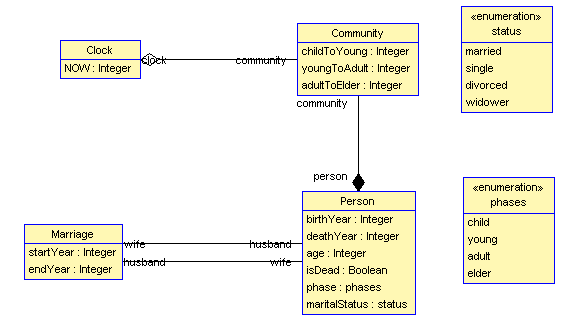
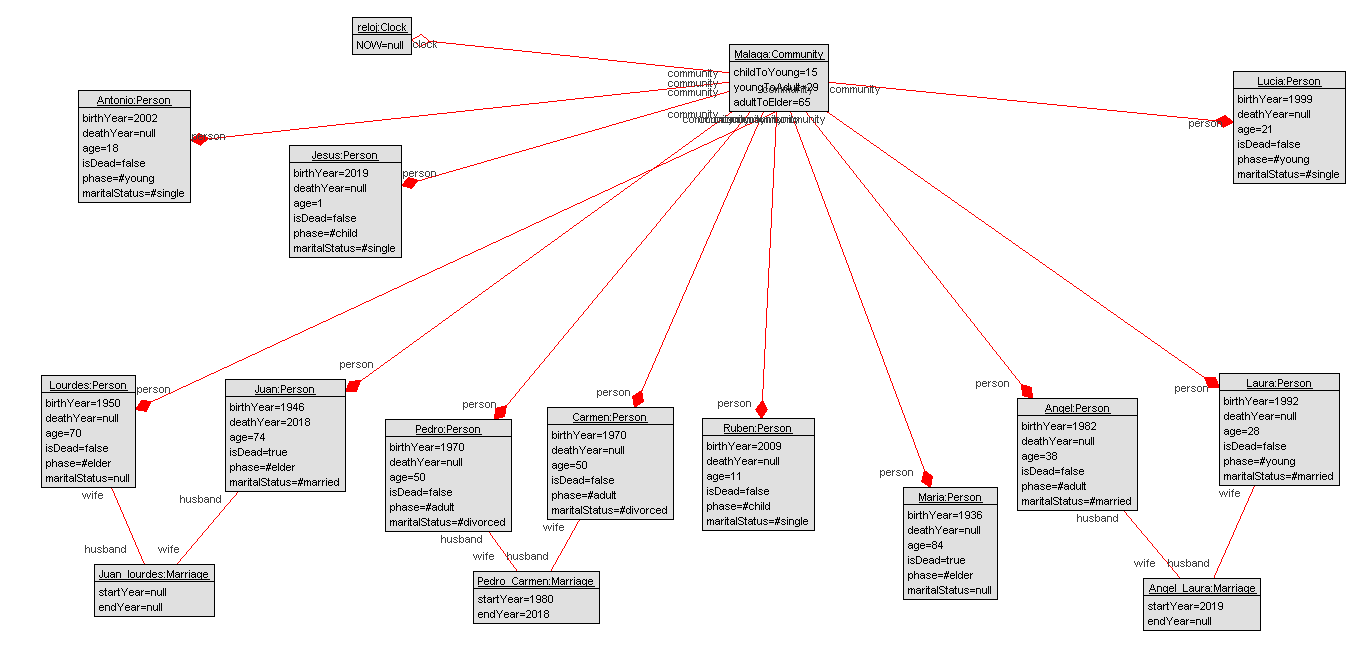
Ejercicio Tema 3

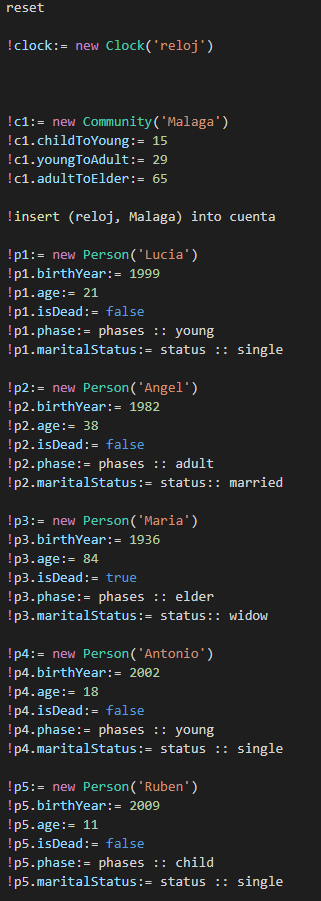
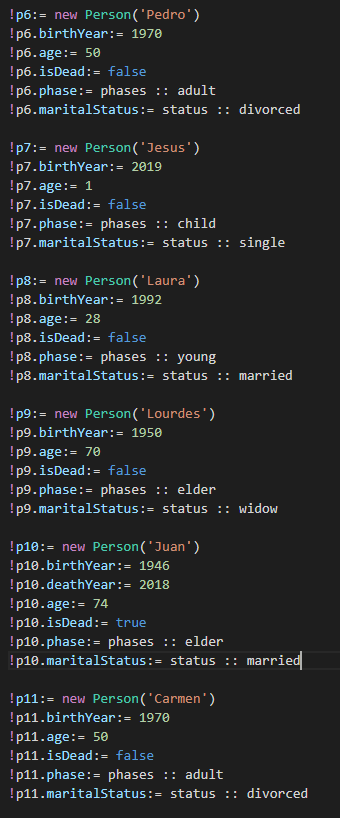
El diagrama de clases que he creado para este modelo es:



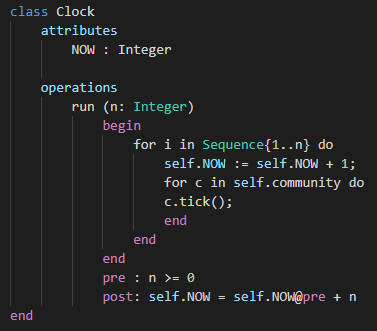
Al diagrama dado en el enunciado le he añadido atributos que completan el modelo, como por ejemplo las edades en las cuales cada comunidad pasa de etapa de la vida.

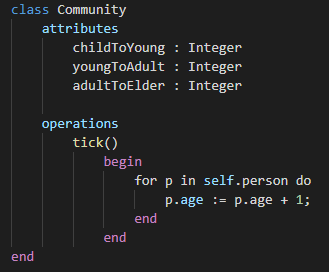
Según este diagrama de clases se puede modelar el siguiente diagrama de objetos:



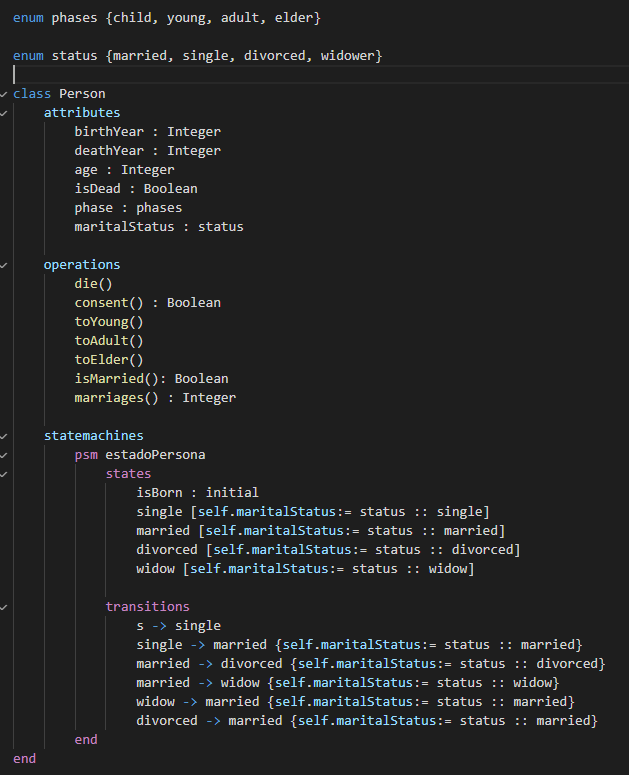
El código del archivo .soil que he escrito para el diagrama de objetos es:

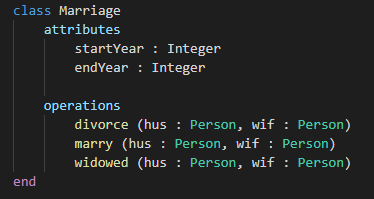
El código con el que he modelado las clases es:

Para la clase Clock he declarado un método que hace que el tiempo pase, incrementando el atributo “NOW”.

Para la clase Community, por cada vez que se incremente el valor del reloj, el método tick() aumenta la edad de las personas de la comunidad:

La clase Person tiene como atributos dos enumeraciones, “phases” y “status”, que representan las fases de la vida y el estado civil. Y una máquina de estados que ilustra el cambio de estados civiles de las personas.



La clase Marriage contiene los métodos que permiten a dos personas casarse y divorciarse, además del que hace viudo a una persona:

Las precondiciones y postcondiciones de los métodos:

context Marriage :: divorce(hus : Person, wif : Person)

    pre: self.wife -> select (w | w = wif) -> notEmpty()

    and self.husband -> select (h | h = hus) -> notEmpty() and self.endYear.oclIsUndefined()

    post: self.endYear = self.husband.community.clock.NOW and hus.maritalStatus:= status :: divorced and wif.maritalStatus:= status :: divorced and hus.consent() = true and wif.consent() = true

context Person :: marry (hus : Person, wif : Person)

    pre : self.wife -> select (w | w = wif) -> isEmpty() and self.husband -> select (h | h = hus) -> isEmpty()

post : self.startYear = self.husband.community.clock.NOW and hus.maritalStatus:= status :: married and wif.maritalStatus:= status :: married and hus.consent() = true and wif.consent() = true

context Person :: die()

    pre : self.isDead = false and self.deathYear.oclIsUndefined()

    post : self.isDead = true and self.deathYear = self.community.clock.NOW

context Person :: toYoung()

    pre : self.community.clock.NOW - self.birthYear = self.community.childToYoung and self.phase :: child

    post : self.phase :: young

context Person :: toAdult()

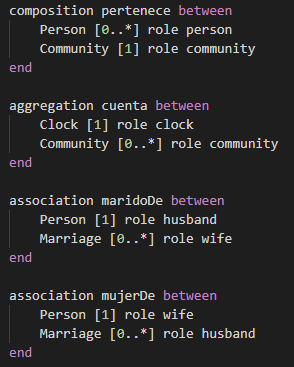
    pre : self.community.clock.NOW - self.birthYear = self.community.youngToAdult and self.phase :: young

    post : self.phase :: adult

context Person :: toElder()

    pre : self.community.clock.NOW - self.birthYear = self.community.adultToElder and self.phase :: adult

    post : self.phase :: elder

Las relaciones:

Las precondiciones y postcondiciones que he visto necesarias para el buen funcionamiento del modelo son:

constraints

--aleticas

--Una persona no puede fallecer antes de nacer

context Person inv fechaNacimientoMenorQueFallecimiento: self.birthYear < self.deathYear

--Un matrimonio no puede acabar antes de empezar

context Marriage inv fechaInicioMenorQueTermino: self.startYear < self.endYear

--Los muertos no pueden estar casados.

context Person inv siMuertoNoCasado: self.isDead = true

implies ((self.wife -> select (w | w.endYear.oclIsUndefined) -> size() = 0) and

(self.husband -> select (h | h.endYear.oclIsUndefined) -> size() = 0))

--No puede haber matrimonios futuros (“acordados”): Clock.NOW tiene que ser siempre mayor o igual que

--cualquier fecha de comienzo de un matrimonio

context Marriage inv matrimonioNoAcordado: self.startYear < self.husband.community.clock.NOW

--No se puede nacer en el futuro: Clock.NOW tiene que ser mayor o igual que cualquier fecha de nacimiento.

context Community inv fechaNacimiento: self.person.birthYear < self.clock.NOW

--No se puede divorciar una persona de alguien con quien no esté actualmente casado.

--esta restricción está ya recogida en la precondición del método divorce() de la clase persona.

--deonticas

--Una persona no puede estar casada consigo misma.

context Person inv noCasadoConsigo: self.husband.husband -> forAll (p : Person | p <> self)

and self.wife.wife -> forAll (p : Person | p <> self)

--No puede haber más que un reloj en toda la aplicación (y por tanto compartido por todas las comunidades)

context Community inv soloUnReloj: self.clock -> size() = 1

--Monogamia: Una persona no puede tener más de un matrimonio activo en un momento dado.

context Person inv monogamia: (self.husband -> select(h | h.endYear.oclIsUndefined()) -> size() < 2) and (self.wife -> select(w | w.endYear.oclIsUndefined()) -> size() < 2)

--No se permite ni la eutanasia ni el suicidio

context Community inv noSuicidioNiEutansia: self.person.deathYear < self.clock.NOW or self.person.deathYear.oclIsUndefined()

--Los niños no pueden estar casados

context Persona inv ninyoNoCasado: self.age < self.community.childToYoung implies (self.wife -> size() = 0 and self.husband -> size() = 0)

--queries

--isMarried():Boolean que devuelve si una persona está actualmente casada o no

context Person :: isMarried(): Boolean

body: (self.husband -> notEmpty()) or (self.wife -> notEmpty())

--marriages():Integer que devuelve el número de matrimonios que ha tenido una persona a lo largo de su vida

--(incluido el actual, si es que estás casado)

context Person :: marriages(): Integer

body: (self.husband -> size()) + (self.wife -> size())

**CUESTIONES**

**¿Sería posible que se casasen personas de diferentes comunidades? En ese caso, ¿habría algún problema en la forma en la que habéis especificado las restricciones y las máquinas de estado de las personas, para que, a pesar de estar casadas, siempre se respeten las reglas de las dos comunidades?**

Sí podría hacerse a no ser que tengan una edad distinta de paso de niñez a juventud (los niños no pueden estar casados). Salvo ese matiz, sería posible ya que las restricciones del modelo son para todas las comunidades y solo existe un reloj para todas ellas, por los que los tiempos son iguales en todas.

**De acuerdo a tu modelo, ¿cuál sería el estado civil de una persona muerta? ¿Qué ocurre en la realidad? Es decir, de acuerdo a nuestro sistema legal en España, ¿Cuál es el estado civil de una persona fallecida? ¿Es lo mismo en otros países?**

Según mi modelo, el estado civil de una persona muerta es el que tuviese a la hora de morir, con la diferencia de que su pareja aparece como viuda. En España es igual pero en otros países puede ser diferente, según la legislación de cada uno.

**¿Has comprobado cuáles son los estados civiles que definen nuestras leyes? ¿Es lo mismo en todos los países? Si no, ¿cómo se debería modelar la aplicación que cada persona tuviera diferentes estados dependiendo de la comunidad a la cual perteneciera?**

Los estados civiles en España son: soltero, casado, separado judicialmente, divorciado y viudo. En otros países, por ejemplo, el divorcio no es legal, por lo que el estado civil de divorciado no existiría.

En mi modelo, todas las comunidades tienen los mismos estados civiles pero se podría modelar de la manera que la clase Community tuviese un atributo con los estados civiles disponibles en cada comunidad.

**En algunos países, casarse con menores no es delito civil, solo un problema ético. ¿Afectaría este hecho de alguna forma a tu modelo?**

Sí, porque en mi modelo existe una restricción que impide que los niños se puedan casar. Aunque la edad a la que pasan a ser jóvenes puede variar de comunidad a comunidad, por lo que una persona que en una comunidad sea joven, en otra puede ser todavía niño.