4.3 OCL (LENGUAJE DE ESPECIFICACIÓN DE OBJETOS)

Este documento introduce y define el Lenguaje de Especificación de Objetos (Object Constraint Language, OCL), es un lenguaje formal para expresar restricciones libres de efectos colaterales. Los usuarios del Lenguaje Unificado para Modelado (UML) y de otros lenguajes, pueden usar el OCL para especificar restricciones y otras expresiones incluidas en sus modelos. El OCL tiene características de un lenguaje de expresión, de un lenguaje de modelado y de un lenguaje formal.

Lenguaje de expresión

El OCL es un lenguaje de expresión puro. Por lo tanto, garantiza que una expresión OCL no tendrá efectos colaterales; no puede cambiar nada en el modelo. Esto significa que el estado del sistema no cambiará nunca como consecuencia de una expresión OCL, aun cuando una expresión OCL puede usarse para *especificar* un cambio de estado, por ejemplo, en una post-condición. Todos los valores, de todos los objetos, incluyendo todos los enlaces, no cambiarán, cuando una expresión OCL es evaluada, simplemente devuelve un valor.

Lenguaje de modelado

El OCL no es un lenguaje de programación, por lo tanto, no es posible escribir lógica de programa o flujo de control en OCL. No es posible invocar procesos o activar operaciones que no sean consultas en OCL. Dado que el OCL es un lenguaje de modelado en primer lugar, es posible que haya cosas en él que no sean directamente ejecutables.

Como el OCL es un lenguaje de modelado, toda consideración de implementación está fuera de su alcance, y no puede ser expresada en el lenguaje OCL. Conceptualmente, cada expresión OCL es atómica. El estado de los objetos en el sistema no puede variar durante la evaluación.

Lenguaje Formal

OCL es un lenguaje formal donde todos los constructores tienen un significado formalmente definido, la especificación del OCL es parte del UML. El OCL no pretende reemplazar lenguajes formales existentes como VDM y Z.

¿Porqué se requiere usar un lenguaje Formal?

En el modelado orientado a objetos, un modelo gráfico como el de clases no es suficiente para lograr una especificación precisa y no ambigua. Existe la necesidad de describir características adicionales sobre los objetos del modelo. Muchas veces estas características se describen en lenguaje natural. La práctica ha revelado que muy frecuentemente esto produce ambigüedades. Para escribir características no ambiguas se han desarrollado los lenguajes formales.

¿Porqué OCL y no otros lenguajes formales?

La desventaja de los lenguajes formales tradicionales es que son adecuados para personas con una fuerte formación matemática, pero difíciles para el modelador de sistemas.

El OCL ha sido desarrollado para cubrir esa brecha. Es un lenguaje formal, fácil de leer y escribir. Ha sido desarrollado como un lenguaje de modelado para negocios dentro de la división Seguros de IBM, y tiene sus raíces en el método *Syntropy*.

4.3.1 En donde usar OCL

El OCL puede ser usado con distintos propósitos:

- Para especificar características estáticas sobre clases y tipos en un modelo de clases.
- Para especificar características estáticas de tipo para Estereotipos.
- Para especificar pre y post-condiciones sobre Operaciones y Métodos.
- Como lenguaje de navegación.
- Para especificar restricciones sobre operaciones:

Dentro del documento *Semántica del UML*, el OCL es usado en la sección reglas bien formuladas, como constantes estáticas sobre las meta-clases en la sintaxis abstracta. En varios lugares también es usado para definir operaciones 'adicionales', que son tomadas en cuenta en la formación de reglas.

El diagrama servirá como ejemplo para las siguientes explicaciones

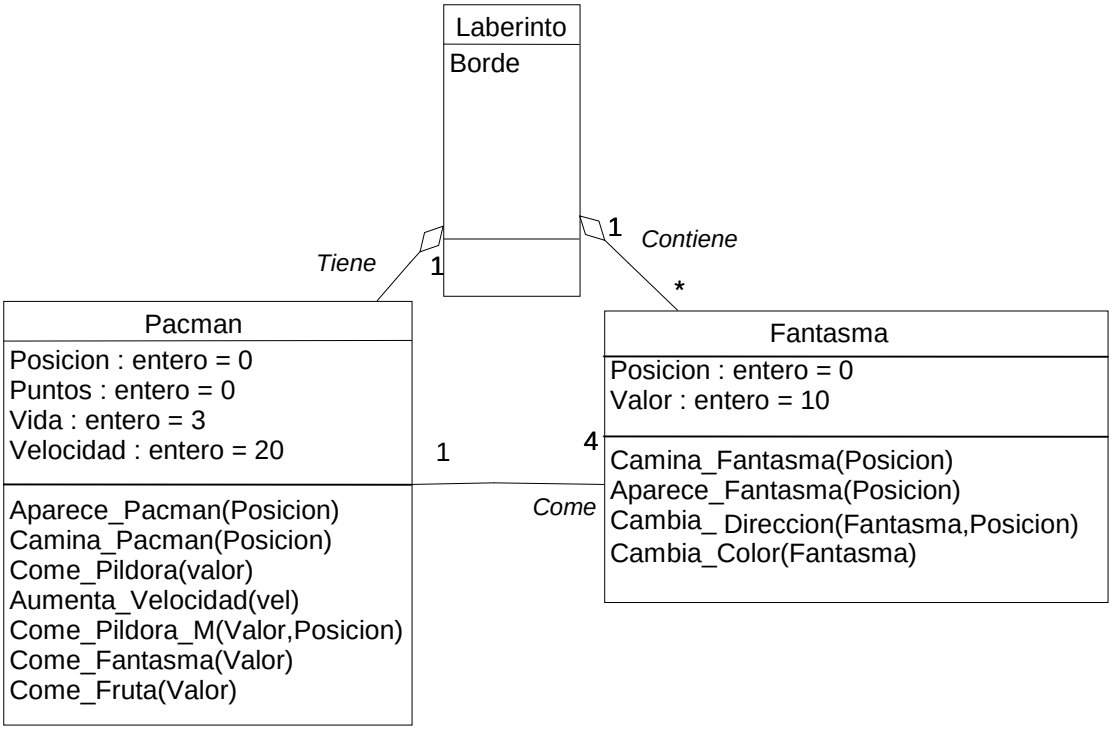

Figura # 2

4.3.2 Conexión con el metamodelo UML

self: Cada expresión OCL está escrita en el contexto de una instancia de un tipo en particular. En una expresión OCL el nombre ***self*** es usado para referirse a dicha instancia.

Características Estáticas: Una expresión OCL puede ser parte de una característica estática, que es una Restricción estereotipada con «invariant». Cuando una Característica estática está asociada a un Clasificador, la expresión es una invariante del tipo, y debe ser verdadera para todas las instancias de ese tipo, en todo momento. Si el contexto es Pacman, entonces ***self*** se refiere a una instancia de Pacman. En la expresión

self.Posicion

Pre y Post-condiciones:

Las expresiones OCL pueden ser parte de pre-condiciones o post-condiciones, que son Restricciones estereotipadas con «pre-condition» y «post-condition» respectivamente. Las pre-condiciones o post-condiciones se aplican tanto a Métodos como a Operaciones. En este caso la instancia que da contexto es del tipo al que pertenece la operación. La notación usada en este documento es subrayar el tipo y la declaración de la operación, y poner la frase 'pre:' y 'post:' antes de las pre-condiciones y post-condiciones.

nombreDeTipo::nombreDeOperacion(parametro1 : Tipo1, ...): TipoDevolucion
pre : parametero1 > …
post: resultado = ...

4.3.3 Objetos y propiedades

Las expresiones OCL pueden referirse a tipos, clases, interfaces, asociaciones (actuando como tipos) y tipos de datos. También todos los atributos, extremos de asociación, métodos y operaciones que no tengan efectos colaterales y estén definidos en esos tipos pueden ser usados. En un modelo de clases, una operación o método no tiene efectos colaterales si el atributo isQuery de la operación es verdadero. Dentro de este documento, se va a referir a los atributos, extremos de asociación, métodos y operaciones libres de efectos colaterales como *propiedades*. Por consiguiente una propiedad puede ser una de las siguientes opciones:

- un Atributo,
- un Extremo de Asociación,
- una Operación con isQuery en verdadero,
- un Método con isQuery en verdadero

Propiedades

El valor de una propiedad para un objeto que está definido en un diagrama de clases se especifica con un punto seguido del nombre de la propiedad:

UnTipo
self.propiedad

Si self es una referencia a un objeto, entonces *self.propiedad* es el valor de la propiedad *propiedad* para self.

Propiedades: atributos

Por ejemplo, la vida de un Pacman se escribe como:

Pacman
self.vida

El valor de esta expresión es el valor del atributo *vida* para Pacman. El tipo de esta expresión es el tipo del atributo vida, que es del tipo básico **Entero**.

Propiedades:operaciones

Las operaciones pueden tener parámetros. Por ejemplo, en el diagrama mostrado anteriormente, un objeto Pacman tiene una función Camina_Pacman . Esta operación puede ser expresada como sigue, para una instancia Pacman: *unPacman* y *Posición*:

unPacman.Camina_Pacman(Posición)

Propiedades: extremo de asociación y navegación

Comenzando desde un objeto en particular, podemos navegar una asociación en un diagrama de clases para referirnos a otro objeto y a sus propiedades. Para hacerlo, se navega la asociación usando su extremo opuesto :

objeto.nombreDeRol

El valor de esta expresión es el conjunto de objetos que están del otro lado de la asociación *nombreDeRol*. Si la multiplicidad del extremo de la asociación tiene un máximo de uno ("0..1" o "1"), entonces el valor de la expresión es un objeto. En el diagrama de clases del ejemplo, si empezamos en el contexto de Pacman (self es una instancia de Pacman), podemos escribir :

Pacman
self.Come -- es de tipo Set(Fantasma)

La evaluación de la expresión resultará en un conjunto de Fantasmas. Por defecto, la navegación resultará en un conjunto. Cuando la asociación, en el diagrama de clases, está adornada por {ordered), la navegación resultará en una Sequencia (Sequence). Las Colecciones, así como los Set, las Bag y las Sequence, son tipos predefinidos en OCL. Tienen un gran número de operaciones predefinidas. Una propiedad de la colección en sí es accedida utilizando la flecha, '->' seguida del nombre de la propiedad. El siguiente ejemplo es en el contexto de Pacman:

Pacman
self.Come->tamaño

El ejemplo aplica la propiedad *tamaño* al Set *self.Come*, que da como resultado el número de Fantasmas que come Pacman.self

Pacman
self.Come->esVacio

Este ejemplo aplica la propiedad *esVacio* al Set *self.Come*. El resultado es verdadero si el conjunto de Fantasmas es vacío, y falso si no lo es.

Navegación hacia tipo asociación

Para especificar la navegación en asociaciones como clase, OCL usa el punto y el nombre de la asociación como clase comenzando con minúscula. Las asociaciones como clase no tienen un nombre de rol específico en el diagrama de clases.

Navegación desde la asociación como clase

Se puede navegar desde la asociación como clase hacia el objeto que participa en la asociación. Esto se hace utilizando el punto y el nombre de rol del extremo de la asociación.

La navegación desde una asociación como clase a uno de los objetos de la asociación siempre entregará exactamente un objeto. Este es el resultado de la definición de una asociación como clase. Por lo tanto el resultado de esta navegación es exactamente un objeto, aunque también puede usarse como un Conjunto utilizando la flecha (->).

E**xpresiones generales**

Cualquier expresión OCL puede ser usada como valor para un atributo en una *expresión* de una clase del UML o sus subtipos. En este caso, el documento *Semántica del UML* describe el significado de la *expresión.*

4.3.4 T**ipos predefinidos en OCL**

Se define dentro del OCL los tipos de datos básicos, así como las operaciones que pueden realizar, los cuales son mostrados en la siguiente tabla:

TIPO	VALOR	OPERACION
Boolean (Booleano)	True, false	And, or, xor, not
Integer (Entero)	1, 2, 34, 2656....	*, +, /, abs
Real (Flotante)	1.5, 3.14, 6.78...	*, +, -, /, floor(base)
String (Cadena)	To be, or, not	toUpper, concat

Tabla # 1

Teniendo la definición de los tipos ya mencionados se hace referencia a sus operaciones, propiedades o características más básicas, mediante las siguientes tablas.

- Real
- Entero
- Cadena
- Boolean

Real:

En OCL el tipo Real representa el concepto matemático de Real. Tome en cuenta que Entero es subclase de Real, por lo tanto en cada parámetro de tipo Real, puede usar un Entero. Características de Real, una instancia de Real es llamada r.

Operación	**Interpretación**
r = (r2 : Real) : Boolean	Verdadero si *r* es igual a *r2*.
r + (r1 : Real) : Real	El valor de sumar *r* y *r1*.
r - (r1 : Real) : Real	El valor de restar *r1* a *r*.
r * (r1 : Real) : Real	El valor de multiplicar *r* y *r1*.
r / (r1 : Real) : Real	El valor de *r* divido *r1*.

Tabla # 2

Integer:

El tipo Entero del OCL representa el concepto matemático de entero.
Características de Entero, una instancia de Entero es llamada *i*.

Operación	Interpretación
i = (i2 : Integer) : Boolean	Verdadero si *i* es igual a *i2*.
i + (i2 : Integer) : Integer	El valor de sumar *i* e *i2*.
i + (r1 : Real) : Real	El valor de sumar *i* y *r1*.
i - (i2 : Integer) : Integer	El valor de restar *i2* de *i*.
i - (r1 : Real) : Real	El valor de restar *r1* de *i*.

Tabla # 3

String:

El tipo Cadena de OCL representa a los caracteres ASCII.
Características de Cadena, una instancia de Cadena es llamada String.

Operación	Interpretación
string = (string2 : String) : Boolean	Verdadero si s*tring* y s*tring2* contienen los mismos caracteres, en el mismo orden.
string.size : Integer	El número de caracteres que contiene s*tring*.
string.substring (lower : Integer, upper : Integer) : String	El substring de s*tring* que comienza en la letra número *inferior*, hasta la letra número *superior* inclusive.
string.concat(string2 : String) : String	La concatenación de s*tring* y s*tring2.* Post: result.size = string.size + string2.size Post: result.substring(1, string.size) = string Post: result.substring(string.size + 1, string2.size) = string2
string.toUpper : String	El valor de s*tring* con todas sus minúsculas convertidas a mayúsculas. Post: result.size = string.size

Tabla # 4

Boolean

El tipo Boolean del OCL representa los valores verdadero/falso.
Características de Boolean, una instancia de Boolean es llamada *b*

Operación	Interpretación
b = (b2 : Boolean) : Boolean	Verdadero si *b* es el mismo que *b2*.
b or (b2 : Boolean) : Boolean	Verdadero si o *b* o *b2* son verdaderos.
b xor (b2 : Boolean) : Boolean Post: (b o b2) y no (b = b2)	Verdadero si o *b* o *b2* son verdaderos, pero no los dos.
b and (b2 : Boolean) : Boolean	Verdadero si ambos, *b1* y *b2*, son verdaderos.
not b : Boolean	Verdadero si *b* es falso Post: si b entonces resultado = falso sino resultado = verdadero finSi

Tabla # 5

Los tipos básicos usados son Integer, Real, String, y Boolean. Son suplementados con OclExpression, OclType y OclAny.

OclType:

Todos los tipos definidos en un modelo UML, o predefinidos dentro del OCL tienen un tipo. Este tipo es una instancia del tipo OCL llamado OclType. El acceso a este tipo le permite al modelador acceder al meta-nivel del modelo. Esto puede ser útil para modeladores avanzados.
Características de OclType, una instancia de OclType es llamada *type*.

Operación	Interpretación
type.name : String	El nombre del *type*.
type.attributes : Set(String)	El conjunto de nombres de los atributos de *type*, tal como están definidos en el modelo.
type.AssociationEnds: Set(String)	El conjunto de nombres del los AssociationEnds navegables de *type*, tal como están definidos en el modelo.
type.operations : Set(String)	El conjunto de nombres de las operaciones de *type*, tal como están definidas en el modelo.
type.allSupertypes : Set(OclType)	La clausura transitiva del conjunto de todos los supertipos de *type*.
type.supertypes : Set(OclType)	El conjunto de todos los supertipos directos de *type*. Post: type.allSupertypes->includesAll(result)

type.allInstances : Set(Type)	El conjunto de todas las instancias de *type* y todos sus subtipos.

Tabla # 6

OclAny:

En el contexto del OCL, el tipo OclAny es el supertipo de todos los tipos del modelo. Las características de OclAny están disponibles para cada objeto en todas las expresiones OCL.
Todas las clases en un modelo UML heredan todas las características definidas para OclAny. Para evitar conflictos de nombres entre características del modelo y características heredadas de OclAny, todos los nombres de las características de OclAny comienzan con 'ocl'. De todas maneras, teóricamente aún podrían existir conflictos de nombres, pero pueden evitarse usando las construcciones con caminos para referirse explícitamente a las propiedades de OclAny.
Características de OclAny, una instancia de OclAny es llamada *object*.

Operación	**Interpretación**
object = (object2 : OclAny) : Boolean	Verdadero si *object* es el mismo objeto que *object2*.
object <> (object2 : OclAny) : Boolean	Verdadero si *object* es un objeto diferente de *object2*. Post: result = not (object = object2)
object.oclAny : OclAny	El tipo del *object*
object.oclIsKindOf(type : OclAny) : Boolean	Verdadero si *type* es un supertipo (transitivo) del tipo de *object*. Post: result = typeallSuperTypes-> includes(object.oclType) or type = object-> oclType
object.oclIsTypeOf(type : OclType : Boolean	Verdadero si *type* es igual al tipo de *object.* Post: result = (object.oclType = type)
object.oclIAsType(type : OclType) : type	Da como resultado *object*, pero conociendo que su tipo es *type*. El resultado es Indefinido si el tipo real del *object* no es tipo o alguno de sus subtipos. pre : object.oclIsKindOf(type) post: result = object post: result.oclIsKindOf(type)

Tabla # 7

OclExpression:

Cada expresión OCL en sí es un objeto en el contexto del OCL, el tipo de la expresión es OclExpression. Este tipo y sus características se usan para definir la semántica de aquellas características que toma una expresión como uno de sus parámetros: select, collect, forAll, etc.
Una OclExpression opcionalmente incluye la variable iterador y tipo, y opcionalmente la variable acumulator y type.
Características de OclExpression, una instancia de OclExpression es llamada *expression*.

Operación	**Interpretación**
Expresión.evaluationType : OclType	El tipo del objeto que resulta de evaluar la *expression*.

Tabla # 8

Dentro de los tipos predefinidos para OCL también se encuentran las Colecciones que juegan un rol muy importante en las expresiones OCL debido a que una navegación frecuente da como resultado una Colección.

Collection: es un tipo abstracto con subtipos concretos, se distinguen 3 tipos diferentes de colecciones:

- Set: Conjunto matemático, que no contiene elementos duplicados
- Bag: Conjunto el cual puede contener elementos duplicados. El mismo elemento puede estar en la bolsa 2 o más veces.
- Sequence: Es como una bolsa, la diferencia es que los elementos están en forma ordenada.

TIPO	EJEMPLO
SET	Set { 1, 2, 5, 8 }
SEQUENCE	Sequence { 1, 3, 45, 2, 3 }
BAG	Bag { 1, 3, 4, 2,1 }

Tabla # 9

A continuación se describe una lista completa de las operaciones de Collection y sus subtipos.

Collection:

En OCL, collection es el supertipo abstracto de todas las colecciones. Cada ocurrencia de un objeto en una colección es llamada elemento. Si un objeto ocurre dos veces en una colección, hay dos elementos.
Características de Collection, una instancia de Collection es llamada *collection*.

Operación	Interpretación
collection>iterate(expr : OclExpression) : expr.evaluationType	Itera sobre la colección. Esta es la operación básica de las colecciones a partir de la que las demás pueden describirse.
collection->size: Integer	El número de elementos en la colección *collection* post: result = collection->iterate(elem; acu : Integer = 0 | acu + 1)
collection->count(object:oclAny):Integer	El número de veces que el objeto aparece en la colección *collection.* post:result=collection->iterate (elem;acu:Integer=0|if elem=object then acu+1 else acu end if)
	Verdadero si *object* es un elemento de *collection*, falso si no. post: result = (collection->count(object) > 0)
collection->isEmpty : Boolean	¿La *collection,* no es la colección vacía ? post: result = (collection->size <> 0)
collection->forAll(expr : OclExpression) : Boolean	Es verdadero si *expr* evalúa en verdadero para cada elemento de la *collection*. Si no es así el resultado es falso. post: result = collection->iterate(elem; acu : Boolean = true | acu and expr)

Tabla # 10

Set:

El Set representa el conjunto matemático. Contiene elemento no duplicados.
Características de Set, una instancia de Set es llamada *set.*

Operación	Interpretación
set->union(set2 : Set(T)) : Set(T)	La unión de *set* y *set2* post: T.allInstances->forAll(elem | result->includes(elem) = set-> includes(elem) or set2->includes(elem))
set->intersection(set2 : Set(T)) : Set(T)	La intersección de *set* y *set2*. O sea, el conjunto de todos los elementos comunes de *set* y *set2*. post: T.allInstances->forAll(elem | result->includes(elem) = set->includes(elem) and set2->includes(elem))
set->count(object : T) : Integer	El número de ocurrencias de *object* en el *set* post: result <= 1

Tabla # 11

Bag:

Una bag es una colección en la que los duplicados están permitidos. Esto es, un objeto puede ser un elemento de una bag muchas veces. No hay un orden preestablecido para los elementos de una bag.
Características de Bag, una instancia de Bag es llamada *bag*.

Operación	Interpretación
bag->count(object : T) : Integer	La cantidad de ocurrencias de *object* en *bag*.
bag = (bag2 : Bag) : Boolean	Verdadero si *bag* y *bag2* contienen los mismos elementos, la misma cantidad de veces. post: result = T.allInstances->forAll(elem | bag->count(elem) = bag2->count(elem))
bag->intersection(bag2 : Bag) : Bag(T)	La intersección de *bag* y *bag2* Post: T.allInstances->forAll(elem | resul->count(elem) = bag->count(elem).min(bag2->count(elem)))

Tabla # 12

Sequence:

Una sequence es una colección donde los elementos están ordenados. Un elemento puede ser parte de una sequence más de una vez.
Características de Sequence(T), una instancia de Sequence es llamada *sequence*.

Operación	Interpretación
sequence->count(object : T) : Integer	El número de ocurrencias de *objeto* en la *sequence.*
sequence->at(i : Integer) : T	El *i-ésimo* elemento de *sequence*. post: i <= 0 o sequence->size < i implies result = Undefined
sequence->first : T	El primer elemento de *sequence* post: result = sequence->at(1)

Tabla # 13