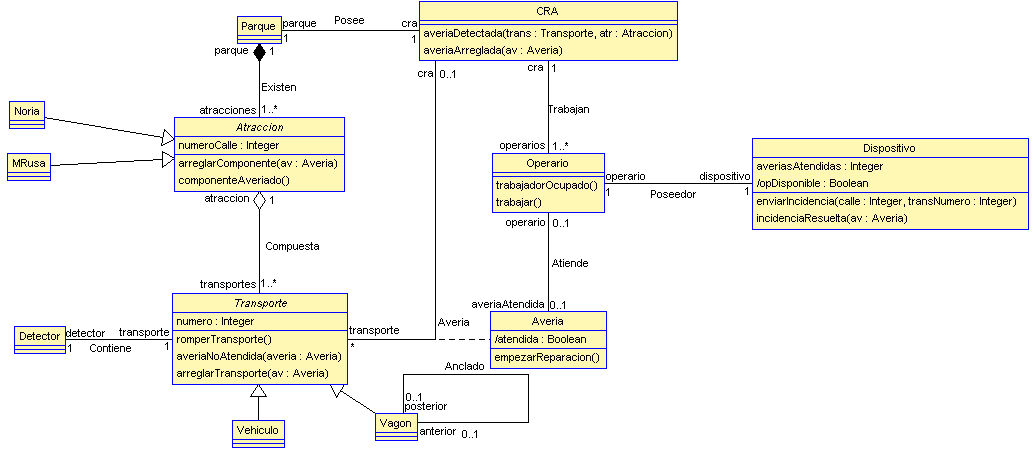
**Divertimento S.A.**

## **Modelo en USE**



El modelo consta de las siguientes clases:

### Parque

Hemos decidido añadir la clase Parque ya que, como bien dice el enunciado, “tiene varios parques de atracciones repartidos por la geografía nacional.”, y, a pesar de que el enunciado dice *“para ello ya ha decidido poner en marcha un proyecto piloto cuyo objetivo será el de dotar a uno de sus parques de atracciones”*, hemos pensado que en un futuro, a la hora de desplegar el modelo a todo el dominio, sería más fácil añadir el resto de tipos de atracciones, y de tipos de Transportes (que realmente, se puede entender transporte como componente de una atracción).

### CRA

El CRA es la Central Receptora de Averías de cada Parque de Atracciones. Aunque en el enunciado del problema no se especifica, hemos supuesto que, por cada parque de atracciones, tendría una de estas centrales, ya que los parques estarán en distintas ciudades separadas por varios kilómetros de distancia.

### *Atraccion*

### Noria

### MRusa (Montaña Rusa)

Hemos decidido modelar las atracciones como clase abstracta y, a partir de ahí, en caso de existir atracciones distintas a estas dos, añadir clases que hereden de la abstracta, pues la idea del proyecto es, en un futuro, implementarlo en todos los parques de la Empresa. En el caso del prototipo, sólo se nos especifican 2 tipos de atracciones distintas, Noria y MRusa (Montaña Rusa).

### *Transporte*

### Vehiculo

### Vagon

Siguiendo el mismo razonamiento que en la clase *Atraccion*, al sólo indicarse dos tipos de componente o transporte, tenemos las clases Vehiculo y Vagon.

La única diferencia en este caso, es que los vagones tienen una referencia al vagón que llevan anclado detrás.

### Averia

Realmente Avería es una clase de asociación entre el CRA y un componente de una atracción. En nuestra solución hemos decidido añadir avería como una clase, ya que creemos que puede ser interesante saber el estado (Pendiente, en proceso, terminada) de la misma con una máquina de estados. Además, la relación está modelada como se observa ya que al arreglar una avería la eliminamos del sistema puesto que no se indica que haya que llevar un registro de las averías que surgen.

### Operario

Esta clase representa un operario y tiene un atributo derivado que indica si está o no libre. Si está libre depende de si tiene o no una avería asociada.

### Dispositivo

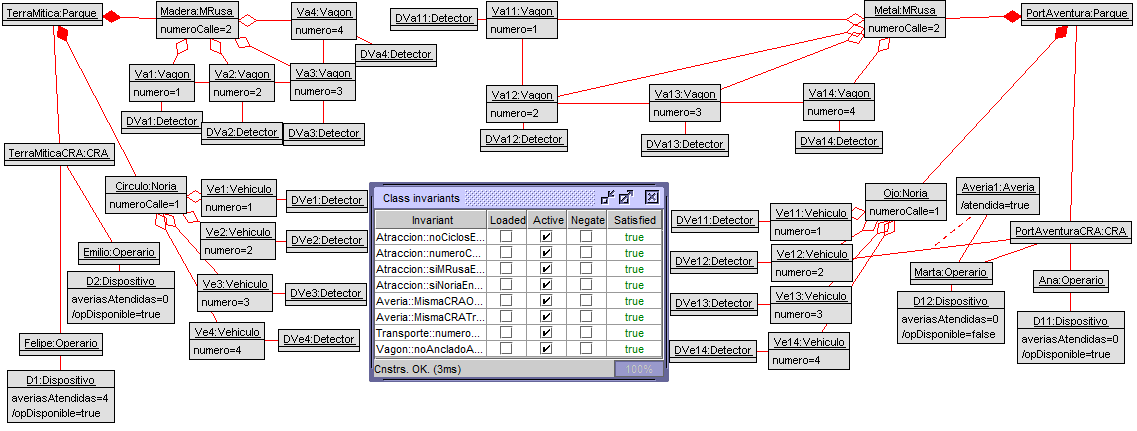
Esta clase representa al dispositivo asociado al operario que registra las averías que ha reparado un operario.

### 

### Detector

A pesar de no tener operaciones ni ser relevante en el modelo, hemos decidido añadirlo puesto que así se indica en el problema. “La noria tiene una serie de vehículos dotados cada uno de ellos de un detector” y “la montaña rusa cada vagón está dotado de igual modo de un detector de anclaje”.

## **Diagrama de Objetos**



## **Restricciones en OCL**

* Averia(context Averia):
  + El operario que atiende la avería trabaja en la CRA a la que se asigna.

inv MismaCRAOperarios:

self.operario.cra = self.cra

* + La CRA del parque del transporte averiado es la misma que a la que se

envía la avería

inv MismaCRATransporte:

self.transporte.atraccion.parque.cra = self.cra

* Atraccion (context Atraccion):
  + El número de calle está definido.

inv numeroCalleDefinido:

**not** self.numeroCalle.oclIsUndefined()

* + El tipo de transporte de las norias es Vehiculo.

inv siNoriaEntoncesVehiculos:

self.oclIsTypeOf(Noria) implies transportes.forAll(t | t.oclIsTypeOf(Vehiculo))

* + El tipo de transporte de las montañas rusas es Vagon.

inv siMRusaEntoncesVagones:

self.oclIsTypeOf(MRusa) implies transportes.forAll(t | t.oclIsTypeOf(Vagon))

* + Los vagones de las montañas rusas no forman ciclos

inv noCiclosEnVagones:

self.oclIsTypeOf(MRusa) implies

(self.transportes->one(t | t.oclAsType(Vagon).anterior->isEmpty()) **and**

self.transportes->one(t | t.oclAsType(Vagon).posterior->isEmpty()))

* Vagon (context Vagon):
  + Está anclado a un vagón distinto a sí mismo o a ningún vagón

inv noAncladoASiMismo:

posterior <> anterior **or** (self.anterior->size() = 0 **and** self.posterior->size() = 0)

* Transporte (context Transporte):
  + Está definido el numero de transporte

inv numeroDefinido:

**not** self.numero.oclIsUndefined()

## **Operaciones**

**Operaciones del CRA**

**averiaDetectada(trans: Transporte, atr: Atraccion)**

begin

declare a: Averia, op: Operario;

**if** trans.averia->isEmpty() then

a := new Averia between (trans, self);

**if** self.operarios->select(op | op.dispositivo.opDisponible)->notEmpty() then

op := self.operarios->any(op | op.dispositivo.opDisponible);

op.dispositivo.enviarIncidencia(atr.numeroCalle, trans.numero);

**else**

trans.averiaNoAtendida(a);

end

**else**

a := trans.averia;

**if** self.operarios->select(op | op.dispositivo.opDisponible)->notEmpty() then

op := self.operarios->any(op | op.dispositivo.opDisponible);

op.dispositivo.enviarIncidencia(atr.numeroCalle, trans.numero);

end

end

end

**pre** transportePerteneceAtraccion: atr.transportes->includes(trans)

**pre** mismoParque: atr.parque = self.parque

**post** nuevaAveria: Averia.allInstances->select(av | av.oclIsNew())->notEmpty() implies

Averia.allInstances->size() = Averia.allInstances**@pre**->size() + 1

Cuando se llama a esta operación comprueba si el transporte pasado por parámetro tiene ya o no una avería (se hace esta comprobación para poder realizar el polling del transporte al CRA). Después de esto, se busca algún operario libre, si se ha encontrado un operario, se le manda a su dispositivo asociado el número de transporte y de calle en el que se encuentra la avería. La operación consta de 2 precondiciones y una postcondición, transportePerteneceAtracción que comprueba que ese transporte pertenezca a la atracción que se pasa por parámetro, mismoParque, que comprueba que el CRA y la atracción pertenezcan al mismo parque, y una post-condición que selecciona todas las averías creadas en la operación y si no es vacío, implica que el tamaño ha aumentado.

**Clase Transporte**

**romperTransporte()**

begin

self.atraccion.parque.cra.averiaDetectada(self, self.atraccion);

self.atraccion.componenteAveriado();

end

**pre** noUltimoVagon: self.oclIsTypeOf(Vagon) implies self.oclAsType(Vagon).posterior -> notEmpty()

Esta función “rompe” el transporte y avisa al CRA y a la Atracción de que se ha roto. La única precondición es que no se puede romper el último vagón, puesto que en el enunciado se dice que *“cada vagón está dotado de igual modo de un detector de* ***anclaje con el vagón que lleva detrás.****”* Por tanto, no tiene sentido que el último vagón, que no está anclado a ninguno, mande una señal de poco anclaje con el anterior. Además, esta operación sirve para que el estado de la máquina de estados de la clase transporte pase de “Funcionando” a “Averiado”.

**averiaNoAtendida(averia: Averia)**

begin

**while** (**not** averia.atendida) do

self.atraccion.parque.cra.averiaDetectada(self, self.atraccion);

end

end

**pre** existeAveria: self.averia->notEmpty()

**pre** mismaAveria: self.averia = averia

Esta operación simula el polling que realiza el transporte en el momento en el que no hay operarios libres, sin embargo, en el momento en el que entra en el while, el transporte toma el control de la ejecución del script .soil y no permite a ningún operario liberarse, porque se queda dentro del bucle. Tiene 2 precondiciones la operación, una que indica que exista una avería, y que la avería que se pasa por parámetro sea la misma avería que tiene el transporte asignado.

**arreglarTransporte(av: Averia)**

begin

self.atraccion.arreglarComponente(av);

end

**pre** existeAveria: self.averia->notEmpty

**pre** averiaAtendida: self.averia.atendida

**pre** mismaAveria: self.averia = av

**post** AveriaEliminada: self.averia->isEmpty() **and** Averia.allInstances->excludes(av)

Esta operación notifica a la atracción de que el transporte actual ha sido arreglado, existen varias precondiciones y postcondiciones, por ejemplo, que existan averías, que la avería a cerrar, ha sido atendida, que sean la misma avería la atendida con la que te pasan por parámetro y de postcondición, que la avería ha sido eliminada. Esta operación sirve también para transitar de un estado a otro, en concreto de “Averiado” a “Funcionando”.

**Clase Atraccion**

**arreglarComponente(av: Averia)**

begin

destroy(av);

end

**pre** averiaEnAtrasion: self.transportes->collect(averia)->includes(av)

**post** noMoreAverias:

self.transportes->select(t | t.averia->notEmpty())->size() =

self.transportes->select(t | t.averia**@pre**->notEmpty())->size() - 1

Esta operación elimina la avería al ser resuelta. La precondición sería que exista una avería y la postcondición, es que el número de averías después de la ejecución de la operación es menor al número de averías que hay antes de la ejecución de la operación, en concreto, hay una menos después de la ejecución de la operación. La operación sirve como notificación para la clase atracción y para que así transite de un estado a otro en la máquina de estados.

**componenteAveriado()** -- When a component **is** broken **and** notifies this **class**

begin

end

**pre** hayTransporteConAveria: self.transportes->exists(t | t.averia->notEmpty())

Esta operación sirve para cambiar de estado la atracción, cuando un transporte se rompe, éste llama a esta operación. La operación sirve para notificar a la noria de que se le ha roto un transporte. La precondición es que cuando en el momento en el que se llama a la operación, existe al menos una avería en el conjunto de todos los transportes de esa atracción. La operación también notifica a la clase de que un transporte ha sido arreglado, transitando así de un estado a otro.

## **Clase Operario**

**trabajadorOcupado()**

begin

self.averiaAtendida.empezarReparacion();

end

**pre** existeAveriaAReparar: self.averiaAtendida->notEmpty()

Esta operación asigna un operario a una avería para que este empiece a repararla. La única precondición es que debe existir una avería para reparar. Además, notifica a la avería de que ésta está siendo atendida para que la avería cambie de estado de Pendiente a En proceso. Además, el operario transita del estado de Libre a Ocupado.

**trabajar()**

begin

self.dispositivo.incidenciaResuelta(self.averiaAtendida);

end

**pre** ocupado: **not** self.dispositivo.opDisponible

**post** operarioLibre: self.dispositivo.opDisponible

Esta operación simula la finalización de la tarea de un operario y notifica al dispositivo que ha resuelto la avería que tenía asignada. Para ello, el operario debe estar ocupado (tener asignado una avería) antes de repararla y tras solucionarlo, debe estar libre para poder recibir una nueva avería. Además la operación hace que el operario transite de “Ocupado” a “Libre”.

**Clase Dispositivo**

**enviarIncidencia(calle: Integer, transNumero: Integer)**

begin

declare av: Averia;

av := Averia.allInstances->select(av | av.transporte.numero = transNumero

**and** av.transporte.atraccion.numeroCalle = calle)->asSequence()->first();

insert (self.operario, av) into Atiende;

self.operario.trabajadorOcupado();

end

**pre** averiaEnCalleYTransporte: Averia.allInstances->collect(transporte)->asSet()

->exists(t | t.numero = transNumero **and** t.atraccion.numeroCalle = calle)

**post** trabajadorOcupado: let av: Averia = Averia.allInstances->

select(av | av.transporte.numero = transNumero

**and** av.transporte.atraccion.numeroCalle = calle **and**

self.operario.cra.parque.atracciones->includes(av.transporte.atraccion))

->asSequence()->first() **in**

**not** self.opDisponible **and** self.operario.averiaAtendida = av

Esta operación asigna una avería sin atender al operario propietario del dispositivo y se lo notifica a su operario. Una precondición sería que exista una avería en esa calle y en ese transporte; y una postcondición es que el trabajador tenga asignada la avería y esté ocupado.

**incidenciaResuelta(av: Averia)**

begin

declare cra: CRA;

cra := self.operario.cra;

cra.averiaArreglada(av);

self.averiasAtendidas := self.averiasAtendidas + 1;

end

**pre** trabajadorOcupado: **not** self.opDisponible

**pre** trabajadorAtiendeAveria: self.operario.averiaAtendida = av

**post** averiasResueltasIncrementada: self.averiasAtendidas = self.averiasAtendidas**@pre** + 1

**post** trabajadorLibre: self.opDisponible

Esta operación simula el fin de una avería, es decir, que está arreglada, provocando así que el contador de averías atendidas aumente en uno. En las precondiciones tenemos como un trabajador que está ocupado significa que está atendiendo alguna avería, y como segunda precondición, el operario atiende en concreto la avería que se le ha pasado por parámetro. En las postcondiciones una vez atendida la avería el contador de averías atendidas aumentará y el trabajador que la atendía cambiará de estado a trabajador disponible.

**Clase Averia**

**empezarReparacion()**

begin

end

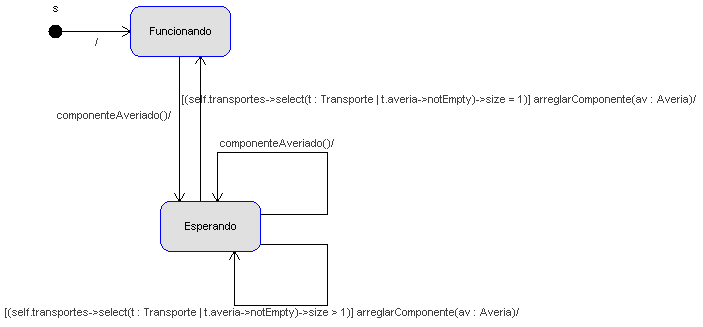
**pre** existeOperarioAtendiendo: self.atendida

Esta operación sirve para notificar a la avería de que está siendo atendida para que así cambie su estado en la máquina de estados.

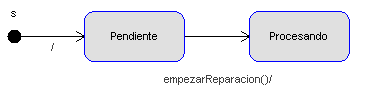
## 

## **Máquinas de estado**

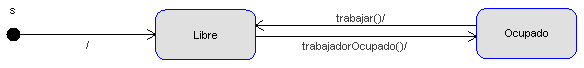
* Atracción:



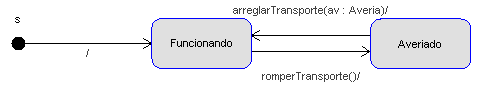
* Avería:



* Operario:



* Transporte:



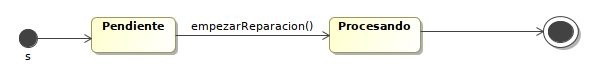
## 

## **Modelo en Magic Draw**

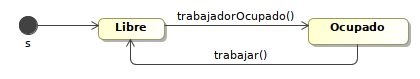
## Atracción:

## 

* Avería:



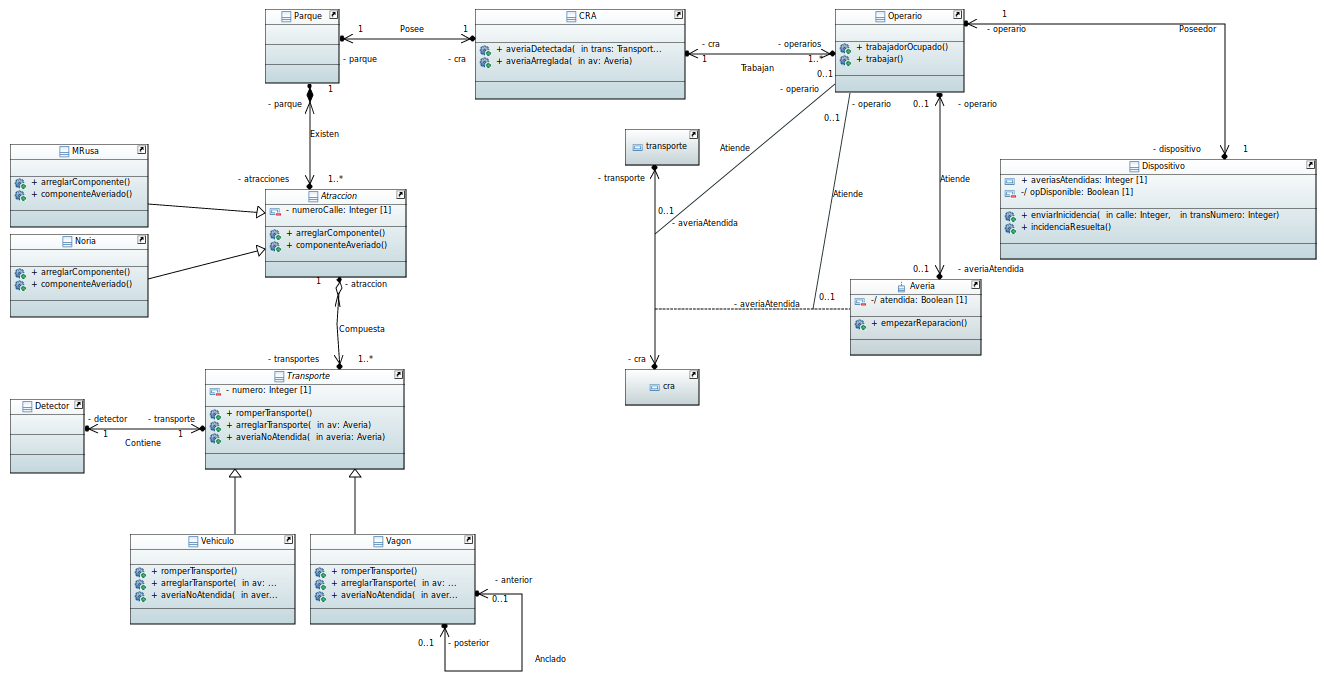
* Operario:



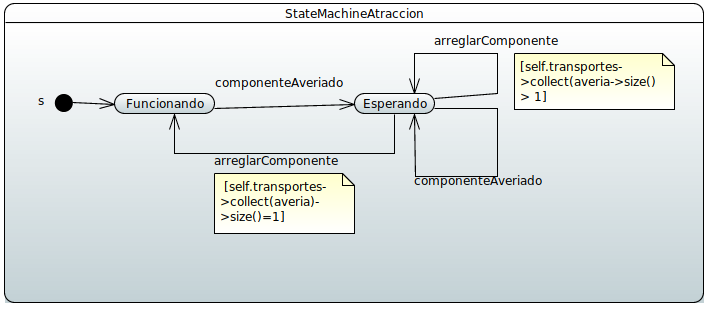
* Transporte:

## 

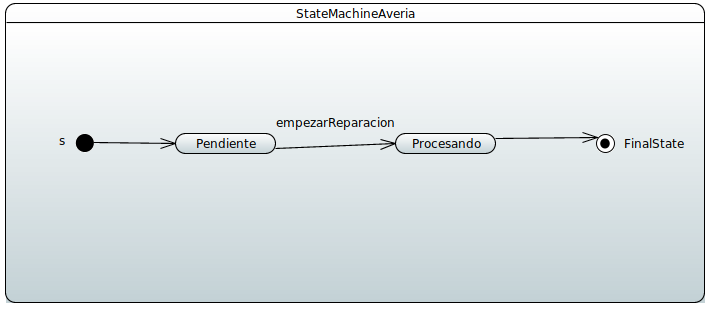
## **Modelo en Papyrus**



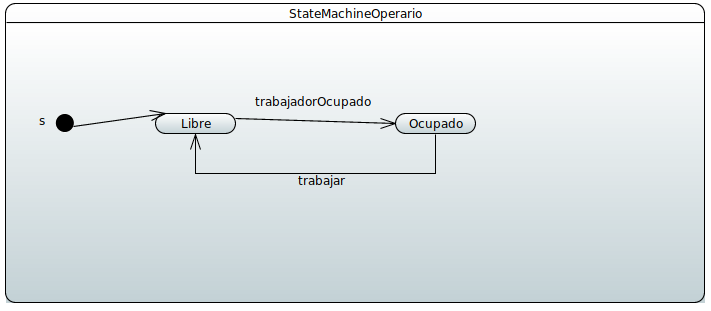
* Atracción:



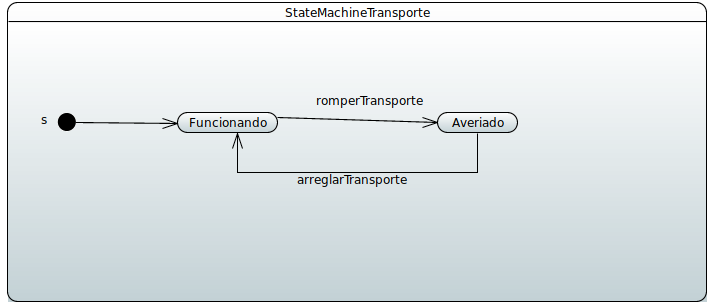
* Avería:



* Operario:



* Transporte:



## **Diagrama de secuencia main.soil+todoBien.soil**

**Main.soil**

reset

**open** TerraMiticaBueno.soil

**open** PortAventuraBueno.soil

**open** todoBien.soil

**PortAventuraBueno.soil**

!new Parque('PortAventura')

!new CRA('PortAventuraCRA')

!insert(PortAventura, PortAventuraCRA) into Posee

!new Noria('Ojo')

!Ojo.numeroCalle := 1

!insert(PortAventura, Ojo) into Existen

!new Vehiculo('Ve11')

!Ve11.numero := 1

!insert(Ojo, Ve11) into Compuesta

!new Detector('DVe11')

!insert(DVe11,Ve11) into Contiene

!new Vehiculo('Ve12')

!Ve12.numero := 2

!insert(Ojo, Ve12) into Compuesta

!new Detector('DVe12')

!insert(DVe12,Ve12) into Contiene

!new Vehiculo('Ve13')

!Ve13.numero := 3

!insert(Ojo, Ve13) into Compuesta

!new Detector('DVe13')

!insert(DVe13,Ve13) into Contiene

!new Vehiculo('Ve14')

!Ve14.numero := 4

!insert(Ojo, Ve14) into Compuesta

!new Detector('DVe14')

!insert(DVe14,Ve14) into Contiene

!new MRusa('Metal')

!Metal.numeroCalle := 2

!insert(PortAventura,Metal) into Existen

!new Vagon('Va11')

!Va11.numero := 1

!insert(Metal,Va11) into Compuesta

!new Detector('DVa11')

!insert(DVa11,Va11) into Contiene

!new Vagon('Va12')

!Va12.numero := 2

!insert(Metal,Va12) into Compuesta

!new Detector('DVa12')

!insert(DVa12,Va12) into Contiene

!new Vagon('Va13')

!Va13.numero := 3

!insert(Metal,Va13) into Compuesta

!new Detector('DVa13')

!insert(DVa13,Va13) into Contiene

!new Vagon('Va14')

!Va14.numero := 4

!insert(Metal,Va14) into Compuesta

!new Detector('DVa14')

!insert(DVa14,Va14) into Contiene

!insert(Va11,Va12) into Anclado

!insert(Va12,Va13) into Anclado

!insert(Va13,Va14) into Anclado

!new Operario('Ana')

!insert(PortAventuraCRA,Ana) into Trabajan

!new Operario('Marta')

!insert(PortAventuraCRA,Marta) into Trabajan

!new Dispositivo('D11')

!insert(Ana,D11) into Poseedor

!new Dispositivo('D12')

!insert(Marta,D12) into Poseedor

**TerraMiticaBueno.soil**

!new Parque('TerraMitica')

!new CRA('TerraMiticaCRA')

!insert(TerraMitica, TerraMiticaCRA) into Posee

!new Noria('Circulo')

!Circulo.numeroCalle := 1

!insert(TerraMitica, Circulo) into Existen

!new Vehiculo('Ve1')

!Ve1.numero := 1

!insert(Circulo, Ve1) into Compuesta

!new Detector('DVe1')

!insert(DVe1,Ve1) into Contiene

!new Vehiculo('Ve2')

!Ve2.numero := 2

!insert(Circulo, Ve2) into Compuesta

!new Detector('DVe2')

!insert(DVe2,Ve2) into Contiene

!new Vehiculo('Ve3')

!Ve3.numero := 3

!insert(Circulo, Ve3) into Compuesta

!new Detector('DVe3')

!insert(DVe3,Ve3) into Contiene

!new Vehiculo('Ve4')

!Ve4.numero := 4

!insert(Circulo, Ve4) into Compuesta

!new Detector('DVe4')

!insert(DVe4,Ve4) into Contiene

!new MRusa('Madera')

!Madera.numeroCalle := 2

!insert(TerraMitica,Madera) into Existen

!new Vagon('Va1')

!Va1.numero := 1

!insert(Madera,Va1) into Compuesta

!new Detector('DVa1')

!insert(DVa1,Va1) into Contiene

!new Vagon('Va2')

!Va2.numero := 2

!insert(Madera,Va2) into Compuesta

!new Detector('DVa2')

!insert(DVa2,Va2) into Contiene

!new Vagon('Va3')

!Va3.numero := 3

!insert(Madera,Va3) into Compuesta

!new Detector('DVa3')

!insert(DVa3,Va3) into Contiene

!new Vagon('Va4')

!Va4.numero := 4

!insert(Madera,Va4) into Compuesta

!new Detector('DVa4')

!insert(DVa4,Va4) into Contiene

!insert(Va1,Va2) into Anclado

!insert(Va2,Va3) into Anclado

!insert(Va3,Va4) into Anclado

!new Operario('Felipe')

!insert(TerraMiticaCRA,Felipe) into Trabajan

!new Operario('Emilio')

!insert(TerraMiticaCRA,Emilio) into Trabajan

!new Dispositivo('D1')

!insert(Felipe,D1) into Poseedor

!new Dispositivo('D2')

!insert(Emilio,D2) into Poseedor

**todoBien.soil**

**step on**

**-- Averías**

**!Va1.romperTransporte()**

**!Va12.romperTransporte()**

**!Ve3.romperTransporte()**

**!Ve13.romperTransporte()**

**-- Arreglan 3 y se queda una por arreglar**

**!Felipe.trabajar()**

**!Emilio.trabajar()**

**!Ana.trabajar()**

**-- Dos averias mas**

**!Va3.romperTransporte()**

**!Ve14.romperTransporte()**

**-- Se arreglan todas las averias**

**!Marta.trabajar()**

**!Ana.trabajar()**

**!Felipe.trabajar()**