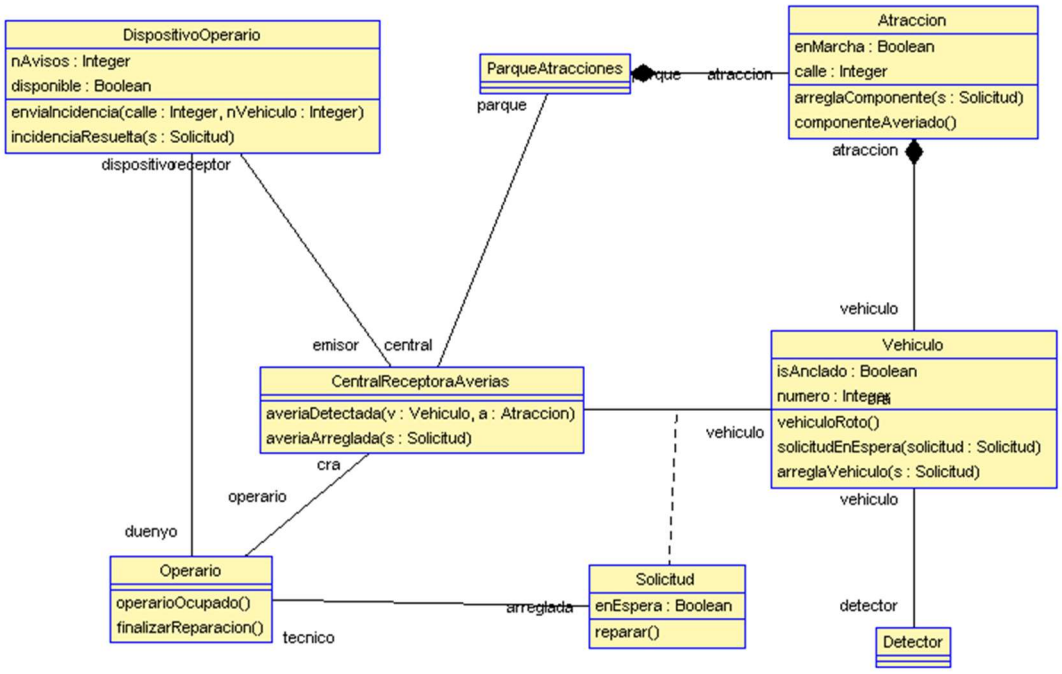
Modelado y Diseño de Software

Lucía Montiel Asensio

Práctica 2: CRA

USE

Diagrama de clases



Aclaraciones

La relación entre ParqueAtracciones y CentralReceptoraAverias tiene por las dos partes una multiplicidad de 1 ya que cada parque tiene solo una CRA y una CRa solo pertenece a un parque.

La relación entre Atraccion y Vehiculo es de composición ya que si una atracción desaparece, el vehículo también. Es lo mismo con Atraccion y ParqueAtracciones, si un parque desaparece, la atracción también.

Solicitud es una clase de asociación ya que es el resultado de relacionar la CRA con un vehículo.

Restricciones

```
--El operario que atiende la averia trabaja en la CRA a la que se asigna
context Solicitud inv OperarioTrabajaCRA: self.tecnico.cra = self.cra
```

```
--
La atraccion del vehiculo que manda la solicitud debe pertenecer a la mis
ma CRA donde se manda esta
context Solicitud inv CRAVehiculoIgualAveria: self.cra = self.vehiculo.at
raccion.parque
```

```
--El número del vehículo no es nulo
context Vehiculo inv nVehiculoDefinido: not self.numero.oclIsUndefined()
```

Operaciones

CentralReceptoraAverias

```
averiaDetectada(v: Vehiculo, a: Atraccion)
        begin
            declare s: Solicitud, op: Operario;
            if v.solicitud -> isEmpty() then
                s := new Solicitud between (v, self);
                if self.operario->select(op | op.dispositivo.disponible)-
>notEmpty() then
                op := self.operario->any(op | op.dispositivo.disponible);
                op.dispositivo.enviaIncidencia(a.calle, v.numero);
            else
                v.solicitudEnEspera(s);
            end
        else
            s := v.solicitud;
            if self.operario->select(op | op.dispositivo.disponible)-
>notEmpty() then
                op := self.operario->any(op | op.dispositivo.disponible);
                op.dispositivo.enviaIncidencia(a.calle, v.numero);
            end
        end
    end
    pre atraccionContieneVehiculo: a.vehiculo->includes(v)
    pre mismoParque: a.parque = self.parque
    post incrementoSolicitudes: Solicitud.allInstances-
>select(so | so.oclIsNew())->notEmpty() implies
     Solicitud.allInstances->size() = Solicitud.allInstances@pre-
>size() + 1
```

Esta operación permite detectar una avería, buscando así un operario libre que pueda atenderla. Las precondiciones aseguran que la atracción pasada como parámetro contenga al vehículo, además de que pertenezca al mismo parque. La postcondición incrementa el número de solicitudes.

```
averiaArreglada(s : Solicitud)
        begin
            destroy(s)
        end
        pre existeAveria: self.parque.atraccion.vehiculo -
> collect(solicitud)->includes(s)
        post sinSolicitud: (self.vehiculo->select(v | v.solicitud-
>notEmpty())->size() =
        self.vehiculo->select(v | v.solicitud@pre->notEmpty())-
>size() - 1)
```

Esta operación elimina la solicitud de reparación, primero comprueba que haya solicitudes y luego comprueba que el número de solicitudes se ha decrementado.

Atraccion

```
componenteAveriado()
    begin
    end
    pre existeSolicitud: self.vehiculo->exists(v | v.solicitud-
>notEmpty())
```

Esta función comprueba que algún vehículo de la atracción ha enviado una solicitud.

```
arreglaComponente(s : Solicitud)
    begin
        destroy(s);
    end
    pre hayAveria: self.vehiculo->collect(solicitud)-
>includes(s) and not self.enMarcha
    post sinAveria:
        (self.vehiculo->select(v | v.solicitud->notEmpty())->size() =
        self.vehiculo->select(v | v.solicitud@pre->notEmpty())-
>size() - 1) and self.enMarcha
```

Esta función elimina la solicitud del vehículo una vez arreglada la avería.

Vehiculo

```
vehiculoRoto()
        begin
            self.atraccion.parque.central.averiaDetectada(self, self.atra
ccion);
            self.atraccion.componenteAveriado();
    end
```
Esta función hace que el vehículo se rompa, llamando así a otras funciones.

```
solicitudEnEspera(solicitud: Solicitud)
    begin
        while (solicitud.enEspera) do
            self.atraccion.parque.central.averiaDetectada(self, self.atra
ccion);
        end
    end
    pre haySolicitud: self.solicitud ->notEmpty()
```

En esta operación se comprueba que la solicitud sigue en espera, no se sale del bucle hasta que esta esté atendida.

```
arreglaVehiculo(s: Solicitud)
    begin
        self.atraccion.arreglaComponente(s);
    end
    pre existeSolicitud: self.solicitud->notEmpty
    pre averiaAtendida: not self.solicitud.enEspera
    post solicitudEliminada: self.solicitud-
>isEmpty() and Solicitud.allInstances->excludes(s)
```

Esta operación llama a la función de la clase Atraccion y comprueba que antes de ejecutarse hay solicitudes y esta está atendida. Como postcondición se comprueba que la solicitud pasada como parámetro no existe ya.

Solicitud

```
reparar()
    begin
    end
    pre existeOperarioAtendiendo: not self.enEspera
```

Comprueba que la solicitud no esté en espera.

Operario

```
operarioOcupado()
    begin
        self.arreglada.reparar();
    end
    pre existeSolicitud: self.arreglada->notEmpty()
```

Empieza la reparación del vehículo y comprueba que el operario lo está arreglando.

```
finalizarReparacion()
    begin
        self.dispositivo.incidenciaResuelta(self.arreglada);
    end
    pre ocupado: not self.dispositivo.disponible
    post operarioLibre: self.dispositivo.disponible
```

DispositivoOperario

```
enviaIncidencia(calle: Integer, nVehiculo: Integer)
    begin
        declare s : Solicitud;
        s := Solicitud.allInstances-
>select(s | s.vehiculo.numero = nVehiculo
            and s.vehiculo.atraccion.calle = calle)->asSequence()-
>first();
        insert (self.duenyo, s) into Atiende;
        self.duenyo.operarioOcupado();
    end
    pre solicitudEnviada: Solicitud.allInstances->collect(vehiculo)-
>asSet()
                        -
>exists(v | v.numero = nVehiculo and v.atraccion.calle = calle)
    post trabajadorOcupado: let s: Solicitud = Solicitud.allInstances->
                        select(s | s.vehiculo.numero = nVehiculo
                        and s.vehiculo.atraccion.calle = calle and
                        self.duenyo.cra.parque.atraccion-
>includes(s.vehiculo.atraccion))
                        ->asSequence()->first() in
                        not self.disponible and self.duenyo.arreglada = s
```

Esta función envía un mensaje con la calle y el número del vehículo al dispositivo del operario y relaciona a este con la solicitud de reparación del vehículo en cuestión. Como precondición comprueba que la calle y el vehículo existan. Y como postcondición comprueba que el trabajador esté ocupado.

```
incidenciaResuelta(s : Solicitud)
    begin
        declare cra: CentralReceptoraAverias;
        cra := self.duenyo.cra;
        cra.averiaArreglada(s);
        self.nAvisos := self.nAvisos + 1;
    end
    pre trabajadorOcupado: not self.disponible
    pre trabajadorAtiendeAveria: self.duenyo.arreglada = s
    post averiasResueltasIncrementada: self.nAvisos = self.nAvisos@pre +
1
    post trabajadorLibre: self.disponible
```

Esta función da por resuelta la reparación de la avería, incrementando el número de avisos del operario. Como precondición comprueba que el operario está ocupado y que atiende la avería. La postcondición hace que el número de avisos se incremente y qe el trabajador vuelva a estar disponible.

Diagramas de estado

Diagrama de estados de Atraccion

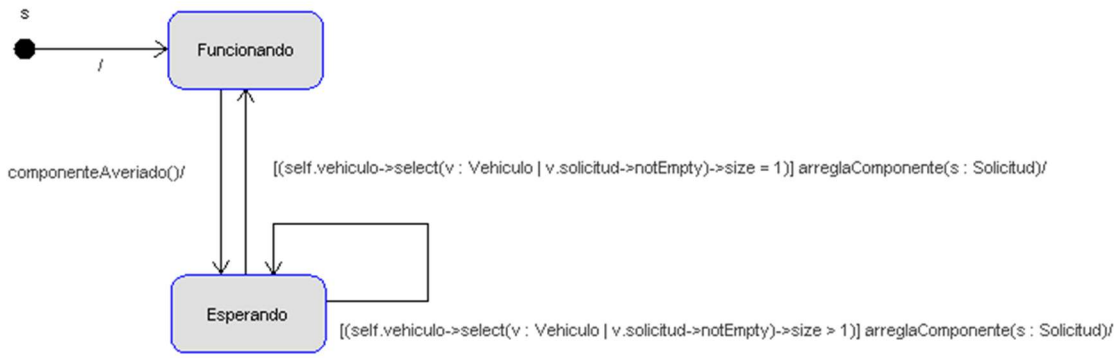


Diagrama de estados de Solicitud

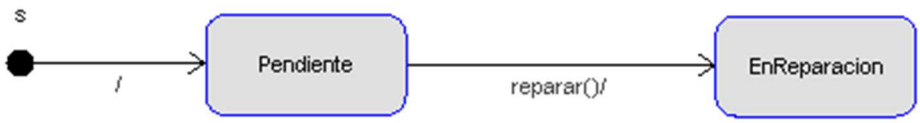


Diagrama de estados de Vehiculo

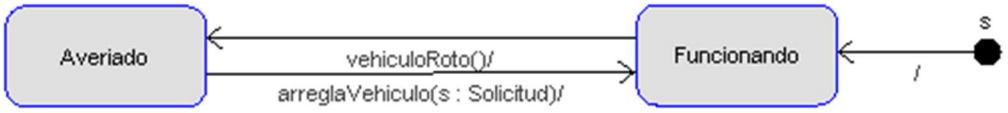


Diagrama de estados de Operario

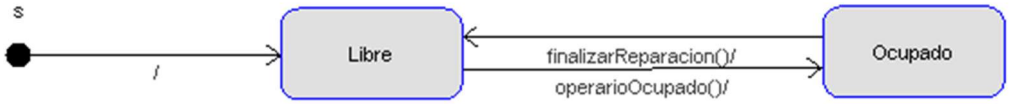
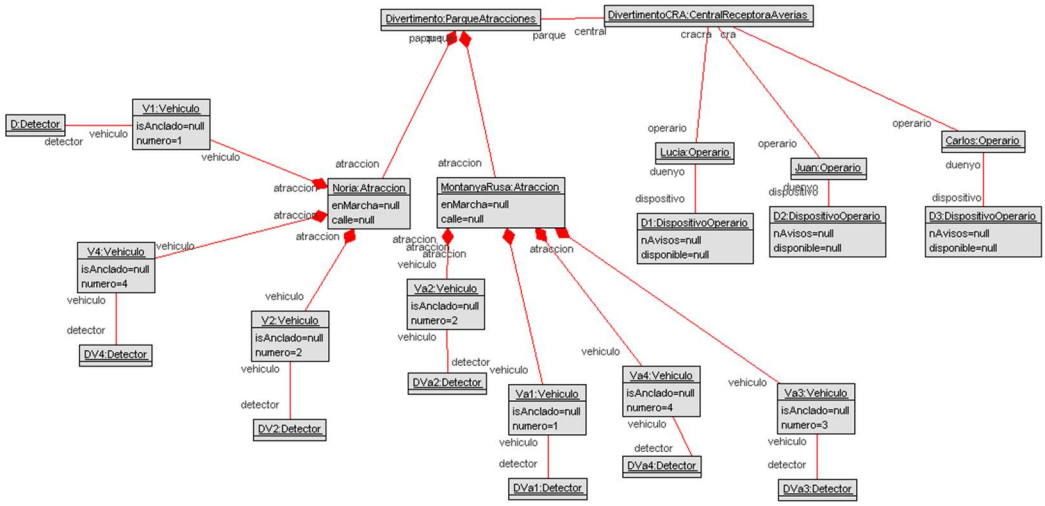
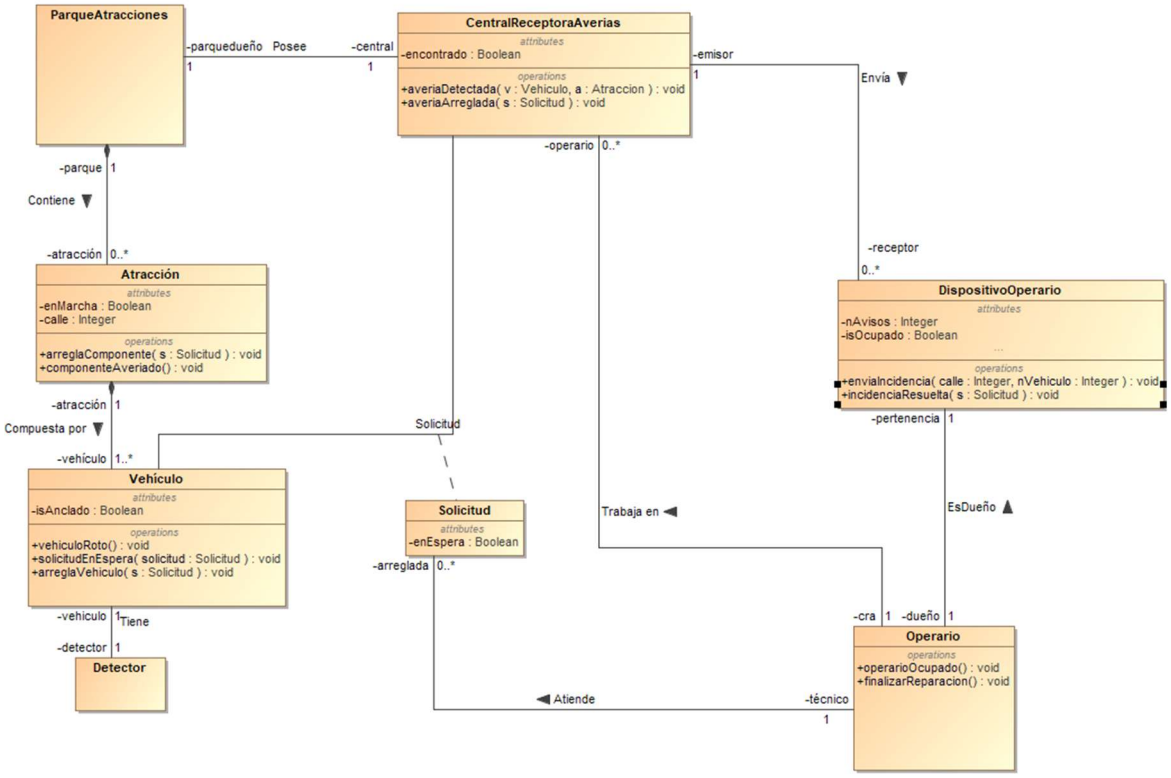


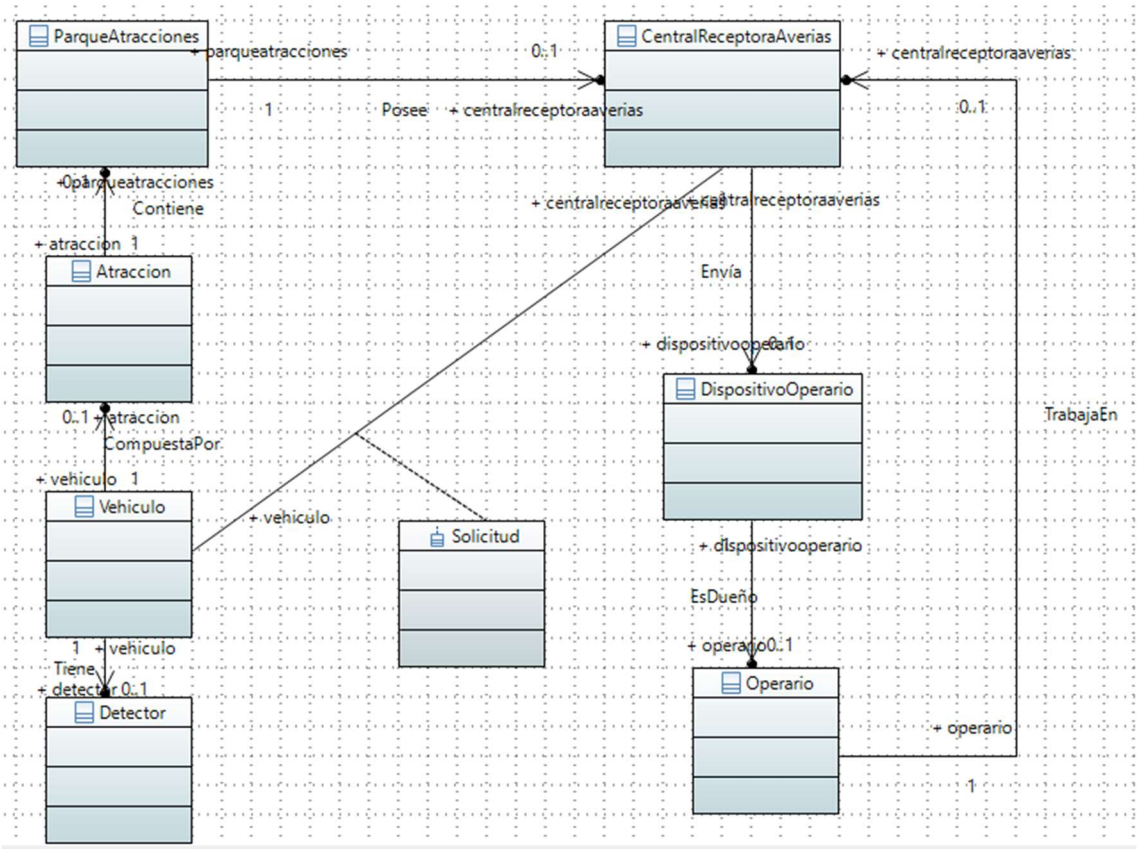
Diagrama de objetos



MagicDraw


ParqueAtracciones
-parquedueño
Posee
-central
1
1
CentralReceptoraAverias
attributes
-encontrado : Boolean
operations
+averiaDetectada(v : Vehiculo, a : Atraccion) : void
+averiaArreglada(s : Solicitud) : void
-emisor
1
Envía
-operario
0..*
-parque
1
Contiene
-atracción
0..*
Atracción
attributes
-enMarcha : Boolean
-calle : Integer
operations
+arreglaComponente(s : Solicitud) : void
+componenteAveriado() : void
-receptor
0..*
DispositivoOperario
attributes
-nAvisos : Integer
-isOcupado : Boolean
...
operations
+enviaIncidencia(calle : Integer, nVehiculo : Integer) : void
+incidenciaResuelta(s : Solicitud) : void
-pertenencia
1
-atracción
1
Compuesta por
Solicitud
-vehículo
1..*
Vehiculo
attributes
-isAnclado : Boolean
operations
+vehiculoRoto() : void
+solicitudEnEspera(solicitud : Solicitud) : void
+arreglaVehiculo(s : Solicitud) : void
Solicitud
attributes
-enEspera : Boolean
Trabaja en
EsDueño
-arreglada
0..*
-vehiculo
1
Tiene
-detector
1
Detector
-cra
1
-dueño
1
Operario
operations
+operarioOcupado() : void
+finalizarReparacion() : void
Atiende
-técnico
1

Papyrus


ParqueAtracciones
+ parqueatracciones
0..1
CentralReceptoraAverias
+ centralreceptoraaverias
1
Posee
+ centralreceptoraaverias
0..1
+ parqueatracciones
Contiene
+ centralreceptoraaverias
+ centralreceptoraaverias
+ atraccion 1
Atraccion
Envía
+ dispositivooperario
DispositivoOperario
0..1 + atraccion
CompuestaPor
TrabajaEn
+ vehiculo 1
Vehiculo
+ vehiculo
Solicitud
+ dispositivooperario
EsDueño
1 + vehiculo
Tiene
+ operario 0..1
+ detector 0..1
Detector
Operario
+ operario
1