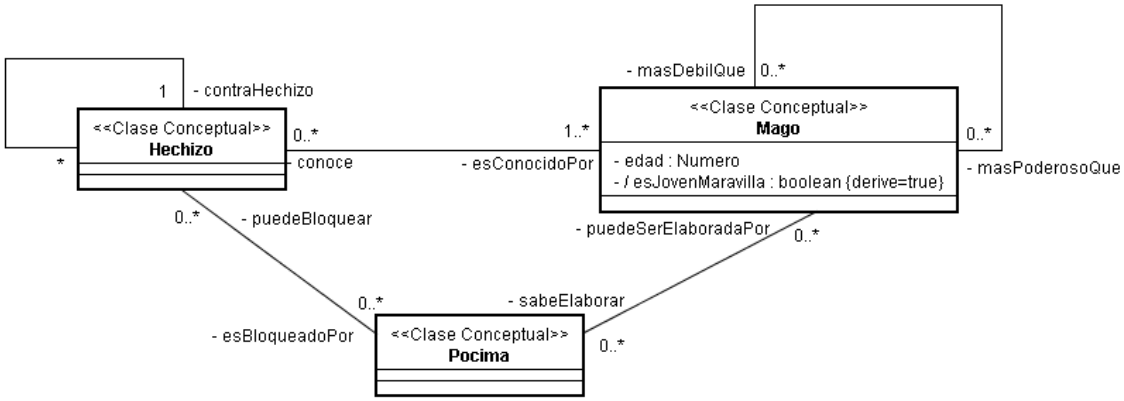
Enunciado

Modelo conceptual y OCL

Dado el siguiente modelo conceptual:



1. Modelar con OCL los siguientes puntos:

   a. Un invariante que garantice que un mago sea más poderoso que otro, sólo si conoce algún hechizo tal que el otro mago no conoce su contrahechizo.

   b. Un invariante que garantice que ninguna pócima puede bloquear a un hechizo y a su contrahechizo.

   c. Un invariante que garantice que ningún mago puede elaborar una pócima que bloquee a todos los hechizos conocidos (lo cual no implica que dicha pócima no exista).

   d. El atributo derivado esJovenMaravilla. Se dice que un mago es un joven maravilla, cuando es el mago más joven que más hechizos conoce. Es decir, de aquellos magos que poseen la menor edad, el que conoce más hechizos.

2. Modificar el modelo (agregando / quitando entidades, relaciones y / o predicados en OCL) de manera tal de contemplar el siguiente texto:

   a. "El bloqueo de un hechizo por parte de una pócima, no es indefinido. Según cuál sea el hechizo, el efecto de bloqueo tiene una duración determinada.

   b. Las pócimas están compuestas por ingredientes, que las componen con ciertas cantidades. Los ingredientes pueden ser de origen animal, vegetal o mineral. Aquellas pócimas que están compuestas sólo por ingredientes de origen mineral, no son capaces de bloquear a ningún hechizo.

   c. Existen magos que no son capaces de elaborar ninguna pocima, y sólo son capaces de realizar hechizos. Estos magos son conocidos como hechiceros."

Resolución

Aclaraciones sobre el enunciado

- Las pócimas que bloquean hechizos inhiben que un hechizo pueda realizarse. Una vez que el hechizo está realizado, la forma de cortarlo es con un contrahechizo.
- Un mago que conoce un hechizo, puede realizar ese hechizo.

Analizando el modelo...

A modo de análisis preliminar, miremos un poco el modelo y veamos algunas de las reglas de dominio que surgen de él.

- HECHIZO
  - Todo hechizo tiene que ser conocido al menos por un mago
  - Todo hechizo tiene un único contrahechizo.
  - Un hechizo puede no ser contrahechizo de ningún hechizo.
  - Un hechizo puede ser contrahechizo de muchos hechizos
  - Un hechizo que es contrahechizo, puede a su vez tener un contrahechizo.
  - Un hechizo puede ser contrahechizo de sí mismo
  - Un hechizo puede no ser bloqueado por ninguna pócima
- MAGO
  - Un mago puede conocer un hechizo que es contrahechizo de otro, sin conocer el hechizo primario.
  - Un mago puede no conocer hechizos ni saber elaborar pócimas
  - Un mago puede no ser más poderoso ni más debil que otro
  - Un mago puede ser más poderoso que sí mismo
- POCIMA
  - Una pócima puede no ser elaborada por ningún mago
  - Una pócima puede no bloquear ningún hechizo.
  - Un mago puede saber una pócima que bloquee hechizos que no conoce

Pasemos al ejercicio

1.a. Modelar con OCL un invariante que garantice que un mago sea más poderoso que otro, sólo si conoce algún hechizo tal que el otro mago no conoce su contrahechizo.

Vamos a pensarlo a partir de la clase conceptual ***Mago***

Primera aproximación:
Definamos

- ***hechizosConocidos***(m) como el conjunto de los hechizos conocidos de un mago m
- **contraHechizosDeHechizosConocidos**(m) como el conjunto de los hechizos que son contrahechizos de un hechizo conocido por el mago m.

El invariante buscado debería garantizar que para todo par de magos (p, d) tal que p es más poderoso que d, existe un contrahechizo de un hechizo conocido por p que no es conocido por d, es decir, ***contraHechizosDeHechizosConocidos***(p) no está incluido en ***hechizosConocidos***(d).

Formalizando un poco con notación OCL
hechizosConocidos(m) := m.conoce
contraHechizosDeHechizosConocidos(m) := m.conoce.contraHechizo→asSet()

Llegamos al OCL
context Mago
inv: self.masPoderosoQue → forAll(d | not (d.conoce → includesAll (self.conoce.contraHechizo→asSet())))

Notar que es necesario utilizar asSet() ya que self.conoce.contraHechizo es un bag

1.b. Modelar con OCL un invariante que garantice que ninguna pócima puede bloquear a un hechizo y a su contrahechizo.

Vamos a pensarlo a partir de la clase conceptual ***Pócima***

Primera aproximación:
Definamos

- ***hechizosBloqueados***(p) como el conjunto de los hechizos que son bloqueados por la pócima p
- **contraHechizosDeHechizosBloqueados**(p) como el conjunto de los hechizos que son contrahechizos de un hechizo bloqueado por la pócima p.

El invariante buscado debería garantizar que para toda pócima p, no existe ningún contrahechizo de un hechizo bloqueado por p que sea bloqueado por p, es decir, la intersección de ***contraHechizosDeHechizosBloqueados***(p) y ***hechizosBloqueados***(d) es vacía.

Formalizando un poco con notación OCL
hechizosBloqueados(p) := p.puedeBloquear
contraHechizosDeHechizosBloqueados(p) := p.puedeBloquear.contraHechizo→asSet()

Llegamos al OCL
context Pócima
inv: self.puedeBloquear → excludesAll(self.puedeBloquear.contraHechizo→asSet())

¿Podríamos pensarlo a partir de la clase conceptual ***Hechizo?***

Primera aproximación:
Definamos

- ***pocimasBloqueadoras***(h) como el conjunto de las pócimas que bloquean al hechizo h.
- ***pocimasBloqueadorasContraHechizo***(h) como el conjunto de las pócimas que bloquean al contrahechizo del hechizo h.

El invariante buscado debería garantizar que para todo hechizo h, no existe ninguna pócima que lo bloquee a él y a su contrahechizo, es decir, la intersección de ***pocimasBloqueadoras***(h) y ***pocimasBloqueadorasContraHechizo***(h) es vacía.

Formalizando un poco con notación OCL
pocimasBloqueadoras(h) := h.esBloqueadoPor
pocimasBloqueadorasContraHechizo(h) := h.contraHechizo.esBloqueadoPor

Llegamos al OCL
context Hechizo
inv: self. esBloqueadoPor → excludesAll(self. contraHechizo.esBloqueadoPor)

Notar que en este caso no utilizamos asSet() ya que self.contraHechizo.esBloqueadoPor ya es un set a causa de que la multiplicidad de contraHechizo es exactamente 1

1.c. Modelar con OCL un invariante que garantice que ningún mago puede elaborar una pócima que bloquee a todos los hechizos conocidos (lo cual no implica que dicha pócima no exista).

Al referirse a los hechizos conocidos se habla de todos los hechizos conocidos por algún mago, es decir, todos los hechizos existentes (recordemos que todo hechizo es conocido por al menos un mago)

Vamos a pensarlo a partir de la clase conceptual ***Mago***

Primera aproximación:
Definamos

- ***pócimasElaboradas***(m) como el conjunto de las pócimas a las que el mago m sabe elaborar
- ***hechizosBloqueados***(p) como el conjunto de los hechizos que son bloqueados por la pócima p
- **bloqueaTodo** (p), que representa si una pócima p bloquea todos los hechizos existentes.
- **hechizosTotales**() como el conjunto de todos los hechizos que existen.

El invariante buscado debería garantizar que para todo mago m, no existe ninguna pócima que sepa elaborar que bloquee a todos los hechizos que existen, es decir, si p es una pócima de ***pócimasElaboradas***(m), no se cumple que ***bloqueaTodo***(p).

Formalizando un poco con notación OCL
pócimasElaboradas(m) := m.sabeElaborar
hechizosBloqueados(p) := p.puedeBloquear
hechizosTotales() := Hechizo.allInstances()
bloqueaTodo(p) := p.puedeBloquear→includesAll(Hechizo.allInstances())

Llegamos al OCL
context Mago
inv: self. sabeElaborar → forAll (p | not (p.puedeBloquear→includesAll(Hechizo.allInstances())))

1.d. Modelar con OCL el atributo derivado *esJovenMaravilla*. Se dice que un mago es un joven maravilla, cuando es el mago más joven que más hechizos conoce. Es decir, de aquellos magos que poseen la menor edad, el que conoce más hechizos.

Vamos a pensarlo a partir de la clase conceptual ***Mago***

Primera aproximación:
Definamos

- ***cantHechizosConocidos***(m) como la cantidad de hechizos que son conocidos por el mago m.
- **minEdadMago** () como la menor edad de un mago.
- ***magosJovenes***() como el conjunto de los magos que poseen la menor edad.
- **maxCantHechizosMagoJoven**(), como la cantidad máxima de hechizos que conoce un mago joven. Es decir, es el máximo de CantHechizosConocidos(m) para m perteneciente a MagosJovenes().

El invariante buscado debería definir que un mago m es joven maravilla si pertenece a ***magosJovenes***() y ***cantHechizosConocidos***(m)=***maxCantHechizosMagoJoven***()

Formalizando un poco con notación OCL
cantHechizosConocidos () := m.conoce→size()
minEdadMago() := Mago.allInstances()→min()
magosJovenes() := Mago→ select (m | m.edad = Mago.allInstances().edad→min())
maxCantHechizosMagoJoven () :=
((Mago → select (m | m.edad = Mago.allInstances().edad→min())) →
collect(j | j.conoce→size())) → max()

Llegamos al OCL
context Mago::esJovenMaravilla : boolean
derive: self.edad = Mago.allInstances()→min())
and self.conoce→size() = ((Mago → select (m | m.edad =
Mago.allInstances()→min())
) → collect(j | j.conoce→size())
) → max()

Una forma más elegante:
Definamos

- ***cantHechizosConocidos***(m) como la cantidad de hechizos que son conocidos por el mago m.

El invariante buscado debería definir que un mago m es joven maravilla si su edad es menor o igual que la de todos los demás magos , y además si hay otro mago de igual edad, entonces el otro mago no conoce más hechizos que m.

Escribámoslo en OCL
context Mago::esJovenMaravilla : boolean
derive: Mago.allInstances()→forAll(m |
(self.edad <= m.edad)
and (self.edad = m.edad
implies self.conoce→size() >= m.conoce->size())
)

Notar que self es de tipo Mago por estar dentro del contexto de la clase conceptual Mago y no por estar dentro de un forAll de un conjunto de elementos de tipo Mago.

Un ejemplo para aclarar este último comentario
Supongamos que tenemos un atributo ***antiguedad*** en la clase Pocima (perteneciente al dominio numero) y queremos modelar en OCL un atributo derivado de Pocima llamado ***esPocimaAncestral*** que sea de tipo boolean. Se dice que una pócima es una pócima ancestral si su antigüedad es mayor a la edad de todos los magos existentes.

En este caso, nuestra especificación OCL quedaría:

context Pocima::esPocimaAncestral : boolean
derive: Mago.allInstances()→forAll(m |
(self.antiguedad <= m.edad)
)

Notar que self claramente es de tipo Pocima, que es la clase del contexto.
2.a. Modificar el modelo (agregando / quitando entidades, relaciones y / o predicados en OCL) de manera tal de contemplar el siguiente texto:

> ***"El bloqueo de un hechizo por parte de una pócima, no es indefinido. Según cuál sea el hechizo, el efecto de bloqueo tiene una duración determinada."***

La idea general es mantener un atributo que indique la duración del bloqueo.

Podríamos pensar en que sea un atributo de las pócimas, pero en realidad con ello no modelaríamos el requerimiento, ya que de esa forma cada pócima tendría una duración **independiendiente** del hechizo que bloquea. Tampoco puede ser un atributo de los hechizos, ya que en ese caso la duración sería **independiente** de la pócima que lo bloquea.

La solución es agregar una clase de asociación a puedeBloquear/esBloqueadoPor.
Llamaremos a la clase ***Bloqueo*** y tendrá un atributo llamado ***duracionBloqueo*** perteneciente al dominio ***numero***

2.b. Modificar el modelo (agregando / quitando entidades, relaciones y / o predicados en OCL) de manera tal de contemplar el siguiente texto:

> ***"Las pócimas están compuestas por ingredientes, que las componen con ciertas cantidades. Los ingredientes pueden ser de origen animal, vegetal o mineral. Aquellas pócimas que están compuestas sólo por ingredientes de origen mineral, no son capaces de bloquear a ningún hechizo."***

La idea general es agregar una nueva clase conceptual ***Ingrediente***. Esta clase tendrá una asociación con ***Pocima.*** La modelaremos como una ***agregación***, ya que entendemos que los ingredientes tienen existencia independientemente de su inclusión en una pócima***.*** La multiplicidad del rol ***estaCompuestoPor*** es 1..*. La multiplicidad de ***componeA*** es 0..*.

Pero con esto no alcanza para modelar el requerimiento. Nos falta modelar el origen.
Podriamos pensarlo como una nueva clase conceptual ***Origen*** con una asociación hacia ingrediente. También podríamos pensarlo como una partición de la clase conceptual ***Ingrediente***: ***IngredienteOrigenAnimal, IngredienteOrigenVegetal, IngredienteOrigenMineral.*** Ambas modelan de una manera diferente. Pero entendemos en principio que las dos le dan una relevancia mayor a los orígenes de la que surge del planteo del problema. Por lo tanto, lo vamos a modelar como un atributo de la clase ***Ingrediente*** que llamaremos ***Origen*** y pertenece al dominio enumerado ***Origen : {Animal, Vegetal, Mineral}.***

Ninguna de las 3 variantes mencionadas nos da ventajas para modelar la restricción adicional directamente en el diagrama. Por lo tanto, lo vamos a modelar con OCL, asumiendo la última variante.

Primera aproximación
Definamos

- ***pocimaMineral***(p) que representa que una pócima p sólo está formada por ingredientes de origen mineral.
- ***hechizosBloqueados***(p) como el conjunto de los hechizos que son bloqueados por la pócima p.

El invariante buscado debería definir que si una pócima p sólo tiene atributos minerales entonces no puede bloquear hechizos, es decir, para toda pócima vale que si ***pocimaMineral***(p) entonces ***hechizosBloqueados***(p) es vacío.

Formalizando un poco con notación OCL

pocimaMineral(p) := p.estaCompuestoPor.Origen → asSet() = Set {Mineral}
hechizosBloqueados (p) := p.puedeBloquear

Llegamos al OCL
context Pocima
inv: self.estaCompuestoPor.Origen → asSet() = Set {Mineral}
implies self.puedeBloquear→empty()

2.c. Modificar el modelo (agregando / quitando entidades, relaciones y / o predicados en OCL) de manera tal de contemplar el siguiente texto:

"Existen magos que no son capaces de elaborar ninguna pocima, y sólo son capaces de realizar hechizos. Estos magos son conocidos como hechiceros."

Diferentes variantes a analizar:

- Modificar el diagrama agregando una partición de ***Mago***, como ***MagoHechicero*** / ***MagoNoHechicero***. La asociación con pócima debería llevarse desde ***Mago*** a ***MagoNoHechicero***, y la multiplicidad de ***sabeElaborar*** debería cambiarse a 1..*. *Notar que aquí estamos diciendo que algunos magos son hechiceros.*

- Modificar el diagrama agregando una nueva clase conceptual Hechicero de la cual ***Mago*** herede. La asociación con ***Hechizo*** debería llevarse desde ***Mago*** a ***Hechicero***, y además la multiplicidad de la otra relación en el rol ***sabeElaborar*** debería cambiarse a 1..*. *Notar que aquí estamos diciendo que todos los magos son hechiceros.*

- Modificar el diagrama agregando una subclase que herede de ***Mago***, llamada ***Hechicero***, y sin cambiar las asociaciones existentes. Esto así no nos permite representar lo que nos piden. Podríamos llegar a hacerlo complementándolo con alguna restricción OCL, que garantice que si un mago es de clase Hechicero, entonces la cantidad de pócimas que puede realizar es 0.

  Context: Mago
  Inv: self.OCLisTypeOf(Hechicero) implies self.sabeElaborar→empty()

  Notar que aquí estamos complejizando la lectura de nuestro modelo, ya que estamos perdiendo de representar el requerimiento en el diagrama para llevarlo a OCL