**B2**

Samuel Jurado Quintana, Alejandro Garau Madrigal, Pedro Díaz Gutiérrez,

Jesús Parejo Aliaga, Juan García Ruiz

# Marriage

**Restricciones Físicas:**

**Una persona no puede fallecer antes de nacer:**

context Person

inv dontDieBeforeBeingBorn:

deathDate <> -1 implies deathDate>=self.birthdate

**Un matrimonio no puede acabar antes de empezar:**

context Marriage

inv dontEndMarriageBeforeWedding:

finishDate <> -1 implies finishDate>=self.weddingDate

**Los muertos no pueden estar casados:**

context Marriage

inv noDeadMarriage:

self.isDead implies (self.husband->union(self.wife))->forAll(m| self.deathDate >= m.finishDate)

**No se puede nacer en el futuro:**

context Person

inv noFutureBorn:

self.birthdate<=self.system.clock.now

**No puede haber matrimonios futuros:**

contect Marriage

inv noFutureMarriage:

self.weddingDate<=Clock.allInstances->collect(now)->asSequence()->first()

**Restricciones deónticas:**

**Una persona no puede estar casada consigo misma:**

context Marriage

inv dontMarryYourself:

husband <> wife

**Una persona no puede tener más de un matrimonio activo en un momento dado:**

context Person

inv Monogamy:

self.husband->union(self.wife)->select(m | m.finishDate = -1)->size <= 1

**Los niños no pueden estar casados:**

En este caso hemos realizado una precondición en la operación marry() de System (que es quien casa a las dos personas) y una restricción, así nos aseguramos de que en el sistema no existen matrimonios entre niños y que ningún niño pueda ser casado mediante la operación marry() de System.

marry(p1: Person, p2: Person)

[...]

pre noKiddosAllowed: p1.age > 12 and p2.age > 12

context Person:

inv noChildMarried:

self.age < 12 implies not self.isMarried()

**No se permite ni la eutanasia ni el suicidio:**

No hemos sabido definir esta restricción.

**isMarried() query:**

isMarried(): Boolean= self.wife->union(self.husband)-> exists(m | m.finishDate=-1)

**marries() query:**

marriages(): Integer = self.wife->union(self.husband)->size()

**Operaciones del sistema:**

**Paso del tiempo:**

El paso del tiempo está representado con un reloj único en todo el modelo que controla el tiempo que pasa en ticks de 1 año. La clase *Clock*, tiene un atributo *now* que contiene el año actual. La operación *tick()* se define de la siguiente manera:

tick() -- 1 year

begin

self.now:=self.now+1;

for p in self.system.people do

if not p.isDead then p.birthday();

end

end

end

post oneYearHasPassed: self.now = self.now@pre+1

Que aumenta en 1 el año actual, y, si la persona no está muerta dentro del sistema, llama a la operación *birthday* de persona.

Sólo existe una única postcondición, *oneYearHasPassed* que controla que se haya sumado 1 a la variable now respecto al valor que tenía en la precondición.

**Casamiento de 2 personas:**

A la hora de que 2 personas se casen, el sistema es quien crea los enlaces entre las 2 personas, y éstas son notificadas mediante una llamada a una operación *marry()*.

class System

[...]

marry(p1:Person, p2:Person)

begin

declare m:Marriage;

m := new Marriage;

m.weddingDate:=self.clock.now;

m.finishDate:=-1;

insert(p1, m) into MarriageHusband;

insert(p2, m) into MarriageWife;

p1.marry(m, p2);

p2.marry(m, p1);

end

pre minimumAge: (p1.age > self.marryAge and p2.age > self.marryAge)

pre notMyself: p1<>p2

pre bothAlive: not(p1.isDead or p2.isDead)

pre noOneCurrentlyMarried:

not (p2.isMarried() or p1.isMarried())

post onlyOneNewMarriage: p1.marriages() = p1.husband@pre->union(p1.wife@pre)->size() + 1 and

p2.marriages() = p2.husband@pre->union(p2.wife@pre)->size() + 1

post newMarriage: let m:Marriage = Marriage.allInstances->select(m | m.oclIsNew())->asSequence()->first()

in m.weddingDate = Clock.allInstances->collect(now)->asSequence()->first()

and p2.husband->includes(m) and p1.wife->includes(m)

end

class Person

[...]

marry(m: Marriage, p: Person)

begin

end

pre marriageActive: m.weddingDate = system.clock.now and m.finishDate = -1

pre marriedWithTheOther: m.husband = self and m.wife = p or

m.husband = p and m.wife = self

Las precondiciones de Marry controlan que no se puedan casar 2 niños, ni consigo mismo, y que ambos estén vivos y no estén casados. Además al acabar la operación solo se crea un único matrimonio al acabar la operación.

Dentro de la operación *marry(p1: Person, p2: Person)* se hace una llamada a la operación *marry(m: Marriage, p: Person)* de la clase Person, que notifica a la persona de que se ha casado. Recibe el matrimonio recién creado para evitar que se llame desde fuera del sistema y la persona con la que se casa para comprobar que es la persona con la que te has casado y pasar del estado de *Soltero* al estado de *Casado* sin realmente haber creado un matrimonio.

Más abajo en el documento se explica porqué se ha utilizado esta solución respecto a las otras que pensamos.

**Divorciarse:**

Divorciarse funciona de manera similar a la operación *marry()*. También está controlado desde System y se le manda una notificación a las personas de que se han divorciado.

class System

[...]

operations

divorce(p1: Person, p2: Person)

begin

declare currentMarriage:Marriage;

currentMarriage := p1.husband->union(p1.wife)->select(m | m.finishDate = -1)

->asSequence()->first();

currentMarriage.finishDate := self.clock.now;

p1.divorce(currentMarriage);

p2.divorce(currentMarriage);

end

pre activeMarriageAndSamePersons:

p1.husband->union(p1.wife)->select(m | m.finishDate = -1)

->asSet()->intersection(p2.husband->union(p2.wife)->select(m | m.finishDate = -1)

->asSet())->size() = 1

post currentMarriagesFinished:

p1.husband->union(p1.wife)

->forAll(m | m.finishDate<=p1.system.clock.now@pre)

and

p2.husband->union(p2.wife)

->forAll(m | m.finishDate<=p2.system.clock.now@pre)

class Person

[...]

divorce(m: Marriage)

begin

end

pre divorcedNow: m.finishDate = system.clock.now

post everyMarriageEnd:

self.husband->union(self.wife)

->forAll(m | m.finishDate <= self.system.clock.now@pre)

El sistema divorcia a dos personas y les manda una señal mediante *divorce(m: Marriage)*, que, como en el caso anterior,

**Cumplir años:**

Una persona cumple años en el momento en el que se llama a la operación *tick()* de la clase Reloj. Nunca se podrá realizar una llamada de *birthday()* desde fuera puesto que tiene la siguiente precondición: system.clock.now > birthdate + age

birthday()

begin

self.age:=self.age+1

end

pre: system.clock.now > birthdate + age

post: self.age = self.age@pre + 1

**Fallecer:**

kill()

begin

self.deathDate := self.system.clock.now;

if self.isMarried() then

declare currentMarriage:Marriage;

currentMarriage := self.husband->union(self.wife)->select(m | m.finishDate = -1)

->asSequence()->first();

if currentMarriage.husband = self then

currentMarriage.wife.notifySpouseDead(self);

currentMarriage.finishDate := self.system.clock.now;

else

currentMarriage.husband.notifySpouseDead(self);

currentMarriage.finishDate := self.system.clock.now;

end

end

end

pre **isAlive**: not self.isDead

post **deathNoted**: self.deathDate=self.system.clock.now@pre

post **everyMarriageEnd**:

self.husband->union(self.wife)

->forAll(m | m.finishDate<=self.system.clock.now@pre)

Para la operación kill, terminamos todos los matrimonios que tuviera activos usando la operación *notifySpouseDead()* para notificar al cónyuge de que su pareja ha fallecido, para así cambiar de estado en la máquina de estados.

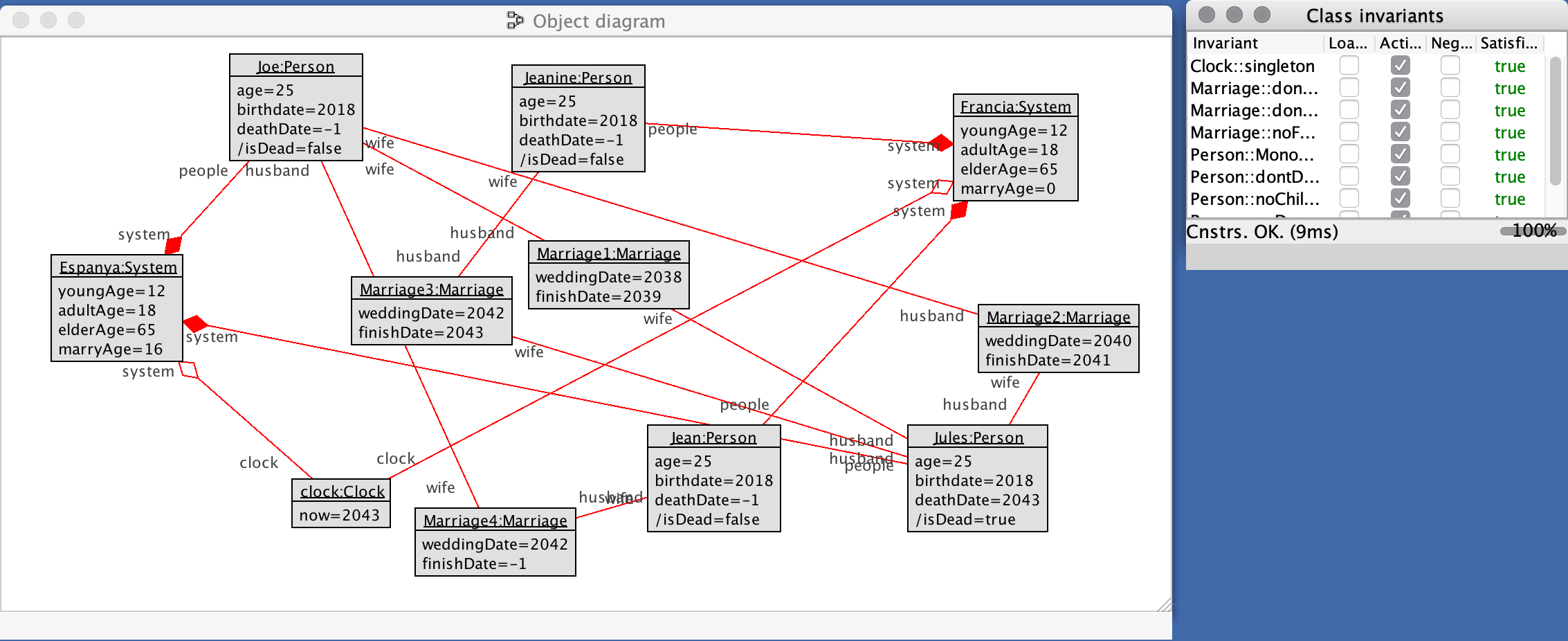
1. **¿Te obliga eso a cambiar muchas cosas en vuestro modelo inicial, o estaban todas esas variables recogidas en un solo sitio? ¿Tuvisteis en el modelo anterior alguna precaución con la posible variabilidad de las especificaciones?**

Lo único que tuvimos que cambiar fue añadir unas variables al sistema con las edades de joven, adulto y anciano, ya que en un principio no lo tuvimos en cuenta.

1. **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?**

Si sería posible que personas de distintas comunidades se casen, aunque ambas personas tendrían que adaptarse a las normas del sistema en el que se casan, el problema es que no puedes respetar las normas de ambas comunidades, puesto que cada sistema tiene tiene sus restricciones distintas y el casamiento se tiene que hacer únicamente en un sistema.

En este ejemplo podemos observar como varias personas de distintos sistemas se pueden casar entre ellos.



1. **De acuerdo a vuestro 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?**

El estado civil de una persona que ha fallecido sería el último estado civil que ha tenido mientras estaba vivo. Nuestro modelo se asemeja a la realidad, es decir al sistema legal en España. En el resto de países, la persona fallecida seguiría teniendo el estado civil anterior a la defunción, aunque existe una excepción en Francia, donde se una persona se puede casar con una persona ya fallecida, siempre y cuando se demuestre que la pareja tenía planes de boda.

Fuente: [Matrimonios Postmortem](https://www.legifrance.gouv.fr/affichCodeArticle.do?idArticle=LEGIARTI000024025844&cidTexte=LEGITEXT000006070721)

1. **¿Habéis comprobado cuales 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 del sistema al cual perteneciera?**

Sí, en España por ejemplo, tenemos los estados de soltero,casado, divorciado y viudo, en cambio en distintos países como México solo existen dos, soltero y casado, o en cambio Uruguay, soltero, casado, divorciado, viudo, hijo/a natural, hijo/a legítimo, padre/madre.

Podríamos modelarlo con una clase que fuese *CivilState* y que reflejase para un sistema, los distintos estados civiles que existen en él, además, añadir una relación con la clase *Person* que fuese el estado civil actual de esa persona, ya que la máquina de estados no sería viable, puesto que podría cambiar el número de estados de la misma.

1. **En Francia el incesto no es delito civil, solo un problema ético. ¿Afectaría este hecho de alguna forma a tu modelo?**

No, ya que en el sistema tenemos una variable que es la edad mínima a la que el matrimonio está permitido.

USE:

model Marriage

class Clock

attributes

now:Integer init: 2018

operations

tick() -- 1 year

begin

self.now:=self.now+1;

for p in self.system.people do

if not p.isDead then p.birthday();

end

end

end

post oneYearHasPassed: self.now = self.now@pre+1

end

class System

attributes

youngAge : Integer

adultAge : Integer

elderAge : Integer

marryAge : Integer

operations

divorce(p1: Person, p2: Person)

begin

declare currentMarriage:Marriage;

currentMarriage := p1.husband->union(p1.wife)->select(m | m.finishDate = -1)

->asSequence()->first();

currentMarriage.finishDate := self.clock.now;

p1.divorce(currentMarriage);

p2.divorce(currentMarriage);

end

pre activeMarriageAndSamePersons:

p1.husband->union(p1.wife)->select(m | m.finishDate = -1)

->asSet()->intersection(p2.husband->union(p2.wife)->select(m | m.finishDate = -1)

->asSet())->size() = 1

post currentMarriagesFinished:

p1.husband->union(p1.wife)

->forAll(m | m.finishDate<=p1.system.clock.now@pre)

and

p2.husband->union(p2.wife)

->forAll(m | m.finishDate<=p2.system.clock.now@pre)

marry(p1:Person, p2:Person)

begin

declare m:Marriage;

m := new Marriage;

m.weddingDate:=self.clock.now;

m.finishDate:=-1;

insert(p1, m) into MarriageHusband;

insert(p2, m) into MarriageWife;

p1.marry(m, p2);

p2.marry(m, p1);

end

pre minimumAge: (p1.age > self.marryAge and p2.age > self.marryAge)

pre notMyself: p1<>p2

pre bothAlive: not(p1.isDead or p2.isDead)

pre noOneCurrentlyMarried:

not (p2.isMarried() or p1.isMarried())

post onlyOneNewMarriage: p1.marriages() = p1.husband@pre->union(p1.wife@pre)->size() + 1 and

p2.marriages() = p2.husband@pre->union(p2.wife@pre)->size() + 1

post newMarriage: let m:Marriage = Marriage.allInstances->select(m | m.oclIsNew())->asSequence()->first()

in m.weddingDate = Clock.allInstances->collect(now)->asSequence()->first()

and p2.husband->includes(m) and p1.wife->includes(m)

end

class Person

attributes

age:Integer init: 0

birthdate:Integer init: Clock.allInstances->collect(now)->asSequence()->first()

deathDate:Integer init: -1

isDead:Boolean derive = deathDate<>-1

operations

marriages(): Integer = self.wife->union(self.husband)->size()

isMarried(): Boolean= self.wife->union(self.husband)->exists(m | m.finishDate=-1)

marry(m: Marriage, p: Person)

begin

end

pre marriageActive: m.weddingDate = system.clock.now and m.finishDate = -1

pre marriedWithTheOther: m.husband = self and m.wife = p or

m.husband = p and m.wife = self

birthday()

begin

self.age:=self.age+1

end

pre: system.clock.now > birthdate + age

post: self.age = self.age@pre + 1

notifySpouseDead(p:Person)

begin

end

pre bothMarriedWithTheOther:

let activeMarriage:Marriage = self.husband->union(self.wife)

->select(m | m.finishDate = -1)->asSequence()->first() in

self.isMarried() and

(activeMarriage.husband = p or activeMarriage.wife = p)

pre spouseIsDead: p.isDead

-- post: self.oclIsInState(Widowed)

kill()

begin

self.deathDate := self.system.clock.now;

if self.isMarried() then

declare currentMarriage:Marriage;

currentMarriage := self.husband->union(self.wife)->select(m | m.finishDate = -1)

->asSequence()->first();

if currentMarriage.husband = self then

currentMarriage.wife.notifySpouseDead(self);

currentMarriage.finishDate := self.system.clock.now;

else

currentMarriage.husband.notifySpouseDead(self);

currentMarriage.finishDate := self.system.clock.now;

end

end

end

pre isAlive: not self.isDead

post deathNoted: self.deathDate=self.system.clock.now@pre

post everyMarriageEnd:

self.husband->union(self.wife)

->forAll(m | m.finishDate<=self.system.clock.now@pre)

divorce(m: Marriage)

begin

end

pre divorcedNow: m.finishDate = system.clock.now

post everyMarriageEnd:

self.husband->union(self.wife)

->forAll(m | m.finishDate <= self.system.clock.now@pre)

statemachines

psm LifeCycle

states

s: initial

Kid

Young

Adult

Elder

Dead

transitions

s -> Kid

Kid -> Dead { kill() }

Young -> Dead { kill() }

Adult -> Dead { kill() }

Elder -> Dead { kill() }

Kid -> Kid { [self.age < self.system.youngAge - 1] birthday ()}

Kid -> Young { [self.age = self.system.youngAge - 1] birthday() }

Young -> Young { [self.age < self.system.adultAge - 1] birthday() }

Young -> Adult { [self.age = self.system.adultAge - 1] birthday() }

Adult -> Adult { [self.age < self.system.elderAge - 1] birthday() }

Adult -> Elder { [self.age = self.system.elderAge - 1] birthday() }

Elder -> Elder { birthday() }

end

psm CivilStatus

states

s: initial

Single

Divorced

Widowed

Married

transitions

s -> Single

Single -> Married { marry() }

Married -> Widowed { [self.isMarried()] notifySpouseDead() }

Married -> Divorced { divorce() }

Divorced -> Married { marry() }

Divorced -> Divorced { kill() }

Married -> Married { kill() }

Single -> Single { kill() }

Widowed -> Widowed { kill() }

Widowed -> Married { marry() }

end

end

class Marriage

attributes

weddingDate:Integer

finishDate:Integer

end

association MarriageWife between

Person[1]

role wife

Marriage[\*]

role husband

end

association MarriageHusband between

Person[1]

role husband

Marriage[\*]

role wife

end

composition Community between

System[1]

role system

Person[\*]

role people

end

aggregation ClockSystem between

System[\*] role system

Clock[1] role clock

end

constraints

context Person

inv dontDieBeforeBeingBorn:

deathDate <> -1 implies deathDate>=self.birthdate

inv noFutureBorn:

self.birthdate<=self.system.clock.now

inv noDeadMarriage:

self.isDead implies (self.husband->union(self.wife))->forAll(m| self.deathDate >= m.finishDate)

inv Monogamy:

self.husband->union(self.wife)->select(m | m.finishDate = -1)->size <= 1

inv noChildMarried:

self.age < 12 implies not self.isMarried()

context Marriage

inv dontMarryYourself:

husband <> wife

inv dontEndMarriageBeforeWedding:

finishDate <> -1 implies finishDate>=self.weddingDate

inv noFutureMarriage:

self.weddingDate<=Clock.allInstances->collect(now)->asSequence()->first()

context Clock

inv singleton:

Clock.allInstances->size()=1

SOIL:

**Ejecución 1:**

reset

!new System('S')

!new Clock('clock')

!insert (S, clock) into ClockSystem

delay 500

!new Person('Joe')

!insert (S, Joe) into Community

!new Person('Jules')

!insert (S, Jules) into Community

!new Person('Teresa')

!insert (S, Teresa) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Antonia')

!insert (S, Antonia) into Community

!clock.tick()

!clock.tick()

!new Person('Antonio')

!insert (S, Antonio) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Juana')

!insert (S, Juana) into Community

!new Person('Paco')

!insert (S, Paco) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Alfredo')

!insert (S, Alfredo) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Frasquito')

!insert (S, Frasquito) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Pepa')

!insert (S, Pepa) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Pepa.kill() -- F no fue vacunado por sus padres

!S.marry(Teresa, Antonio)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!S.marry(Frasquito, Juana)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Teresa.kill()

!S.marry(Antonio, Antonia)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!S.divorce(Frasquito, Juana)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!S.marry(Frasquito, Juana)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Antonio.kill()

!clock.tick()

!clock.tick()

!Juana.kill()

!clock.tick()

!S.marry(Joe, Jules)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Frasquito.kill()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Joe.kill()

delay 0

**Ejecución 2:**

reset

!new System('S')

!new Clock('clock')

!insert (S, clock) into ClockSystem

-- delay 500

!new Person('Pepa')

!insert (S, Pepa) into Community

!clock.tick()

!new Person('Alfredo')

!insert (S, Alfredo) into Community

!new Person('Joe')

!insert (S, Joe) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!Alfredo.kill() -- sus padres no le vacunaron........... F

!new Person('Jules')

!insert (S, Jules) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Frasquito')

!insert (S, Frasquito) into Community

!clock.tick()

!clock.tick()

!new Person('Juana')

!insert (S, Juana) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Teresa')

!insert (S, Teresa) into Community

!clock.tick()

!clock.tick()

!S.marry(Pepa, Joe)

!clock.tick()

!clock.tick()

!clock.tick()

!new Person('Paco')

!insert (S, Paco) into Community

!clock.tick()

!clock.tick()

!Joe.kill() -- F

!clock.tick()

!clock.tick()

!S.marry(Jules, Frasquito)

!clock.tick()

!clock.tick()

!new Person('Antonio')

!insert (S, Antonio) into Community

!new Person('Antonia')

!insert (S, Antonia) into Community

!clock.tick()

!clock.tick()

!clock.tick()

!S.divorce(Jules, Frasquito)

!clock.tick()

!Jules.kill() -- F

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!Frasquito.kill() -- F

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!S.marry(Paco, Antonia)

!S.marry(Juana, Antonio)

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!clock.tick()

!S.divorce(Paco, Antonia)

!Juana.kill() -- F

delay 0

DIAGRAMAS:

Aparecen en el siguiente orden: Diagrama de secuencia y de comunicación de la primera ejecución y diagrama de secuencia y comunicación de la segunda ejecución.

(Hay que hacer mucho zoom en los diagramas de secuencia).