Proyecto My Pouex- DMSS

Leonardo Rossi

May 6, 2014

Contents

Abstract

Objetivo y dominio de aplicación del DSL

El objetivo de este DSL es modelar el mundo virtual de una mascota, es decir un típico juego "Tamagotchi-like". La mascota se puede considerar como un pequeño cachorro para cuidar y cuidar. Requiere atención por parte del usuario, y se puede interactuar con él a través de la interfaz apropiada. Como resultado de las acciones del usuario, la mascota responderá modificando sus características y la activación de los estados, que también podría dar lugar a cambios en la visión. Por ejemplo, si el usuario hace jugar mucho la mascota, esta podría convertirse en sucio.

A través de el DSL, se puede definir:

- Como es su estructura gráfica.
- Sus características físicas y emocionales que distinguir desde otras.
- Acciones con las que se puede interactuar con la mascota y eventos temporales que se activarán periódicamente.
- Los estados en los que la mascota se puede encontrar mediante la definición de las reglas, también compleja, para su activación.

Chapter 1

Metamodelo del dominio

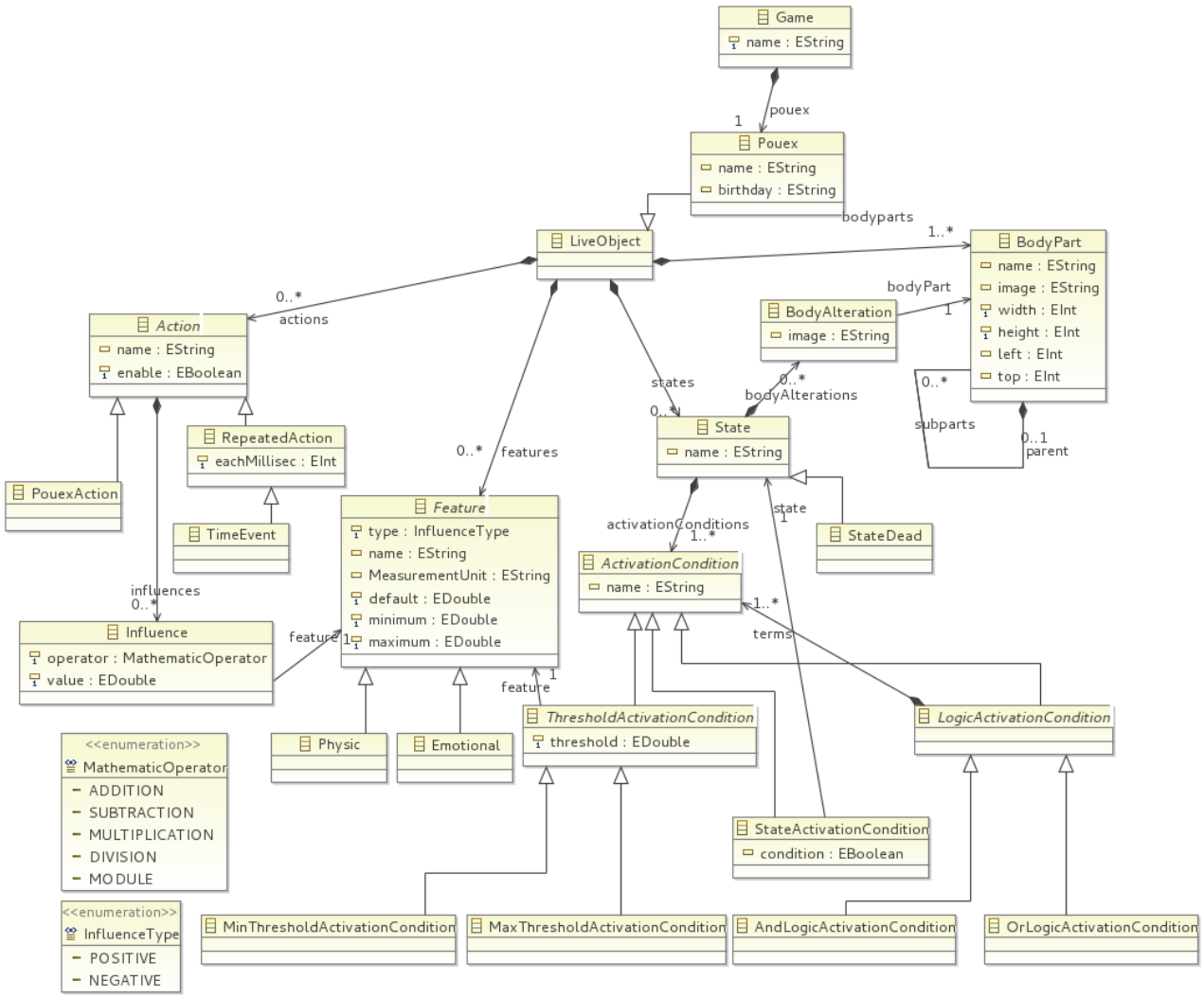

Figure 1.1: Meta-modelo del Dominio

Podemos dividir lógicamente el meta-modelo en dos partes:

Game y Pouex . Estos meta-clases representan la estructura lógica y conformación en la que el juego está dividido. En este caso, el juego está formado

sólo por una mascota Pouex. Sin embargo, en el futuro podría ser fácilmente extendido y refinado por la adición de nuevos personajes y animaciones.

LiveObject y todas las demás meta-clases . Un LiveObject representa la abstracción de un objeto en el juego y la forma en que interactúa con el medio ambiente y la forma en que el usuario puede interactuar con él.

A continuación vamos a discutir todos los conceptos en el meta-modelo de la figura [1.1].

1.1 Game

Este meta-clase representa la abstracción de un juego. Sirve como *Root* del meta-modelo. Contendrá la mascota y todo lo que se refiere a esa. Fue elegido de añadir esta abstracción para permitir que el programador sea capaz de ampliar fácilmente el juego añadiendo nuevos personajes o objetos con los que interactuar.

1.1.1 Atributos

name Representa el nombre de la aplicación. Por ejemplo, se puede dar el nombre Pouex o Tamagotchi.

1.1.2 Referencias a otros conceptos

pouex Un juego incluye una mascota. Esto le permite crear una nueva mascota en el juego.

1.1.3 Pouex

Este meta-clase representa la abstracción de una mascota.

Atributos

name El nombre de la mascota. Este parámetro ha sido pensado como una variable en el juego: se establece un valor por defecto. Pero el usuario puede editar y llamar su mascota como usted desea.

birthday Año de natalidad de la mascota. Esto también se piensa que es un parámetro predeterminado para definir el año de nacimiento de la mascota. En el juego real, cuando se crea la mascota, también se asocia con una fecha de nacimiento.

Referencias a otros conceptos

Este meta-clase se considera como una especialización de un LiveObject. De este modo, hereda todas sus características y capacidades.

Restricciones OCL

defineName() Comprueba que ha asignado un nombre por defecto a la mascota.

1.2 LiveObject

Este meta-clase representa la abstracción de un personaje genérico en el juego o un objeto animado. Representa el corazón del juego. Puede asociar: partes del cuerpo, acciones, características y estados.

1.2.1 Referencias a otros conceptos

bodyparts De esta manera podemos definir cómo se compone físicamente el objeto. Debe definirse al menos uno BodyPart.

features Podemos definir toda su características físico y emocional.

actions Podemos definir todas las acciones que puedan afectar el objeto.

states Definimos todos los estados en los que el objeto puede asumir.

1.2.2 Restricciones OCL

atLeastOneActionForEachFeature() El conjunto de acciones deben modificar el conjunto de características definidas, es decir, todas las características se verán afectadas por alguna acción. Es decir, cada función debe cumplir la siguiente restricción: seleccionar todas las acciones activas, y entre estos, debe haber por lo menos una acción que va a cambiar la función.

ActionNameMustBeUnique() El nombre de las acciones debe ser único. No puede existir dos acciones diferentes con el mismo nombre. Es decir, se seleccionan todas las acciones y, para cada una comprueba, que el nombre es único.

FeatureNameMustBeUnique() El nombre de las funciones debe ser único. No puede existir dos características diferentes del mismo nombre. Es decir, se seleccionan todas las características y para cada una, comprueba que el nombre es único.

StateNameMustBeUnique() El nombre de los estados debe ser único. No puede existir dos estados diferentes con el mismo nombre. Es decir, se seleccionan todas los estados y, para cada uno comprueba, que el nombre es único.

almostExistOneStateDead()) Es necesario que haya por lo menos un estado que va a modelar el final del juego.

1.2.3 BodyPart

Este meta-clase es la abstracción que define la forma en que se compone el objeto, y se muestra como una parte del cuerpo de eso. Cada parte se ve como un objeto en dos dimensiones de tamaño [width]x[height] y dibujado desde el píxel (izquierda, arriba), que se define con el punto de origen (0,0) en la parte superior izquierda de la pantalla.

Atributos

name Definisce il nome della parte del corpo del LiveObject.

image Define la dirección URL de una imagen para asociarla con esta parte del cuerpo.

width Define el ancho del objeto.

height Define la altura del objeto.

left Define la posición del eje X desde el que se posiciona el objeto.

top Define la posición del eje Y desde el que se posiciona el objeto.

Referencias a otros conceptos

subparts - parent La referencia *subparts* permite describir una jerarquía de composición, donde una parte del cuerpo puede contener la otra. El informe le permite definir la cantidad de objetos BodyPart a voluntad que quieras, con la profundidad deseada. Por ejemplo, podemos definir una cara que contiene dos objetos que definen los ojos, uno que define la boca y uno que define la nariz. La referencia opuesta *parent* se ha insertado para que sea capaz de cruzar en sentido contrario la relación de contención y volver fácilmente a los objetos contenedores. Acceleo se ha explotado esta referencia para poder marcar el nombre completo de la BodyPart. En el ejemplo, la boca tendrá como nombre completo " cara.boca". La reconstrucción automática del nombre entonces permite acceder a los elementos.

Restricciones OCL

uniqueNameInSameLevel() Compruebe que no hay dos BodyPart en el mismo nivel que se llaman de la misma manera.

1.2.4 Action

Este meta-clase es abstracta y se utiliza para definir el concepto general de la *acción*.

Atributos

name Nombre asociado a la acción. por ejemplo "Play" o "Clean".

enable Permite definir si la acción está habilitado o no. Puede ser útil si, por ejemplo, si desea desactivar temporalmente ciertas acciones en la versión actual del juego.

Referencias a otros conceptos

influences Define el conjunto de consecuencias que trae la acción, una vez activada. Por ejemplo, la acción "Play" puede tener una influencia positiva en la característica "Happiness". Una acción puede definir distintas influencias.

Restricciones OCL

cantModifyFeatureTwiceOrMore() Una acción no puede modificar dos veces el nivel de una determinada característica.

PouexAction

Meta-clase que extiende Action. Definir las acciones que la mascota Pouex puede realizar.

1.2.5 RepeatedAction

Meta-clase que extiende Action. Define las acciones que se realizan periódicamente y de forma automática.

Atributos

eachMillisec Define cada cuantos milisegundos debe llevarse a cabo esta acción.

1.2.6 TimeEvent

Meta-clase que surge como una especialización de la clase RepeatedAction. Representa un evento temporal.

1.2.7 Influence

Como se menciona en la definición de Action, este meta-clase representa una influencia que puede tener una acción sobre una característica. Se puede definir la forma en que la acción va a cambiar el valor actual de la Feature.

Las operaciones que se pueden definir serán del siguiente tipo:

nuevo valor característica = viejo valor [OPERADOR] [VALOR]

Atributos

operator Operador matemático para aplicarse a la operación de actualizar el valor actual de la característica. Podemos definir uno de los operadores definidos por la Enum MathematicOperator.

value valor numérico para ser utilizado por la actualización.

Referencias a otros conceptos

feature Referencia a la Feature que se beneficiará de la actualización de su valor actual.

Restricciones OCL

notDivByZero() Comprueba que no se trata de hacer una operación de división por cero.

1.2.8 Feature

Este meta-clase representa la abstracción de una característica de un LiveObject. Se define como una clase abstracta y ampliado por Physic (para las características físicas) y por Emotional (para las características emocional).

Atributos

type Define el tipo de característica, si sea positivo o negativo.

name Define el nombre de la característica.

MeasurementUnit Define una unidad de medida para la Característica. Por ejemplo, para definir el peso, su unidad de medida será los kg. No es obligatorio.

default El valor predeterminado para la característica.

minimum Valor mínimo asociado.

maximum Valor máximo asociado.

Restricciones OCL

coherentValueMinDefaultMax() Esta regla tiene la tarea de poner a prueba la consistencia de los valores seleccionados por defecto, mínimo y máximo. Por ejemplo, el valor por defecto no puede ser menor que el valor establecido mínimo.

1.2.9 Physic

Este meta-clase representa la abstracción de una característica física.

1.2.10 Emotional

Este meta-clase representa la abstracción de una característica emocional.

1.2.11 State

Este meta-clase representa un estado en el que el LiveObject puede encontrarse.

Atributos

name Especifica el nombre del estado.

Referencias a otros conceptos

activationConditions Representa el conjunto de condiciones que se deben cumplir para que sea activo.

bodyAlterations Representa la lista de los cambios físicos que sufre el objeto como resultado de la activación del estado.

Restricciones OCL

checkCicleActivationLevelOne() Ejecute una comprobación para evitar que se crea el bucle de activación dentro del estado. Ya que las posibles causas de activación, pueden ser por ejemplo la activación previa de un estado, deben ser evitados la ocurrencia de un bucle de auto-activación de si mismo.

checkCicleActivationLevelTwo El significado es el mismo, pero se evaluó la ActivationCondition del nivel 2.

checkCicleActivationFromLevelThree El significado es el mismo, pero se evaluó la ActivationCondition del nivel 3 y adelante los otros niveles.

1.2.12 StateDead

Especialización de la clase State. Esta clase se utiliza para modelar el caso en que la mascota va a morir y el juego termina.

1.2.13 BodyAlteration

Esta meta-clase representa el modelado de una alteración de una parte del cuerpo del LiveObject.

Atributos

image Es el enlace a la nueva imagen que se mostrará en el BodyPart.

Referencias a otros conceptos

bodyPart Identificar a cual parte del cuerpo se va a aplicar el cambio.

1.2.14 ActivationCondition

Esta meta-clase abstracta representa el modelado de una condición necesaria para la activación de un estado. El meta-modelo permite crear condiciones muy complejas, involucrando diferentes características y estados unidos en una expresión lógica con una capacidad de anidación. Las expresiones que se pueden generar son del tipo:

(característica1 >3.0) OR (característica2 <5.0 AND !estato1 AND estado2) OR (característica3 >6.0 OR estato3) OR ... (términoN)

Atributos

name Nombre que describe la condición de activación.

1.2.15 ThresholdActivationCondition

Esta meta-clase abstracta, subclase de ActivationCondition, representa el modelado de una condición de umbral que se refiere a una característica.

Atributos

threshold Umbral para definir la condición de activación.

Referencias a otros conceptos

feature Indica la caracteristica involucradas.

Restricciones OCL

thresholdShouldIsConsistentValueRespectOfFeatureMinMaxValue() Regla para comprobar que el umbral definido es consistente con los valores posibles que la característica puede asumir.

1.2.16 MinThresholdActivationCondition

Esta meta-clase, subclase de ThresholdActivationCondition, define un tipo particular de condiciones de umbral: se evalúa como verdadero si el valor umbral es menor que el valor actual de la característica.

1.2.17 MaxThresholdActivationCondition

Esta meta-clase, subclase de ThresholdActivationCondition, define un tipo particular de condiciones de umbral: se evalúa como verdadero si el valor umbral es mayor que el valor actual de la característica.

1.2.18 StateActivationCondition

Esta meta-clase, subclase de ActivationCondition, define un tipo particular de condiciones de activación. Es decir, se evalúa como verdadero si un estado en particular es activo o inactivo.

Atributos

condition Representa la condición de activación: si es verdadero, entonces la condición es verdadera si el estato *state* llegó a ser activo, de lo contrario la condición es verdadera si el estado se ha convertido en inactivo.

Referencias a otros conceptos

state Referencia el estado que se debe controlar. De ello depende la activación de la condición.

1.2.19 LogicActivationCondition

Esta meta-clase abstracta, subclase de ActivationCondition, representa el modelado de una condición lógica.

Referencias a otros conceptos

terms Representan la lista de términos de la expresión lógica.

1.2.20 AndLogicActivationCondition

Esta meta-clase, subclase de LogicActivationCondition, representa el modelado de una condición lógica AND.

Es posible modelar condiciones del tipo: termine1 AND término2 AND ... AND términoN

1.2.21 OrLogicActivationCondition

Esta meta-clase, subclase de LogicActivationCondition, representa el modelado de una condición lógica OR.

Es posible modelar condiciones del tipo: término1 OR término2 OR ... OR términoN

1.2.22 MathematicOperator

Enumeración para gestionar los tipos de operadores matemáticos utilizados en objetos Influence para influir de una manera apropiada el valor de una característica. Sus valores posibles son los siguientes:

ADDITION Representa una operación de suma: feature.value += influence.value;

SUBTRACTION Representa una operación de substracción: feature.value -= influence.value;

MULTIPLICATION Representa una operación de multiplicación: feature.value *= influence.value;

DIVISION Representa una operación de división: feature.value /= influence.value;

MODULE Representa una operación de módulo: feature.value %= influence.value;

1.2.23 InfluenceType

Enumeración para manejar los diferentes tipos de función, positivos y negativos. Por ejemplo: valores altos significa que la característica se acentúa.

Chapter 2

Sintaxis concreta

La sintaxis concreta a través de Xtext se compone de un Game (véase la figura [2.1]), donde se puede definir un Pouex.

```
Game returns Game:
    'Game'
    name=EString
    '{'
        pouex=Pouex
    '}';
```

Figure 2.1: Game definido con Xtext

Por su parte, un Pouex es definido principalmente por un nombre. Siguiendo, se puede definir opcionalmente una fecha de nacimiento y una serie de BodyPart, las características, acciones y estados (véase la figura [2.2]).

```
Pouex returns Pouex:
    'Pouex'
    name=EString
    '{'
        ('birthday' birthday=EString)?
        'bodyparts' '{' bodyparts+=BodyPart ( "," bodyparts+=BodyPart)* '}'
        ('features' '{' features+=Feature ( "," features+=Feature)* '}' )?
        ('actions' '{' actions+=Action ( "," actions+=Action)* '}' )?
        ('states' '{' states+=State ( "," states+=State)* '}' )?
    '}';
```

Figure 2.2: Pouex definido en Xtext

Cada uno de los tres elementos (véase la figura [2.3]) son súper-clase. Un estado puede ser un State o un StateDead. Una característica puede ser una característica Physic (física), o Emotional (emocional).

```
Action returns Action:
    TimeEvent | PouexAction | RepeatedAction_Impl;

State returns State:
    State_Impl | StateDead;

Feature returns Feature:
    Physic | Emotional;
```

Figure 2.3: Las acciones, los estados y las características definidas en Xtext

En la figura [2.4] se define un objeto BodyPart como un objeto con un nombre, otros atributos y una lista de objetos BodyPart incluidos dentro el objeto si mismo. Los atributos definen gráficamente: la imagen, las dimensiones (anchura y altura)

y su posición en relación con el objeto que se ha añadido. Se puede crear una estructura de árbol de objetos BodyPart para componer en detalle la estructura deseada.

```
BodyPart returns BodyPart:
    'BodyPart'
    name=EString
    '{'
        'image' image=EString
        'width' width=EInt
        'height' height=EInt
        ('left' left=EInt)?
        ('top' top=EInt)?
        ('subparts' '{' subparts+=BodyPart ( "," subparts+=BodyPart)* '}' )?
    '}';
```

Figure 2.4: BodyPart definido en Xtext

A continuación, se define una característica física y emocional. En ambos casos, constará de un nombre y algunos parámetros: valor predeterminado, mínimo, máximo, el tipo, y, opcionalmente, una unidad de medida (véase la figura [2.5]). Además, se define como un tipo de enumeración para el tipo, que puede tomar dos valores distintos.

```
Physic returns Physic:
    'Physic'
    name=EString
    '{'
        'default' default=EDouble
        'minimum' minimum=EDouble
        'maximum' maximum=EDouble
        'type' type=InfluenceType
        ('MeasurementUnit' MeasurementUnit=EString)?
    '}';

Emotional returns Emotional:
    'Emotional'
    name=EString
    '{'
        'default' default=EDouble
        'minimum' minimum=EDouble
        'maximum' maximum=EDouble
        'type' type=InfluenceType
        ('MeasurementUnit' MeasurementUnit=EString)?
    '}';

enum InfluenceType:
    POSITIVE = 'positive' | NEGATIVE = 'negative'
;
```

Figure 2.5: Feature Physic e Emotional definide en Xtext

En figura [2.6] se puede ver como se define las diferentes acciones y sus influencias sobre las características de Pouex. Todo puede o no puede tener la palabra clave “enable” para determinar si la acción está habilitado o no. A continuación se introduce la palabra clave que identifica el tipo y el nombre asociado.

Si se trata de una RepeatedAction o de un TimeEvent, puede definir cada cuántos milisegundos se repita la acción. En los tres tipos de acción se puede definir una serie de factores que influyen en las características. Una “influencia” se define por el nombre de la Feature, seguido por un operador matemático para aplicar a su valor actual y un valor de número entero.

El operador matemático, definido como una enumeración, permite cinco tipos de operaciones en el valor actual de característica:

1. Suma de un valor
2. Substracción de un valor
3. Multiplicación de un valor
4. división de un valor
5. Cálculo del módulo para un valor

```
Influence returns Influence:
    feature=[Feature|EString] operator=MathematicOperator value=EDouble
;

EBoolean returns ecore::EBoolean:
    'true' | 'false';

TimeEvent returns TimeEvent:
    enable?='enable'
    'TimeEvent'
    name=EString
    '{'
        'eachMillisec' eachMillisec=EInt
        ('influences' '{' influences+=Influence ( "," influences+=Influence)* '}' )?
    '}';

PouexAction returns PouexAction:
    (enable?='enable')?
    'PouexAction'
    name=EString
    '{'
        ('influences' '{' influences+=Influence ( "," influences+=Influence)* '}' )?
    '}';

RepeatedAction_Impl returns RepeatedAction:
    enable?='enable'
    'RepeatedAction'
    name=EString
    '{'
        'eachMillisec' eachMillisec=EInt
        ('influences' '{' influences+=Influence ( "," influences+=Influence)* '}' )?
    '}';
enum MathematicOperator:
    ADDITION = '+=' |
    SUBTRACTION = '-=' |
    MULTIPLICATION = '*=' |
    DIVISION = '/=' |
    MODULE = '%=';
```

Figure 2.6: Diferentes tipos de acciónes definida en Xtext y la influencia sobre una característica.

Por último, vemos cómo se define un estado. Puede ser de dos tipos: normales y StateDead que también define el final del juego. En figura [2.7] se puede ver cómo su definición difiere sólo de palabras clave que les contraddistigue.

Ellos serán identificados por un nombre. Además, puede definir un conjunto de condiciones de activación y, opcionalmente, un conjunto de objetos gráficos que representan alteraciones de Pouex.

```
State_Impl returns State:
    'State'
    name=EString
    '{'
        'activationConditions' '{' activationConditions+=ActivationCondition ( activationConditions+=ActivationCondition)* '}'
        ('bodyAlterations' '{' bodyAlterations+=BodyAlteration ( "," bodyAlterations+=BodyAlteration)* '}' )?
    '}';
StateDead returns StateDead:
    'StateDead'
    name=EString
    '{'
        'activationConditions' '{' activationConditions+=ActivationCondition ( activationConditions+=ActivationCondition)* '}'
        ('bodyAlterations' '{' bodyAlterations+=BodyAlteration ( "," bodyAlterations+=BodyAlteration)* '}' )?
    '}';
```

Figure 2.7: State y StateDead definido en Xtext.

En figura [2.8] se define como la clase abstracta ActivationCondition se compone:

- MinThresholdActivationCondition
- MaxThresholdActivationCondition
- StateActivationCondition
- AndLogicActivationCondition
- OrLogicActivationCondition

La condición de activación MinThresholdActivationCondition y MaxThresholdActivationCondition se definen de la siguiente manera: una descripción opcional, el nombre de la característica, el símbolo que identifica si menor o mayor y el valor de umbral.

La condición de activación StateActivationCondition se define de una manera similar: una descripción opcional, el símbolo de denegación opcionalmente y el nombre del estado que deben tomarse en consideración.

Las condiciónes de Or y And lógico permiten definir un conjunto de condiciones relacionadas con el operador lógico seleccionado. Por ejemplo, si elige el operador lógico AND, entonces todas las condiciones siguientes deben ser simultáneamente verdaderas, para ser verdad, la condición de disparo.

La composición de las condiciones de esta manera nos permite modelar incluso las condiciones complejas que requieren la participación de los diferentes estados y características.

```
ActivationCondition returns ActivationCondition:
    MinThresholdActivationCondition | MaxThresholdActivationCondition |
    StateActivationCondition | AndLogicActivationCondition | OrLogicActivationCondition;
MinThresholdActivationCondition returns MinThresholdActivationCondition:
    '[min]'
        (name=EString)? feature=[Feature|EString] '<' threshold=EDouble
    '[/min]';

MaxThresholdActivationCondition returns MaxThresholdActivationCondition:
    '[max]'
        (name=EString)? feature=[Feature|EString] '>' threshold=EDouble
    '[/max]';

StateActivationCondition returns StateActivationCondition:
    '[state]'
        (name=EString)? (condition?='!')? state=[State|EString]
    '[/state]';

AndLogicActivationCondition returns AndLogicActivationCondition:
    '[and]'
        (name=EString)? terms+=ActivationCondition ( terms+=ActivationCondition)*
    '[/and]';

OrLogicActivationCondition returns OrLogicActivationCondition:
    '[or]'
        (name=EString)? terms+=ActivationCondition ( terms+=ActivationCondition)*
    '[/or]';
```

Figure 2.8: ActivationCondition definida en Xtext.

Chapter 3

Ejemplo de modelo

El siguiente ejemplo muestra un ejemplo de un modelo de juego. Va a generar la mascota Tux, junto con toda la acción, características y estados que son parte del juego.

Se presenta la versión .xmi y la versión escrita con el DSL generado con Xtext.

Tomamos nota de que el juego con la mascota Tux se describirá por cuatro partes básicas. En fin: el cuerpo, características, acciones y estados. El cuerpo se describe como se hace la mascota. El ejemplo se describe a través de la parte del cuerpo del cuerpo "body" que contendrá todas las partes. Se define las cejas, los ojos, la boca, los brazos y los pies.

Posteriormente, se van a describir las "features". Este ejemplo define cinco características: Cuatro física y emocional. En orden: el peso, la felicidad, la suciedad, la edad y el hambre.

Por ejemplo, la edad comienza a partir de un valor cero, que es también el mínimo hasta ser capaz de alcanzar el valor máximo de 100 años. es una característica "positiva" y tiene como unidad de medida el "año".

A continuación se enumeran las acciones que hacen el juego activo. En el ejemplo tenemos cinco acciones diferentes: dos acciones que pueden desempeñar nuestra mascota y dos eventos temporales repetidas en el tiempo.

Tenemos implementado: jugar, comer, limpiar, cumpleaños y un evento que marca el paso del tiempo cada segundo. Tomemos por ejemplo el cumpleaños: el juego se ve como un evento repetido cada 10 segundos, y cada vez que se produce, aumenta el número de años de la mascota.

Finalmente, definimos los estados que pueden ser activados por la mascota. Como se puede ver en el ejemplo, se pueden crear condiciones de disparo muy sofisticadas. En el ejemplo, la terminación del juego, el estado del tipo StateDead, se activa por la condición siguiente (escrito en meta-código):

IF (Weight >80.0 OR Happiness <1.0 OR Age >99.0 OR (Hungry AND Dirty)
OR Hungry >99.0) THEN game over!

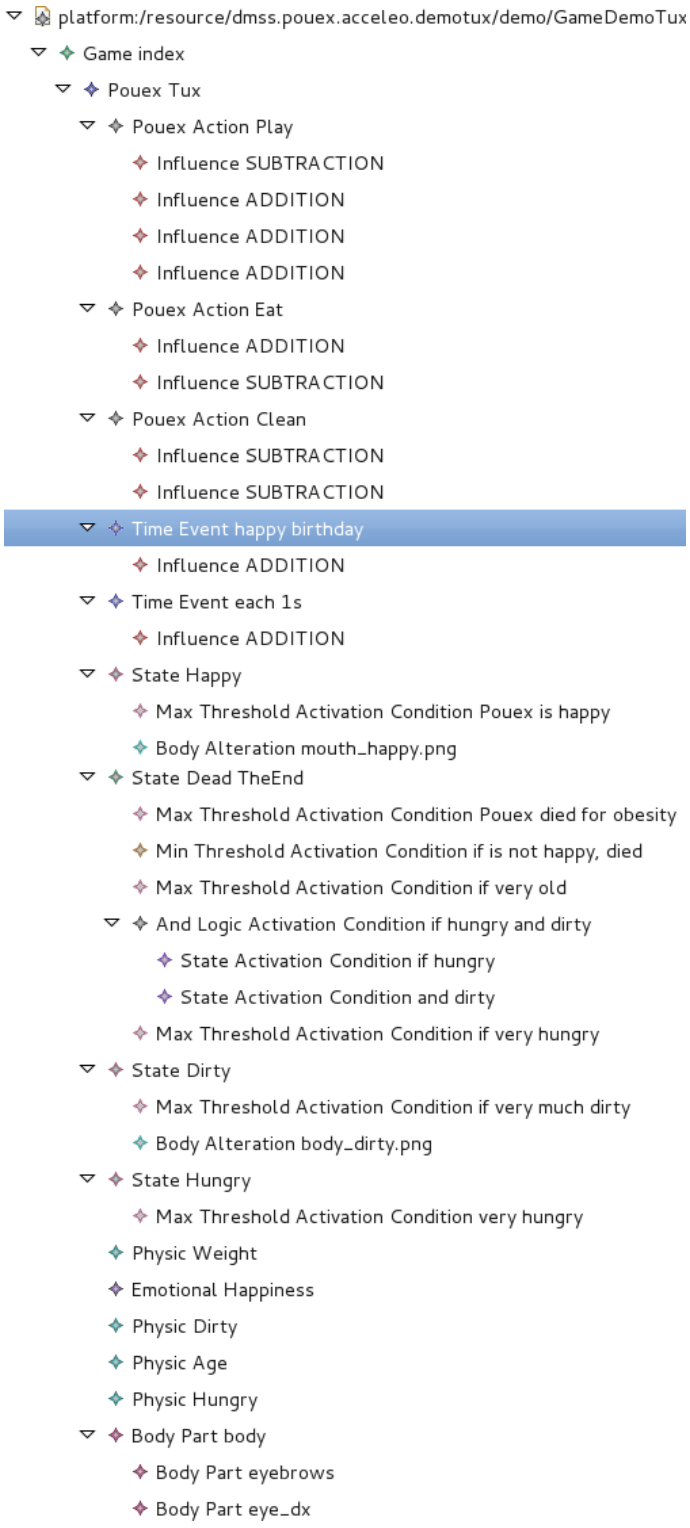

Figure 3.1: Demo di gioco scritto in .xmi

```
Game index {
    Pouex Tux {
        bodyparts {
            BodyPart body {
                image "body.png" width 190 height 179 left 0 top 0 subparts {
                    BodyPart eyebrows {
                        image "eyebrows.png" width 76 height 32 left 58 top 0
                    },
                    BodyPart eye_dx {
                        image "eye_dx.png" width 37 height 45 left -20 top 14
                    },
                    BodyPart eye_sx {
                        image "eye_sx.png" width 35 height 34 left -25 top 11
                    },
                    BodyPart mouth {
                        image "mouth.png" width 30 height 24 left -80 top 28
                    },
                    BodyPart arm_dx {
                        image "arm_dx.png" width 37 height 61 left 10 top 0
                    },
                    BodyPart arm_sx {
                        image "arm_sx.png" width 37 height 61 left 100 top 0
                    },
                    BodyPart foot_dx {
                        image "foot_dx.png" width 61 height 31 left -50 top 48
                    },
                    BodyPart foot_sx {
                        image "foot_sx.png" width 61 height 31 left 90 top 13
                    }
                }
            }
        }
        features {
            Physic Weight {
                default 20.0 minimum 5.0 maximum 100.0 type positive MeasurementUnit kg
            },
            Emotional Happiness {
                default 3.0 minimum 0.0 maximum 10.0 type positive MeasurementUnit hp
            },
            Physic Dirty {
                default 30.0 minimum 0.0 maximum 100.0 type positive MeasurementUnit
                dirtiness
            },
            Physic Age {
                default 0.0 minimum 0.0 maximum 100.0 type positive MeasurementUnit years
            },
            Physic Hungry {
                default 10.0 minimum 0.0 maximum 100.0 type positive
            }
        }
        actions {
            enable PouexAction Play {
                influences {
                    Weight -= 1.0,
                    Happiness += 1.0,
                    Dirty += 5.0,
                    Hungry += 5.0
                }
            },
            enable PouexAction Eat {
                influences {
                    Weight += 10.0,
                    Hungry -= 5.0
                }
            },
            enable PouexAction Clean {
                influences {
                    Happiness -= 1.0,
                    Dirty -= 15.0
                }
            },
            enable TimeEvent "happy birthday" {
                eachMillisec 10000 influences {
                    Age += 1.0
                }
            },
            enable TimeEvent "each 1s" {
                eachMillisec 1000 influences {
                    Hungry += 0.01
                }
            }
        }
        states {
            State Happy {
                activationConditions {
                    [max] "Pouex is happy" Happiness > 7.0 [/max]
                }
                bodyAlterations {
                    BodyAlteration {
                        image "mouth_happy.png" bodyPart mouth
                    }
                }
            },
            StateDead TheEnd {
                activationConditions {
                    [max] "Pouex died for obesity" Weight > 80.0 [/max]
                    [min] "if is not happy, died" Happiness < 1.0 [/min]
                    [max] "if very old" Age > 99.0 [/max]
                    [and] "if hungry and dirty"
                      [state] "if hungry" Hungry [/state]
                      [state] "and dirty" Dirty [/state]
                    [/and]
                    [max] "if very hungry" Hungry > 99.0 [/max]
                }
            },
            State Dirty {
                activationConditions {
                    [max] "if very much dirty" Dirty > 80.0 [/max]
                }
                bodyAlterations {
                    BodyAlteration {
                        image "body_dirty.png" bodyPart body
                    }
                }
            },
            State Hungry {
                activationConditions {
                    [max] "very hungry" Hungry > 80.0 [/max]
                }
            }
        }
    }
}
```

Figure 3.2: Juego escrito con el DSL generado por Xtext

Chapter 4

Generación de Código

A través del proyecto escrito con Acceleo, el ejemplo anterior de la mascota Tux se implementa con la generación automática de código a partir del modelo. En figura [4.1] se puede ver el diseño final del proyecto generado.

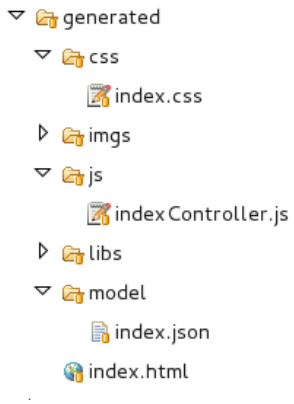

Figure 4.1: Código generado automáticamente con Acceleo

En el ejemplo, se inserta en la carpeta "generated" todo el código generado automáticamente. La aplicación resultante hace uso de las siguientes tecnologías: HTML5 + CSS + JavaScript. En particular, usando los siguientes framework:

Bootstrap v3.1.1 - más popular CSS framework para desarrollar aplicaciones web adaptable.

jQuery v1.11.0 - popular framework JavaScript, requerido da Bootstrap.

AngularJS v1.2.16 - framework JavaScript para la creación de aplicaciones para la web y, sobre todo, dispositivos móviles.

ngStorage - extensión de AngularJS para utilizar el LocalStorage presente en la versión 5 di Html y utilizado para guardar la partida.

Estos framework son explotados para facilitar el desarrollo de aplicaciones, para darle un aspecto uniforme y desarrollar de enlace de datos entre el modelo y el control. El generador de código crea cuatro archivos distintos:

css/index.css Archivo CSS que contiene el estilo de la aplicación.

js/indexController.js Contiene la lógica del juego.

model/index.json Contiene el modelo en formato JSON de la mascota. Este se carga desde el controlador, una vez inicializado el juego.

index.html Contiene el código HTML de la aplicación.

Il termine "index" aparece en los nombres de los archivos porque el juego fue llamado de esta manera. Esto permite generar el código de juegos diferentes vinculadas a diferentes modelos, en la misma carpeta.

Además, el usuario debe insertar las imágenes en la carpeta "imgs", esta también resulta ser la raíz predeterminada. Por ejemplo, en el código de la figura [3.2] se puede ver que el BodyPart "mouth" define como imagen el "mouth.png". En consecuencia, se inserta en la carpeta *imgs* la imagen con ese nombre.

Una de las limitaciones de Acceleo parece ser la copia de archivos. No permite esta operación. Como resultado, la carpeta "libs" tiene que ser copiada manualmente.

```
// state's activation
$scope.$watchCollection("[ $storage.game.pouex.features['Dirty'].value ]", function(newValues, oldValues){
  if(typeof $scope.$storage.game != "undefined"){
    var i = 0;
    if(newValues[ i++ ] > 80.0){
      // set in model the activation
      $scope.$storage.game.pouex.states.Dirty.active = true;
    }else{
     // set in model the activation
     $scope.$storage.game.pouex.states.Dirty.active = false;
    }
  }
});

$scope.$watch("$storage.game.pouex.states['Dirty'].active", function(newValue, oldValue){
  if(typeof $scope.$storage.game != "undefined"){
    if(newValue){
          $scope.$storage.game.pouex.body.body.default = $scope.$storage.game.pouex.body.body.imageUrl;
        // state animation
        $scope.$storage.game.pouex.body.body.imageUrl = 'imgs/body_dirty.png';
    }else{
        // restore animation
        $scope.$storage.game.pouex.body.body.imageUrl = $scope.$storage.game.pouex.body.body.default;
    }
  }
});
```

Figure 4.2: El código generado para administrar el estado Dirty

A continuación, en la figura [4.2], vemos un ejemplo del código generado para administrar el estado "Dirty". La función $watchCollection le permite invocar la función anónima, que se define a continuación, cada vez que la característica "game.pouex.features['Dirty'].value" cambia en el valor. Las variables que definen las características y el estado, podemos verlos en el modelo en formado JSON en la carpeta "model".

El siguiente "if" en el código implementa las condiciones de activación del estado. En este caso, si el valor de la característica "Dirty" es mayor de 80.0. Como resultado, se actualiza el modelo de datos, cambiando el valor de la variable booleana "game.pouex.states.Dirty.active".

El $watch que sigue se utiliza para realizar las alteraciones físicas de la mascota, si los hay. En el ejemplo, si el nuevo estado "game.pouex.states['Dirty'].active" se convirtió en activo, una copia de seguridad se realiza sobre el valor previamente establecido como imagen del BodyPart "game.pouex.body.body.imageUrl" y, luego, cambia la imagen con el nuevo enlace. Si el estado es desactivado, el valor se restablece con la copia de seguridad, es decir "default".

Chapter 5

Herramientas de desarrollo

Para el desarrollo del meta-modelo y la sintaxis concreta se han utilizado:

- Eclipse Kepler versión Eclipse Modeling Tools
- Xtext 2.4.3
- OCL 3.3.2
- Epsilon 1.1
- Empatic 0.8
- Acceleo 3.4.2

Para el desarrollo de código de la aplicación concreta a través Acceleo se han utilizado:

Bootstrap v3.1.1 - más popular CSS framework para desarrollar aplicaciones web adaptable.

jQuery v1.11.0 - popular framework JavaScript, requerido da Bootstrap.

AngularJS v1.2.16 - framework JavaScript para la creación de aplicaciones para la web y, sobre todo, dispositivos móviles.

ngStorage - extensión de AngularJS para utilizar el LocalStorage presente en la versión 5 di Html y utilizado para guardar la partida.