# Reduciendo la Ambigüedad en el Modelo del Dominio mediante invariantes OCL

Elizabeth Vidal-Duarte*
evidal@ucsp.edu.pe

## Resumen

Uno de los principales inconvenientes en la especificación de requerimientos de software es que los diagramas pueden estar sujetos a interpretaciones subjetivas. Esto podría llevar a implementaciones que corren el riesgo de no cumplir con los requerimientos reales. Este artículo busca reducir la ambigüedad en la especificación de requerimientos mediante la aplicación del lenguaje de especificación formal OCL. Nuestro trabajo se ha centrado en la especificación de restricciones en forma de invariantes aplicado al Modelo del Dominio. La aplicación de OCL ayuda a que las reglas del negocio queden claras e explícitas como parte de la especificación.

## Palabra clave:

Análisis de Sistemas Especificación de Requerimientos, Especificación Formal, Ingeniería del Software, OCL, Modelamiento, Métodos Formales.



## 1. Introducción

La parte más crítica del desarrollo de software corresponde a la identificación y especificación de requerimientos [1]. Para facilitar el proceso de especificación de requerimientos se hace uso de modelos y diagramas. Aunque el Lenguaje de Modelado Unificado (UML) [2] es el lenguaje estándar para modelamiento aún no está suficientemente refinado como para proveer toda la información relevante en una especificación. Uno de los modelos UML generados en la especificación de requerimientos es el Modelo del Dominio (MD) [3,4]. El MD modela los principales conceptos del negocio en forma clases y, la relación existente entre dichos conceptos en forma de asociaciones. El MD está sujeto muchas veces a interpretaciones subjetivas. Esto podría llevar a implementaciones que corren el riesgo de no cumplir con los requerimientos reales.

Existe la necesidad de describir restricciones adicionales acerca de los objetos en el modelo. Dichas restricciones se refieren por lo general a las reglas del negocio. Muchas veces las restricciones son descritas en lenguaje natural, mediante los elementos de anotación de UML [2]. La experiencia ha mostrado que esto puede resultar en ambigüedades. Para evitar dichas ambigüedades nuestra propuesta hace uso del lenguaje de especificación formal OCL (Object Constraint Language) [5].

Si bien es cierto que existen otros lenguajes de especificación formal tales como RAISE [6], VDM [7], Z [8], Object Z [9], JML [10] entre otros, la ventaja que se encontró en OCL es que se aplica directamente sobre

*Universidad Católica San Pablo

diagramas UML. Además, OCL permite aplicar restricciones sobre las asociaciones del Modelo del Dominio.

La principal contribución de este artículo es mostrar, de una forma clara y sencilla, la aplicabilidad de especificaciones formales en el MD. Si bien nuestra especificación no cubre el comportamiento del sistema, si permite mostrar cómo pueden capturarse las restricciones del negocio. Lo cual no es posible utilizando solamente el MD y los elementos de anotación.

El resto del artículo está organizado de la siguiente manera: en la Sección 2 se explican las principales características de OCL, y, la sintaxis y semántica que será utilizada para la especificación de restricciones en forma de invariantes. En la Sección 3 se presenta como caso de estudio un subsistema de atención a clientes de un banco. Se describen los requerimientos funcionales de manera informal para luego presentar el Modelo del Dominio junto con las restricciones en OCL. Finalmente en la Sección 5 se exponen las conclusiones.

## 2. Object Constraint Language (OCL)

OCL [11] parte del estándar UML, es un lenguaje que nos permite la especificación de restricciones formales en el contexto de los modelos UML. Las restricciones son condiciones en todos los estados y las transiciones entre los estados de un sistema implementando un modelo determinado [12].

Las restricciones son usadas principalmente para expresar invariantes en las clases y precondiciones y poscondiciones en las operaciones. Una invariante es una expresión referida a todos los objetos en una clase. Adicionalmente, precondiciones y postcondiciones nos permiten determinar especificaciones sobre el comportamiento de las operaciones antes y después de su ejecución [12].

Según [13] el lenguaje provee variables y operaciones que pueden ser combinadas para construir expresiones. OCL define un número de tipos de datos que van desde enteros y booleanos hasta tipos que manejan colecciones de objetos. Todas las expresiones OCL son libres de efectos secundarios, esto es, la evaluación de una expresión no produce un cambio en el estado del sistema. Las expresiones OCL son declarativas en el sentido que nos dicen que restricciones deben permanecer, pero no como deben ser implementadas.

La Figura 1 muestra un subconjunto del MD del caso de estudio presentado en [13]. Utilizaremos este diagrama para ilustrar los conceptos de OCL relevantes para nuestro trabajo: contexto, instancia de clase, invariantes y restricciones sobre asociaciones. Brevemente resumimos los principales aspectos del caso de estudio: la empresa L&R maneja un "Sistema de Lealtad" que ofrece a sus principales clientes varios programas de lealtad por el que obtienen diversos tipos de bonos (descuentos, premios, puntos, etc). Cada cliente entra al programa llenando un formulario por el cual obtiene una tarjeta de membresía. Es necesario que cada cliente sea mayor de edad

para poder inscribirse en el programa [13]. Para facilitar la compresión del artículo se presentan las palabras reservadas de OCL en letra itálica negrita. La semántica de las expresiones se presenta como comentarios en letras itálicas precedidas por los caracteres "–".

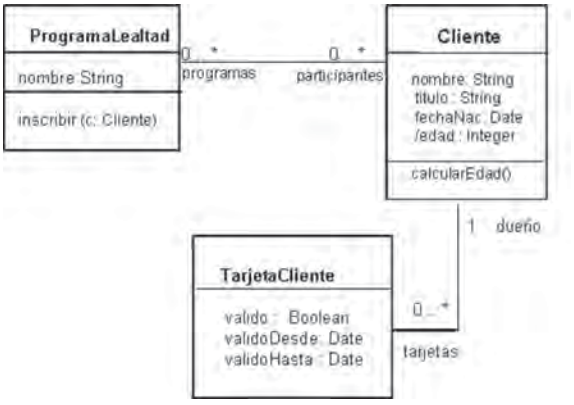


Fig. 1. Modelo del Dominio Parcial caso "Sistema de Lealtad"

## A. Contexto

La declaración del contexto especifica el elemento del modelo en el cual será definida la restricción. Para las invariantes la declaración del contexto será una clase tomada del modelo del dominio [13]. Tomando como referencia la Fig. 1 tomamos la clase Cliente como contexto. La sintaxis se presenta en la Fig. 2.

```
context Cliente
– El contexto es la clase Cliente
```

Fig. 2. Sintaxis de declaración del contexto (una clase)

## B. Contexto de instancia

Cada expresión OCL es escrita en el contexto de una instancia de un tipo específico [13]. Utilizamos la palabra reservada self para referirnos a esa instancia. En la Fig. 3 se toma el atributo edad de una instancia de la clase Cliente:

```
context Cliente
self.edad
– El valor de la subexpresión self.edad se refiere al
valor –del atributo edad en una instancia particular
de la
–clase Cliente. El tipo de esta expresión es el mismo
tipo
– del atributo edad que en - este caso es del tipo
Entero.
```

Fig. 3. Sintaxis de declaración de una instancia de clase referido a un determinado atributo

### C. Invariantes

Las invariantes son condiciones que deben ser verdaderas durante el tiempo de vida del sistema para todas las instancias de una determinada clase. La condición es expresada con una expresión de tipo booleano. De acuerdo a los requerimientos expuestos se sabe que los clientes deben tener más de 18 años de edad para poder ser parte del programa. La Fig. 4 expresa dicha restricción en forma de invariante

```
context Cliente
inv: self.edad > = 18
– La edad del cliente debe ser mayor o igual – a 18 años.
```

Fig. 4. Sintaxis de aplicación de invariante en una clase del Modelo del Dominio

La semántica de las invariantes requiere que la expresión sea verdadera para todos los objetos de la clase dada como contexto [13].

### D. Restricciones sobre Asociaciones

En este caso el contexto es la clase TarjetaCliente. La expresión se refiere a una relación de asociación entre objetos. Para acceder al atributo del objeto asociado (en este caso Cliente) hacemos uso del "rolname" [2] dueño. Esta es la forma en la que OCL nos permite realizar navegabilidad entre los diferentes objetos del modelo.

```
context  TarjetaCliente
inv : self.dueño.edad >= 18
– La edad del cliente dueño de la tarjeta debe ser mayor o
– igual a 18 años
```

Fig. 5. Sintaxis de aplicación de invariantes en una asociación del Modelo de Dominio

### E. Expresiones Booleanas y Tipos

Las expresiones en OCL son construidas utilizando los operadores booleanos: and, or, not e implies. OCL define tipos de datos básicos tales como Int, Bool y String. También define tipos que nos permiten manejar colecciones de objetos [12]. OCL soporta los tipos colección: Set, Bag y Sequence. Un tipo Set se refiere a conjuntos según definición matemática (no contiene elementos duplicados). El tipo Bag es como un conjunto, pero puede tener elementos duplicados. El tipo Sequence es como el tipo Bag, pero sus elementos están ordenados. OCL presenta operaciones predefinidas que son comunes a todos los tipos colección [12][13] (ver Tabla I).

Tabla I. Operaciones comunes a todos los tipos Colección

| Signatura | Semantica |
|---|---|
| size : $Collection(t) \rightarrow Integer$ | $\|C\|$ |
| count : $Collection(t) \times t \rightarrow Integer$ | $\|C \cap \{v\}\|$ |
| includes : $Collection(t) \times t \rightarrow Boolean$ | $v \in C$ |
| excludes : $Collection(t) \times t \rightarrow Boolean$ | $v \notin C$ |
| includesAll : $Collection(t) \times Collection(t) \rightarrow Boolean$ | $C_2 \subseteq C_1$ |
| excludesAll : $Collection(t) \times Collection(t) \rightarrow Boolean$ | $C_2 \cap C_1 = \emptyset$ |
| isEmpty : $Collection(t) \rightarrow Boolean$ | $C = \emptyset$ |
| notEmpty : $Collection(t) \rightarrow Boolean$ | $C \neq \emptyset$ |
| sum : $Collection(t) \rightarrow t$ | $\sum_{i=1}^{\|C\|} c_i$ |

La navegación directa en el diagrama de clases UML resulta en un conjunto (tipo Set de OCL). Así por ejemplo en el diagrama de la Fig, 1 podemos decir que un objeto de la clase ProgramaLealtad tiene un conjunto de clientes. Si quisiéramos añadir una restricción en donde cada objeto del ProgramaLealtad debe contener al menos 50 clientes, nuestra invariante sería la presentada en la Fig. 5.

```
context ProgramaLealtad
inv: self.participantes->size() >= 50
– En  el contexto de la clase  ProgramaLealtad  para
– acceder a  la  clase Cliente se hizo uso del rolname
– participantes. Size() es una operación
- -predefinida sobre el tipo colección presentado en la Tabla I.
```

Fig. 5: Ejemplo de aplicación de restricción utilizando operaciones predefinidas sobre colecciones

La navegación entre asociaciones rotuladas con {ordered} en el Modelo del Dominio [2] resulta en el tipo (Sequence). En la Tabla 2 presentamos las operaciones para el tipo Secuencia por ser relevantes para nuestro caso de estudio. Un ejemplo de uso de {ordered} se aprecia en la Fig. 6.

En [13] se encuentra el detalle y explicación de todas las operaciones sobre colecciones. El ejemplo aclaratorio para las operaciones sobre el tipo Sequence será expuesto en la siguiente sección.

Tabla II. Operaciones definidas para el tipo Sequence

| Signatura | Semantica |
|---|---|
| union : $Sequence(t) \times Sequence(t) \rightarrow Sequence(t)$ | $S_1 \circ S_2$ |
| append : $Sequence(t) \times t \rightarrow Sequence(t)$ | $S \circ \langle e \rangle$ |
| prepend : $Sequence(t) \times t \rightarrow Sequence(t)$ | $\langle e \rangle \circ S$ |
| subSequence : $Sequence(t) \times Integer \times Integer \rightarrow Sequence(t)$ | $\pi_{i,j}(S)$ |
| at : $Sequence(t) \times Integer \rightarrow t$ | $\pi_i(S)$ |
| first : $Sequence(t) \rightarrow t$ | $\pi_1(S)$ |
| last : $Sequence(t) \rightarrow t$ | $\pi_n(S)$ |
| including : $Sequence(t) \times t \rightarrow Sequence(t)$ | $S \circ \langle e \rangle$ |
| excluding : $Sequence(t) \times t \rightarrow Sequence(t)$ | $S - \langle e \rangle$ |
| asSet : $Sequence(t) \rightarrow Set(t)$ | |
| asBag : $Sequence(t) \rightarrow Bag(t)$ | |

## 3. Aplicacion: subsistema de atención de clientes bancarios

### a. Los Requerimientos

Para poder mostrar nuestra propuesta, hemos considerado el caso de un (simple) subsistema de atención de clientes de un banco. Asumimos que el banco en estudio tiene varias estaciones de atención denominadas ventanillas. Un cliente se acerca a una ventanilla para ser atendido.

Se sabe que una ventanilla tiene asociada: a) una fila de clientes b) un estado (cerrada: no atiende clientes, abierta: nuevos clientes pueden colocarse en la fila o cerrando: atiende a los clientes que están actualmente en la fila pero no permiten que nuevos clientes se sumen a la fila) y c) disponibilidad (ocupada o libre).

### b. El Modelo del Dominio

Siguiendo el Proceso de Desarrollo Unificado de Software [4], un Modelo de Dominio es presentado por un diagrama de clases simplificado. Esto es, debe contener clases con sus respectivos atributos, relaciones de asociación con otras clases. Además cada asociación detallada su respectivo rolname y multiplicidad.

Como parte esencial de este artículo, nuestro subsistema también incluirá invariantes OCL para especificar propiedades/restricciones del negocio. El Modelo del Dominio para nuestro subsistema se muestra en la Fig 6.

Es importante resaltar que los componentes de la clase Ventanilla deben satisfacer varias restricciones de manera que la descripción de su comportamiento se ajuste al mundo real. Por ejemplo si la ventanilla está cerrada, entonces no puede estar actualmente atendiendo a un cliente y además su estado debe ser libre. Si está actualmente ocupada debe haber algunos clientes que está siendo atendidos, lo que significa que la fila no puede estar vacía. Dichas restricciones no pueden ser deducidas a simple vista en el Modelo del Dominio.

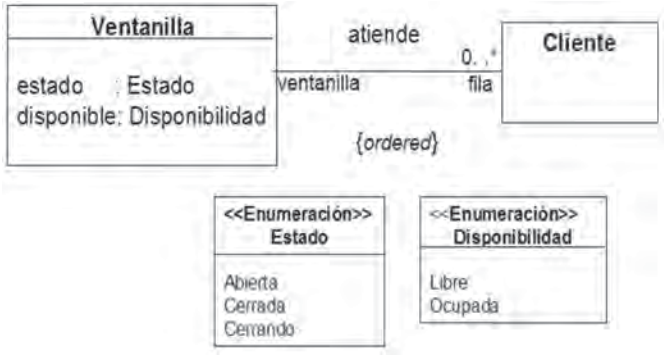


Fig. 6. Modelo del Dominio Parcial Subsistema de Atención de Clientes

En la Fig. 7 utilizamos OCL para describir en forma precisa las restricciones en forma de invariantes aplicadas al Modelo del Dominio de la Fig. 6.

```
context: Ventanilla
inv: self.estado=cerrada  implies
self.fila    isEmpty() and self.disponible=libre
context: Ventanilla
inv: self.disponible=ocupada  implies
 self.fila    notEmpty()
```

Fig 7. Restricciones OCL para el Subsistema Atención de Clientes

### c. Análisis

De acuerdo con las invariantes propuestas en 3.2 podemos ver que la primera invariante expresa que cuando una ventanilla está cerrada no debe haber una fila de clientes esperando ser atendidos. Esta restricción, en la fase de implementación, se refiere a que no deben haber tickets de atención generados para dicha ventanilla. La segunda parte de la expresión refuerza la invariante al asegurar que una ventanilla cerrada no esta atendiendo a ningún cliente.

La segunda invariante expresa que si una ventanilla está ocupada la fila de clientes no puede estar vacía. Esto es, debe existir al menos un cliente asociado a dicha ventanilla.

Si bien las restricciones realizadas son bastante sencillas hemos podido mostrar como reforzar nuestro Modelo del Dominio negocio incluyendo algunas invariantes. Esto permite reducir en alguna medida la ambigüedad del modelo.

Estudios en [12] han demostrado que incluyendo simples invariantes en la especificación y utilizando herramientas de verificación permite incrementar la correctitud del software. Una herramienta que nos permite verificar expresiones OCL  es Octopus (OCL Tool for Precise UML Specifications) [14]. Octopus nos permite verificar de manera estática expresiones OCL. Esto es, verifica la sintaxis de las expresiones, así como sus tipos y el correcto uso de los elementos del modelo: rolname, asociaciones y atributos. La descripción y el análisis de los resultados de la aplicación de la herramienta Octopus está fuera del alcance de este artículo.

## 4. Conclusiones

En este artículo se ha presentado una forma de especificar restricciones en forma de invariantes a un Modelo del Dominio. Esto nos permite reducir la ambigüedad durante la etapa de análisis.

Si bien OCL presenta muchas más funcionalidades que las descritas en este artículo, hemos tomado solo la especificación de invariantes para

mostrar cómo especificar restricciones en el Modelo del Dominio. Nuestro objetivo es reducir la ambiguiedad durante la etapa de análisis. Creemos que el uso invariantes en las primeras etapas del desarrollo nos permite incrementar la correctitud del software que estamos desarrollando.

El principal problema encontrado durante el desarrollo del caso de estudio fue el manejo de los tipos colección (Set y Sequence). La correcta aplicación de restricciones OCL sobre estos tipos requiere que el Modelo del Dominio esté completo. Es decir que tenga navegabilidad, rolname y si fuera el caso etiquetas del tipo {ordered}[2]. Si no tenemos el Modelo del Dominio detallado, nuestras especificaciones OCL podrían no ser lo suficientemente exactas en cuanto a las restricciones en las asociaciones.

Por otro lado aun teniendo un Modelo del Dominio básico todavía es posible definir invariantes que no involucren tipos colección. Con esto queremos decir que siempre es posible incluir especificaciones formales en su forma más básica. Creemos firmemente que una especificación formal es mejor que ninguna.

## 5. Referencias


1. Sommerville, I.: Software Engineering, Sixth Edition. Addison Wesley, (2001)

2. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley (1998)

3. Larman, C.: Applying UML and Patterns. Prentice-Hall International, (1998)

4. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, (1999)

5. Warmer, J., Kleppe, A.: OCL: The constraint language of the UML. Journal of Object- Oriented Programming, 12, pp. 10–13,28 ( May 1999)

6. The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall (1992)

7. VDM and VDM++, http://www.csr.ncl.ac.uk/vdm

8. Spivey,J. M.: The Z Notation: a reference manual (2nd edition ed.). Prentice Hall International Series in Computer Science (1992)

9. Smith, G.: The Object-Z Specification Language. Springer (1999)

10. Burdy, L. et al: An Overview of JML and applications. Intenational Journal on Software Tools for Technology Transfer, Voume 7 Issue 3, Spinger-Verlang (June 2005)

11. Hamie, A., Howse, J., Kent S.: "Interpreting the Object Constraint Language", Division of Computing, University of Brighton, Lewes Rd., Brighton, UK.

12. Richters, M.. A Precise Approach to Validating UML Models and OCL Constraints", PhD Thesis Universitat Bremen, Fachbereich, (2004)

13. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML, Addison-Wesley, (1998)

14. Octopus, http://www.klasse.nl/octopus/