOficial
    inv garantiaValida:
        self.tiempoGarantiaDias <> null and self.tiempoGarantiaDias > 0

context Ciudad
    inv solo1TallerOficial:
        self.taller->select(t|t.oclIsTypeOf(TallerOficial))->size()<=1

context Viaje
    inv fechasViajeValidas:
        ((self.fechaSalida <> null and self.fechaSalida>=0) and
        (self.fechaLlegada <> null and self.fechaLlegada>=0)) and
        (self.fechaSalida <= self.fechaLlegada)
```

GeneralApartadoC.soil

```
--El sistema parte del instante 0
!new Clock ('clock')
!clock.now := 0

!new Coche('coche1')
!coche1.fechaMatriculacion := 0
!coche1.kmAlDia := 27

!insert (clock, coche1) into Time

!new Ciudad ('ciudad1')
!ciudad1.nombre := 'Málaga'

!new Ciudad ('ciudad2')
!ciudad2.nombre := 'Sevilla'

!new Ciudad ('ciudad3')
!ciudad3.nombre := 'Granada'

--El coche está inicialmente en Málaga
!insert (coche1,ciudad1) into CocheCiudad

--Recorrido entre malaga - sevilla 210 km y sevilla - granada 250 km
!insert(ciudad1, ciudad2) into Recorrido
!Recorrido1.numKm := 210

!insert(ciudad2, ciudad3) into Recorrido
!Recorrido2.numKm := 250
```