Enl@ce: Revista Venezolana de Información, Tecnología y Conocimiento
ISSN: 1690-7515
Depósito legal pp 200402ZU1624
Año 8: No. 1, Enero-Abril 2011, pp. 31-54

Cómo citar el artículo (Normas APA):
Rodríguez, R. y Goncalves, M. (2011). Implementación de requisitos en sistemas orientados a datos con lenguaje OCL y lógica difusa. *Enl@ce Revista Venezolana de Información, Tecnología y Conocimiento*, 8 (1), 31-54

Implementación de requisitos en sistemas orientados a datos con lenguaje OCL y lógica difusa[1]

Rosseline Rodríguez[2]
Marlene Goncalves[3]

Resumen

Los sistemas clásicos permiten realizar consultas precisas sobre bases de datos. Sin embargo, durante la especificación de los requisitos, el usuario puede utilizar términos lingüísticos vagos, que son propios del lenguaje natural e involucran términos imprecisos o difusos. Contrariamente a lo necesitado, pocas son las metodologías de consulta de bases de datos que incluyen requisitos difusos provenientes del lenguaje natural. En este trabajo, se propone un método basado en el lenguaje formal OCL y lógica difusa para el desarrollo de sistemas orientados a datos que permite tratar la ambigüedad propia del lenguaje natural en la implementación de requisitos difusos. Finalmente, se presenta un caso de estudio real para ilustrar el uso del método propuesto.

Palabras clave: OCL, consultas difusas, requisitos difusos, sistemas orientados a datos, SQLf.

Recibido: 03-07-10 Aceptado: 17-02-11

[1] Proyecto de investigación "Creación y Aplicación de Manejadores de Bases de Datos Difusas", subvencionada por el Fondo Nacional de Ciencia, Tecnología e Investigación, FONACIT. Registrado bajo el número G-2005000278.

[2] Ingeniera en Computación. Maestría en Ciencias de la Computación. Profesora de la Universidad Simón Bolívar.
Correo electrónico: crodrig@usb.ve

[3] Doctorado en Computación. Profesora Universidad Simón Bolívar. Centro de Análisis, Modelado y Tratamiento de Datos, CAMYTD.
Correo electrónico: mgoncalves@usb.ve

Implementation of Requirements in Data-Oriented Systems with OCL and Fuzzy Logic

Abstract

Classical data-driven systems perform crisp queries on databases. However, users can use vague or fuzzy linguistic terms during the specification of natural language requirements. These user requirements involving vague linguistic terms are called fuzzy requirements and require support fuzzy queries to database. Few efforts have been made in methodologies to include fuzzy requirements during the software development process naturally. In this work, we propose a method based on the formal language OCL and fuzzy logic for the development of data-oriented systems that require the support of fuzzy requirements. Through the OCL language and fuzzy logic is the ambiguity of natural language itself. Finally, we present a real case study to illustrate the use of the proposed method.

Keywords: OCL, Fuzzy Queries, Fuzzy Requirements, Data-Driven Systems, SQLf

Introducción

Los sistemas clásicos orientados a datos recuperan información usando consultas precisas basadas en lógica booleana. Sin embargo, los requisitos de usuarios expresados en lenguaje natural pueden contener términos lingüísticos vagos. Estos términos representan criterios de preferencias de los usuarios sobre los datos y corresponden a conceptos cuyos límites no están claramente definidos. Estos conceptos pueden ser modelados usando conjuntos difusos (Zadeh, 1965).

Los requisitos de usuarios compuestos de estos conceptos son denominados requisitos difusos. Por ejemplo, un requisito difuso de usuario puede ser: conocer cuáles son los vehículos ***nuevos*** de precio ***bajo***. En este requisito, los adjetivos ***nuevos*** y ***bajo*** son términos lingüísticos vagos o difusos, puesto que alguien puede definir un vehículo ***nuevo*** de precio ***bajo*** como aquel cuyo año es mayor a 2008 y el precio está alrededor de Bs. 80.000, otra persona pudiera pensar que un carro ***nuevo*** de precio ***bajo*** es un vehículo de este año con un precio cercano a Bs. 100.000. Es por ello que la definición de los términos ***nuevos*** y ***bajo*** dependen de las preferencias del usuario.

Este requisito difuso de ejemplo se puede traducir en una consulta difusa a base de datos. En este sentido, varias extensiones a SQL (Structured Query Language) con lógica difusa han sido propuestas (Bordogna y Psaila, 2008; Bosc y Pivert, 1995; Galindo, Urrutia y Piattini, 2006). Entre todas estas propuestas, SQLf (Bosc y Pivert, 1995) es la más completa ya que incorpora la mayor cantidad de elementos de la lógica difusa, así como también es el único que está actualizado a los estándares de SQL 2003 (González, Goncalves y Tineo, 2009).

Debido a la ambigüedad propia del lenguaje natural, los requisitos difusos pueden especificarse en un lenguaje formal como OCL (Object Constraint Language) (Object Management Group, 2006). Además, los términos lingüísticos vagos en dichos requisitos pueden ser incorporados en tales especificaciones formales mediante la lógica difusa.

Por otra parte, pocos esfuerzos se han realizado en las metodologías para incluir requisitos difusos durante el proceso de desarrollo de software. En (Goncalves, Rodríguez y Tineo, 2009), los autores proponen una extensión a las metodologías de desarrollo de software para la especificación de requisitos difusos, de manera que los usuarios pueden manejar sus preferencias. En (Rodríguez y Goncalves, 2009), se introduce el perfil UML para el modelado visual de requisitos difusos que es el antecedente principal de este trabajo. Dicho perfil está basado en estereotipos y una extensión de OCL con lógica difusa, posee una semántica formal que permite tanto eliminar la ambigüedad de las descripciones en lenguaje natural como expresar las preferencias del usuario.

Ninguna de estas propuestas permite implementar los requisitos difusos con lenguajes difusos a bases de datos. Así, en este trabajo, se propone un método que integre el modelado, las especificación y traducción de requisitos difusos en el desarrollo de un sistema orientado a datos. Este método no considera aspectos metodológicos, como interfaces de usuarios, inserción de datos, entre otros; éstos pueden ser abordados en una metodología que cubra al sistema completo.

A continuación, en las siguientes secciones, se presentan los antecedentes de la investigación que permiten formalizar los requisitos difusos: la lógica difusa y el lenguaje OCL; posteriormente, se describe la metodología propuesta para implementar requisitos difusos en un sistema orientado a datos; luego se muestra un caso de estudio para observar la aplicabilidad de dicha metodología; y finalmente, las conclusiones de la investigación.

Antecedentes

Para la descripción de requisitos difusos de manera formal es necesario comprender los conceptos básicos de la lógica difusa que se presentan en esta sección. Asimismo, se requiere conocer el lenguaje de especificaciones formales OCL.

Lógica difusa

Los conjuntos difusos fueron introducidos por Zadeh (1965). Ellos están provistos de una gradualidad en la transición entre la inclusión y la exclusión completa de sus elementos. Para esto se hace uso de una función de membresía cuyo rango es el intervalo real [0,1], la cual se denota con el símbolo μ_F y sus formas de representación se describen en la **Tabla 1**.

Un ejemplo de una función de membresía es definida por un trapezoide como se muestra en la **Figura 1**. La teoría de Conjuntos Difusos permite definir una lógica difusa (Zadeh,1965), cuyo valor de verdad de la sentencia está comprendido en el intervalo real [0,1], donde 0 indica completamente falso y 1 es completamente cierto.

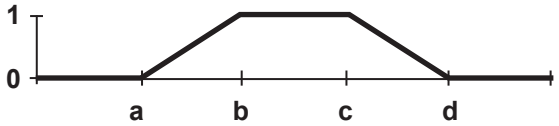
Tabla 1

Formas de representación de los conjuntos difusos

Forma de Representación	**Descripción**
Por Extensión	El conjunto difuso *F* sobre el universo $U=\{x_1,x_2...x_n\}$ se representa mediante la expresión $\mu_F=\{\mu_1/x_1, \mu_2/x_2...\mu_n/x_n\}$ donde $\forall i(\mu_i=\mu_F(x_i))$
Un Trapezoide	La función de membresía tiene forma trapezoide y se representa por la cuádrupla de valores del universo (a,b,c,d) que indican los puntos que definen el trapezoide, como se muestra en la Fig. 1.
Una Expresión	La función de membresía se define mediante una expresión aritmética. Por ejemplo, "$1/(x-1)$" con $x\in(1,\infty]$.

Figura 1

Representación trapezoide de un conjunto difuso



El valor de verdad de una sentencia *s* se denotado con $\mu(s)$. Esta lógica permite darle una interpretación a los llamados términos lingüísticos, tal como se describe en la **Tabla 2**.

La lógica difusa ha sido usada para extender el lenguaje de consulta a bases de datos SQL, que ha dado origen al SQLf (Bosc y Pivert, 1995). SQLf está dotado de construcciones adecuadas para la especificación de distintas clases de términos lingüísticos (Tineo, 1998). La estructura básica de una consulta SQLf es la siguiente:

SELECT <atts> **FROM** <rels> **WHERE** <FuzzyCond> **WITH CALIBRATION** [k|α|k,α];

El resultado de una consulta SQLf es el conjunto difuso de filas con los atributos proyectados de la cláusula SELECT sobre el producto cartesiano de las relaciones en la cláusula FROM que satisfacen la condición difusa de la cláusula WHERE. La cláusula WITH CALIBRATION es opcional e indica la escogencia de las mejores respuestas. Se han propuesto dos tipos de calibraciones: cuantitativa y cualitativa. La cuantitativa indica la escogencia de las mejores *k* respuestas, de acuerdo a su grado de satisfacción. La cualitativa indica la escogencia de las respuestas cuyo grado de pertenencia es mayor o igual a un nivel mínimo de satisfacción "α". Existen otras estructuras de consulta en SQLf que no explicamos aquí. Sin embargo, el lector interesado puede consultar en: Bosc y Pivert (1995), Goncalves y Tineo (2001a), Goncalves y Tineo (2001b) y González (2008).

Tabla 2

Términos lingüísticos presentes en la lógica difusa

Términos Lingüísticos	**Descripción**
Predicados	Componentes atómicos de la lógica difusa cuyo valor de verdad viene dado por la función de membresía del conjunto difuso. Pueden ser definidos sobre un conjunto difuso por extensión, con una expresión o de manera trapezoidal
Modificadores	Términos que permiten definir predicados modificados por medio de transformaciones sobre funciones de membresía. El modificador difuso transforma la función de membresía de un predicado difuso *P*. Esta transformación puede trasladar o desplazar la función con un valor *d* ($\mu_{\text{mod }P}(x)= \mu_P(x+d)$), o la potencia del grado de satisfacción de cada elemento del conjunto ($\mu_{\text{mod }P}(x)= (\mu_P(x))^n$ o $\mu_{\text{mod }P}(x)= (P\theta \ldots \theta P(x))$) donde θ se opera n veces y puede ser una t-norma (operador binario en el intervalo cerrado [0,1], conmutativo, asociativo, 1 como elemento neutral) o una s-norma (operador binario en el intervalo cerrado [0,1], conmutativo, asociativo, 0 como elemento neutral).
Comparadores	Clase de predicados difusos definido sobre pares de elementos; ellos establecen una comparación difusa. También, pueden ser definidos por extensión, o por una distancia o división entre pares de elementos comparados.
Conectores	Operadores definidos para la combinación de condiciones difusas. El conector difuso es definido por una expresión matemática que calcula el grado de satisfacción de cada elemento del conjunto.
Cuantificadores	Describen cantidades imprecisas y pueden ser absolutos o relativos. Los cuantificadores absolutos se refieren a la cantidad de elementos que cumplen con una condición; se representan como un conjunto difuso en el universo de los números reales. Los cuantificadores relativos se refieren a la proporción de elementos que cumplen con una condición, con respecto al número total de elementos del universo o un conjunto de referencia; éstos se representan como un conjunto difuso en el intervalo real [0,1].

El lenguaje OCL

OCL (Object Constraint Language) (OMG, 2006) es un lenguaje formal usado para describir expresiones sobre UML. Estas expresiones modelan condiciones invariantes, pre y post condiciones, así como consultas sobre los objetos del modelo.

En principio, las expresiones OCL son de la forma context TypeName inv Expression, siendo *context* e *inv* palabras reservadas del lenguaje; TypeName el nombre de la clase que representa el contexto y Expression la restricción cuyo resultado es un valor booleano. La declaración del contexto es opcional. La palabra reservada self indica una instancia dentro de la clase especificada por el contexto actual. Una alternativa a self es declarar una variable en el contexto para que juegue el rol de self. Así, la expresión OCL sería context var:TypeName inv Expression, donde var

es el nombre de la variable que sustituye a self. Se utiliza la palabra reservada pre precediendo la expresión, cuando se desea señalar que es una precondición; post para el caso de una postcondición; body para indicar que la expresión es el resultado de una consulta. Esta última permite especificar la semántica de los requisitos difusos. Si se quieren definir nuevos atributos y/o operaciones dentro de un contexto se utiliza def.

Adicionalmente, OCL permite el uso de colecciones. Los tipos colección (Set, OrderedSet, Bag, Sequence y Collection) se corresponden a tipos abstractos que contienen operaciones como: size, count, isEmpty, notEmpty, sum, exists, forAll, including, excluding, entre otras. El símbolo "->" es usado para indicar que una operación se aplica a una colección.

Una operación importante es TypeName. allInstances(), la cual devuelve el conjunto de todas las instancias del tipo TypeName, en el momento que la expresión es evaluada. Es posible componer muchos valores en una tupla, a través del constructor Tuple. Una tupla consiste de diferentes partes, cada una de las cuales tiene un nombre y un tipo, el cual es opcional. La sintaxis general del constructor es Tuple {$name_1$:$Type_1$=$value_1$,..., $name_N$:$Type_N$ = $value_N$}. TupleType combina tipos diferentes en un único tipo agregado.

Otro constructor importante para el manejo de colecciones es la operación select**.** Ésta permite especificar las instancias que nos interesan de una colección, en particular aquellas que formarán parte de la respuesta de un requisito. Su sintaxis es collection->select(v | logic-expression-with-v), siendo v la variable ligada que recorre las instancias de la colección. El resultado de esta operación es un sub-conjunto de collection, formado por las instancias que satisfacen la expresión lógica. Cuando se quiere especificar que una colección es derivada de diferentes colecciones se usa la operación collect. Su sintaxis es collection->collect(v | expression-with-v), y su resultado es la colección de todas las evaluaciones de la expression-with-v.

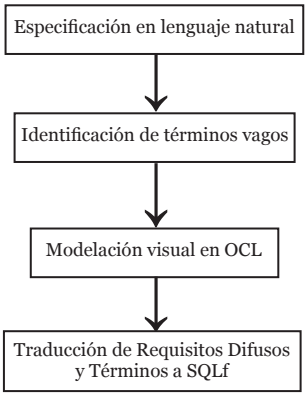
Metodología propuesta

Dentro de nuestro conocimiento, no existe una metodología que considere los requisitos difusos durante todo el proceso de desarrollo de software. Sin embargo, es posible encontrar trabajos que se centran en la especificación, modelación o implementación de requisitos difusos. En (Goncalves et al., 2009), se introduce la especificación de requisitos difusos donde los usuarios expresan sus preferencias mediante el uso de lógica difusa, mientras que en (Rodríguez y Goncalves, 2009), se describe cómo se pueden modelar visualmente los requisitos difusos. Distintas aplicaciones han sido implementadas usando SQLfi como sistema manejador de bases de datos sin seguir una metodología que soporte requisitos difusos (Goncalves y Tineo, 2008). No obstante, no hay propuestas que permitan traducir e implementar los requisitos difusos con lenguajes difusos a bases de datos. En este trabajo se propone una metodología para el desarrollo de aplicaciones caracterizadas por requisitos difusos, cuyas etapas se muestran en la **Figura 2**. En primer lugar, los requisitos serán especificados en lenguaje natural y para cada requi-

sito se identificarán cuáles son sus términos vagos. En segundo lugar, se realizará un modelo conceptual del dominio de aplicación que involucre los requisitos difusos usando el Perfil de Modelado de Requisitos Difusos introducido por Rodríguez y Goncalves (2009). En tercer lugar, los requisitos difusos modelados serán traducidos usando el lenguaje SQLf.

Figura 2

Representación gráfica de los componentes de un requisito difuso



Análisis de requisitos difusos

Inicialmente, el analista debe determinar si el sistema es factible de ser modelado utilizando la lógica difusa a través de las siete características definidas por Rodríguez y Tineo (2009), las cuales están descritas en la **Tabla 3**.

Posteriormente, los requisitos se expresan en lenguaje natural, donde se observan los términos vagos que se corresponden con las preferencias del usuario. El analista del sistema debe distinguir los elementos gramaticales que indican vaguedad en el requisito para su modelación. Los términos lingüísticos vagos a considerar en esta fase fueron analizados por Rodríguez y Tineo (2009) y se detallan en la **Tabla 4**. Cada término lingüístico se modela como un término difuso.

Modelación de requisitos difusos

Los requisitos difusos son representados gráficamente como clases según el perfil UML para el modelado visual de requisitos difusos (Rodríguez y Goncalves, 2009). Éstos son vinculados a través de una relación de dependencia a otras clases del modelo. Las relaciones de dependencia se representan gráficamente en UML con líneas punteadas que terminan en una flecha abierta como se puede observar en la **Figura 3**, donde la clase Clase2 depende de la clase Clase1.

La representación gráfica de un requisito difuso y sus componentes se muestra en la **Figura 4**. La clase del requisito difuso es estereotipada como "Fuzzy Requirement". En ella aparece el nombre o identificador del requisito, los meta-atributos que corresponden a los términos difusos involucrados y su calibración, los atributos presentes en cada instancia del mismo y la especificación formal en OCL que indica la semántica del requisito. Esta última se representa mediante meta-operaciones y operaciones propias del requisito. Los términos difusos son estereotipados como "Fuzzy Term", tienen un nombre y el tipo al que pertenecen (predicados, comparadores, conectores, modificadores o cuantificadores) se-

Tabla 3
Características de factibilidad para modelar un sistema usando lógica difusa

Característica	Descripción
Intuitividad	Cualidad del sistema de ser intuitivo para los usuarios. En las consultas se observan términos lingüísticos importantes que son muy cercanos al lenguaje natural más que valores, rangos o formulaciones complejas propias de lenguajes de consultas clásicas. Ellas deben ser fácilmente entendibles y manejables por el usuario final.
Flexibilidad	Capacidad del sistema de incluir en las respuestas valores en los bordes que pueden ser de interés para el usuario de acuerdo a su preferencia aunque no cumplan de una manera rígida con el ideal buscado. Es decir, el resultado de las consultas difusas puede incluir respuestas que serían rechazadas por una consulta precisa.
Vaguedad	Engloba la imprecisión o incertidumbre que en general está presente en los sistemas de conocimiento y de razonamiento humano. Se incluye cuando en los sistemas las consultas abarcan proposiciones que pueden ser parcialmente ciertas o parcialmente falsas. La información manejada puede estar caracterizada por la incertidumbre, lo cual incluye datos imprecisos o consultas vagas que reflejan preferencias de los usuarios.
Tolerabilidad	Capacidad de establecer niveles de tolerancia dentro de la imprecisión, estableciendo así rangos aceptables del grado de satisfacción de las respuestas obtenidas por el sistema. Esto es lo que en SQLf se conoce como la calibración, la cual limita por un cierto grado la generación de respuestas superpobladas y/o respuestas indeseadas por su bajo nivel de satisfacción o calidad.
Adaptabilidad	El manejo de las preferencias es parametrizable o adaptable por el usuario. Es decir, cada término lingüístico debe ser susceptible a cambios debido a percepciones de los diferentes usuarios o contextos.
Gradualidad	Es importante cuando las respuestas a los requerimientos de las aplicaciones son discriminadas por grados, de acuerdo con la satisfacción de las condiciones involucradas. Cada elemento o registro en el conjunto de respuestas resultantes de una consulta tiene asociado un valor que expresa su importancia o preferencia para el usuario de acuerdo al requerimiento expresado.
Gerencialidad	Cualidad de un sistema de dar soporte a la toma de decisiones. Aunque las consultas difusas pueden ser útiles para personas en cualquier nivel de la organización o para usuarios comunes, ellas toman mayor aplicabilidad e importancia en el entorno gerencial, donde los términos difusos aparecen con mayor frecuencia y tienen mayor necesidad de ser definidos. El principio de incompatibilidad de (Zadeh, 1975) lo describe así "en la medida que crece la complejidad de un sistema en esa misma medida decrece la capacidad de escribir enunciados precisos sobre su funcionalidad". Hemos notado que la lógica difusa es más aplicable cuando el rol del usuario final se encuentra en un nivel gerencial.

gún la enumeración predefinida "FuzzyType". La calibración es estereotipada "Calibration" y puede ser un valor entero mayor que 1 (estereotipo "Quantitative") o un valor real en el intervalo [0,1] (estereotipo "Qualitative"). Los atributos estereotipados con "Output" indican que serán mostrados como salida del requisito.

Tabla 4
Términos lingüísticos vagos

Términos	Descripción	Modelación
Adjetivos calificativos de grado positivo	Añaden una nota de cualidad; el grado positivo corresponde al adjetivo en su forma original, tales como *bueno, malo, alto, bajo, económico, costos, eficaz, productivo, nuevo y antiguo*	Generalmente son modelados como predicados difusos.
Adjetivos calificativos de grado comparativo	Expresan superioridad o inferioridad, tales como *más joven, menos productivo, más eficaz, mejor, peor, mayor, menor, superior, inferior, exterior* e *interior*.	Se modelan como comparadores difusos.
Adjetivos calificativos de grado superlativo relativo	Indican una superioridad o inferioridad relativa a un grupo de entes similares. Se forman con los adjetivos comparativos añadiendo un artículo determinado, así como *el más productivo, el menos costoso*. También hay algunos puros como: *óptimo, pésimo, máximo, mínimo, supremo, sumo, ínfimo, extremo* e *íntimo*.	La modelación de estos términos requiere de un cuantificador o un comparador difuso.
Adjetivos calificativos de grado superlativo absoluto	Denotan la superioridad o inferioridad ideal, se caracterizan por las terminaciones *ísimo* y *érrimo*, tales como *buenísimo, malísimo, antiquísimo, altísimo, bajísimo, riquísimo,* y *paupérrimo*. Éstos tienen equivalencia con el uso del adverbio *muy* seguido de un adjetivo calificativo positivo.	Puede modelarse como un predicado difuso posiblemente afectado por un modificador difuso.
Adjetivos determinativos indefinidos cuantitativos	Añade una nota vaga sobre la cantidad, tales como *algunos, bastantes, cuantiosos, demasiados, muchos, múltiples, ningunos, numerosos, pocos, tantos, todos, unos* y *varios*.	Se modelan como cuantificadores difusos.
Adverbios variables	Son palabras que modifican al verbo a un adjetivo u otro adverbio, particularmente nos interesan adverbios como *bien, mal, grandemente, pequeñamente, altamente, bajamente, externamente, internamente, óptimamente, pésimamente, máximamente, mínimamente, supremamente, sumamente, ínfimamente* y *extremamente*. Otros ejemplos de este grupo son las palabras *cerca* y *lejos*.	Se modelan como modificadores o comparadores difusos
Adverbios invariables de cantidad	Son adverbios que no tienen grado, nos interesan palabras como *bastante, harto, medio, muy, poco, mucho, tanto, tan, casi* y *algo*.	Éstos se modelan como modificadores difusos.

OCL con lógica difusa. Para que los términos difusos puedan ser incluidos en las expresiones OCL, se extendió este lenguaje para que permita condiciones en lógica difusa. Dado que OCL es un lenguaje de especificación, no afecta el estado de los modelos del sistema, por lo que su semántica puede ser extendida para que incluya expresiones que devuelven un valor en el intervalo [0,1], y no sólo los valores clásicos “true” y “false”.

Figura 3

Representación gráfica de los componentes de un requisito difuso

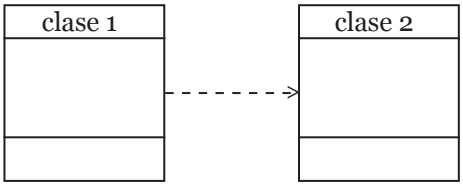


Figura 4

Representación gráfica de los componentes de un requisito difuso

<<Fuzzy Requirement>> Nombre
Meta-atributos
Atributos
Meta-operaciones
Operaciones

Las nuevas condiciones válidas que forman parte de las expresiones lógicas en OCL son de la forma "a is fp", "a is fm fp", "a_1 fc a_2", y "a_1 fc K" donde a, a_1 y a_2 son atributos válidos que pertenecen a las clases de las que depende el requisito, K es una constante, fp es un predicado difuso, fm es un modificador difuso y fc un comparador difuso. Note que fp, fm y fc son declarados según su tipo dentro de la caja del "Fuzzy Requirement" y aparecerán como meta-atributos. En cuanto a los conectores difusos, éstos aparecen en el mismo lugar donde se utilizan los conectores clásicos como and y or, es decir, e_1 fc e_2, siendo e_1 y e_2 dos expresiones válidas en lógica difusa y fc un conector difuso declarado. En caso que los conectores clásicos tengan como operandos expresiones en lógica difusa, su semántica pasa a ser difusa, es decir el mínimo para el and y el máximo para el or (Zadeh, 1977). Por último, los cuantificadores difusos se corresponden a operaciones sobre colecciones, cuya sintaxis es similar a la de los cuantificadores clásicos forall y exists de OCL, es decir, collection->fq(v | logic-expression-with-v) donde fq es un cuantificador declarado en el requisito difuso y la expresión interna en lógica clásica o difusa, se aplica a cada uno de los componentes de la colección, representados por la variable v.

Especificación en OCL de los requisitos difusos. En la **Figura 5**, se muestra un requisito difuso genérico con todos y cada uno de sus componentes, los cuales aparecerán en la semántica del mismo. En primer lugar aparece el identificador del requisito "FRId" estereotipado como "Fuzzy Requirement". A continuación, los términos difusos Ft_1... Ft_k estereotipados como "Fuzzy Term" junto con la calibración (estereotipo "Calibration") la cual puede ser cualitativa (estereotipo "Qualitative"), identificada como L, y/o cuantitativa (estereotipo "Quantitative"), identificada como N. Luego se observan los M atributos a_1.. a_M cuyos tipos son respectivamente $Type_1$..$Type_M$, los cuales aparecerán en la respuesta al requisito. Seguidamente, las funciones OutputRequirement y CalibrateOutput son dos meta-operaciones que

permiten definir la semántica del requisito difuso; éstas serán explicadas más adelante. Finalmente, la operación SatisfationDegree permite obtener para cada instancia del requisito, el grado de satisfacción de dicha instancia en el intervalo [0,1]. La semántica de un requisito difuso es definida a través de una expresión en el lenguaje OCL cuyo resultado es una colección de instancias que lo satisfacen. La expresión OCL que modela la semántica de un requisito será especificada a través de la meta-operación OutputRequirement como:

context FRId::OutputRequirement() :
Set (TupleType(a_1:$Type_1$,..,a_M:$Type_M$))

body:

TypeName.allInstances()->
select(v | FC(v))->collect(c |
Tuple{a_1=Exp_1,..,a_M=Exp_M})

Figura 5

Componentes de un requisito difuso genérico

FRId <<Fuzzy Requirement>>
Ft_1 <<Fuzzy Term>> Ft_K<< Fuzzy Term>> *L* <<Calibration>> <<Qualitative>> *N* <<Calibration>> <<Qualitative>>
a_1:*Type*$_1$ ... a_M:*Type*$_M$
OutputRequirement():Set(TupleType(...)) *CalibrateOutput*(): Set(TupleType(...,*degree*))
SatisfationDegree():float

Donde, FRId corresponde al identificador del requisito difuso; a_1:$Type_1$,..,a_M:$Type_M$ son los atributos indicados como respuesta al requisito, siendo $Type_i$ el tipo del atributo i tal como aparece en la clase a la que pertenece; TypeName es la clase principal de la cual depende el requisito y corresponde al contexto del mismo; FC(v) es la condición difusa que debe cumplir el requisito; Exp_i es el valor que tomará el atributo i en la respuesta.

La interpretación de esta expresión es "seleccionar de todas las instancias de TypeName, aquellas que cumplen la condición difusa FC y la respuesta al requisito es una colección de tuplas (instancias de FRId) cuyos atributos son a_1,..,a_M".

A través de la operación SatisfationDegree, se puede obtener el grado de pertenencia de cada instancia al requisito FRId, es decir, el grado de membresía (μ) al conjunto difuso representado por la condición FC. Esta operación tiene la siguiente declaración:

context FRId::SatisfationDegree() : float

post: result = μ(self)

Entendiéndose como μ(self) la función que calcula dicho grado para la instancia actual. El cálculo de esta función es un proceso recursivo laborioso, por lo que no se incluye en esta especificación. El lector interesado puede encontrarlo en (Bosc y Pivert, 1995; Gonzalez et al, 2009).

Si se especifica una calibración (cualitativa L y/o cuantitativa N) la respuesta al requisito debe

ser limitada a las instancias que satisfacen dicha calibración. En el caso de la calibración cualitativa (L) el grado de satisfacción debe ser mayor o igual que L (self.SatisfationDegree() ≥ L), mientras que en caso de la calibración cuantitativa (N), el número total de instancias del requisito debe ser como máximo N (FRId.allInstances()->size() ≤ N). Por esta razón se define la meta-operación CalibrateOutput() para la respuesta calibrada.

context FRId::CalibrateOutput():
Set(TupleType(a_1:$Type_1$,..,a_M:
$Type_M$, degree:float))
body:
let s : Integer = self.allInstance() ->size in
self.allInstances()->collect(c |
Tuple(a_1=c.a_1 ,.., a_M =c. a_M,
degree = c.SatisfationDegree())) ->
select(t | t.degree ≥ L)->
sortedBy(degree)->subOrderedSet
(if (s>N) then s-N else 1 endif, s)

Nótese que en la respuesta calibrada se toman los M atributos de la instancia, más un atributo adicional (degree) correspondiente al grado de satisfacción de la instancia al requisito. El select ejecuta la calibración cualitativa, es decir, se consideran sólo aquellas instancias con grado mayor que L. La nueva colección obtenida se ordena (usando sortedBy) por el grado de satisfacción para poder aplicar la calibración cuantitativa escogiendo las N mejores instancias (a través de subOrderedSet). Dado que la calibración es opcional dentro del "Fuzzy Requirement", se asume que L tendrá por defecto el valor 0 y N valor infinito cuando dichas calibraciones no sean estereotipadas.

Implementación de requisitos difusos

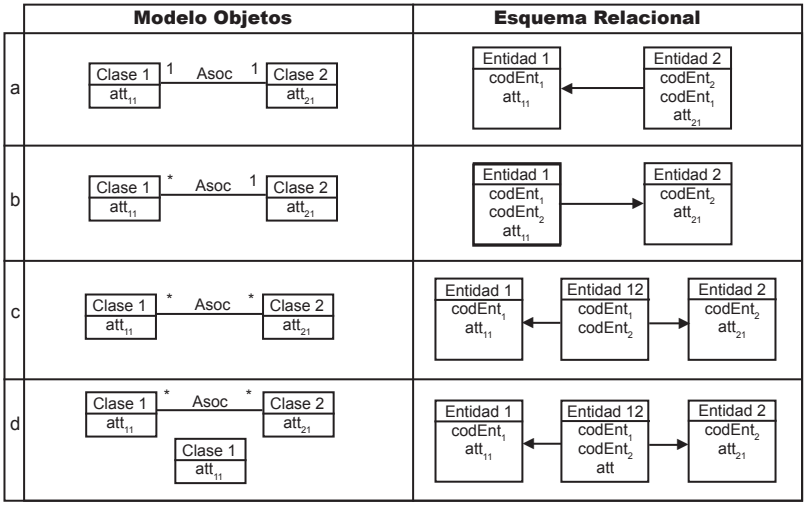
Antes de implementar los requisitos difusos es necesario obtener el esquema relacional correspondiente al modelo de objetos, ya que cada uno de estos requisitos corresponde a una consulta sobre la base de datos. En particular, interesan como se representan las asociaciones entre objetos. Si estas asociaciones no son del tipo "associationClass", sólo poseen como atributos las claves de los objetos que conectan. Dependiendo de la multiplicidad, podría estar representada por una clave foránea que aparece en alguno de estos objetos. En la **Figura 6**, se muestran los diferentes tipos de asociaciones que pueden participar en los requisitos difusos y su esquema relacional correspondiente.

Note que cuando se referencia una asociación dentro de las expresiones OCL, se puede acceder un valor o una colección (expresión de la forma var.Asoc con var del tipo Clase 1 o Clase 2, en los casos a y b). Es por ello que en su esquema relacional equivalente desaparece el nombre de la asociación y se incorporan las claves que permiten construir dicha asociación.

Definición de Términos Difusos en SQLf. Para implementar los requisitos difusos se usará el lenguaje de consultas difusas SQLf. Cada término difuso declarado dentro de una caja "Fuzzy Requirement" será definido usando el lenguaje de definición de datos de SQLf, denominado SQLf-DDL, cuyos comandos se muestran en la **Figura 7**.

Figura 6

Esquema relacional de las diferentes asociaciones presentes en un modelo de objetos



Traducción de la semántica a SQLf. La semántica de los requisitos difusos se especificó en el lenguaje OCL a través de la siguiente expresión:

context FRId::OutputRequirement() :
Set (TupleType(a_1:$Type_1$,..,a_M:$Type_M$))

body:

TypeName.allInstances()->select(v |
FC(v))->collect(c |
Tuple{ a_1=Exp_1,..,a_M=Exp_M })

Ésta será traducida a instrucciones en SQLf por medio de los siguientes pasos:

1. Los atributos dentro de la tupla de la operación collect son incluidos como atributos dentro de la cláusula SELECT, es decir, "SELECT a_1,... ,a_M". Si no se especifica la operación collect es porque se quiere que aparezcan todos los atributos de TypeName, por lo que se escribe el símbolo "*".
2. En la cláusula FROM se coloca la tabla (TableName) correspondiente al clasificador TypeName y se usa como alias la variable auxiliar c. La cláusula FROM entonces queda "FROM TableName AS c".
3. La condición difusa FC(v) es expresada en la cláusula WHERE. Para ello es necesario normalizar dicha condición para que contenga únicamente operadores válidos de SQLf. Asumiremos por simplicidad, que dichas expre-

siones sólo contienen los operadores comunes de la lógica clásica (and, or, not, implies, forall, exists), además de las extensiones propuestas con la lógica difusa en este trabajo.

Para simplificar el proceso de traducción se escribirá la condición difusa FC(v) en términos de átomos y fórmulas. Note que la condición difusa FC(v) es de la forma $E(c_1.a_1,..., c_n.a_n)$ donde $a_1,...,a_n$ son atributos y c_i es el camino para llegar a a_i desde v, siendo E una expresión válida de OCL que usa tales atributos. Si a_i es un atributo de v entonces c_i=v, en caso contrario, el camino comienza con una asociación que parte de v, c_i =asoc... a_i.

Un átomo es una expresión de la forma $E(c_1.a_1,..., c_n.a_n)\ \Omega\ E'(c_1.a_1,..., c_n.a_n)$, $E(c_1.a_1,..., c_n.a_n)\ \Omega\ k$, $k\ \Omega\ E(c_1.a_1,..., c_n.a_n)$, donde cada c_j representa el camino para llegar al atributo a_i; E y E' son expresiones aritméticas válidas que usan los atributos $a_1..a_n$; k es una constante y $\Omega \in \{=, \neq, <, \leq, >, \geq\}$. Estas expresiones deben ser sintáctica y semánticamente correctas. Para incluir los términos difusos, un átomo también puede ser de la forma $c_i.a_i$ is fp, $c_i.a_i$ is fm fp, $c_i.a_i$ fc $c_j.a_j$ donde fp es un predicado difuso, fm es un modificador difuso y fc es un comparador difuso.

Figura 7
Comandos SQLf para definir términos difusos

CREATE FUZZY PREDICATE <*name*> ON <*domain*> AS <*fuzzy set*>;
<*name*>: nombre del término.
<*domain*>: tipo de datos.
<*fuzzy set*> puede ser por:
- Por Extensión: {<grado$_1$>/<valor$_1$>,...,<grado$_n$>/<valor$_n$>}
- Una expresión aritmética que usa "x"
- Trapezoidal: (<val_1>,<val_2>,<val_3>,<val_4>)

CREATE FUZZY MODIFIER <*name*> AS <*transformation*>;
<*transformation*>: representa la función de membresía (μp) de un predicado *P*, transformada a $\mu_{mod\ P}$ por medio de:
- *TRANSLATION* <α>: *trasladar en un grado* α
- *POWER* <*n*>: *potencia n* del grado de satisfacción de cada elemento
- <θ> POWER <*n*>: para la potencia *n* del operador θ.

CREATE FUZZY CONNECTIVE <*name*> AS <*expression*>.

CREATE COMPARATOR <*name*> ON <*domain*> AS <*pair*> IN <*fuzzy set*>;
<*pair*>: puede ser el par ordenado (*x,v*), *la diferencia* (*x-y*) *o la división* (*x/y*)

CREATE <*nature*> FUZZY QUANTIFIER <*name*> AS <$value_1$,<$value_2$,<$value_3$,<$value_4$>;
<*nature*>: ABSOLUTE *o PROPORTIONAL*
<$value_1$,<$value_2$,<$value_3$,<$value_4$> definición trapezoidal para un cuantificador

Las fórmulas se construyen en base a los átomos. Un átomo es una fórmula válida. Si F es una fórmula válida, entonces (F) y not F (negación) son fórmulas válidas. Si F_1 y F_2 son fórmulas válidas, entonces F_1 and F_2 (conjunción), F_1 or F_2 (disyunción) y F_1 implies F_2 (implicación) son fórmulas válidas. Si F_1 y F_2 son fórmulas válidas, y fc es un conector difuso entonces F_1 fc F_2 es una fórmula válida. Adicionalmente, se tienen fórmulas válidas en el contexto de las colecciones: collection->forall(v | F(v)), collection->exists(v | F(v)), collection->select(v | F(v)) y collection->fq(v | F(v)), son fórmulas válidas si F es una fórmula válida, siendo fq un cuantificador difuso.

El procedimiento de normalización permite eliminar aquellos operadores lógicos que no pueden ser representados en SQLf. Este procedimiento fue introducido para lógica de primer orden por Kawash (2004) y consiste de cuatro pasos: primero, eliminar todas las implicaciones aplicando la Equivalencia de la Implicación, que en lenguaje OCL sería: (F_1 implies F_2) ≡ (not F_1 or F_2); segundo, eliminar todos los cuantificadores universales, aplicando la Equivalencia de la Cuantificación Universal, lo cual no se realizará porque SQLf posee el cuantificador universal; tercero, eliminar la doble negación usando la Equivalencia de la Doble Negación: not not F ≡ F; cuarto, eliminar las negación internas aplicando las leyes De Morgan: not (F_1 and F_2) ≡ (not F_1 or not F_2) y not (F_1 or F_2) ≡ (not F_1 and not F_2). Las negaciones internas aparecen en las fórmulas de la forma not collection->exists(t | not expression(t)). Así, una expresión válida E está normalizada si no contiene ni Doble Negación, ni implicación, ni negaciones internas.

Si FC(v) está normalizada podemos definir una función recursiva Traducción como:

– Traducción(TableName, E(v.a_1,..., v.a_n)) = E(c.a_1,..., c.a_n) si E es una expresión aritmética donde sólo aparecen atributos de v, es decir, atributos que no se acceden a través de asociaciones, c es la variable auxiliar correspondiente a TableName que aparece en la cláusula FROM y a_1,..., a_n son atributos de TableName

– Traducción(TableName, E(v.instanceRole.a_1,..., v. instanceRole.a_n)) = c.$fk_{TableName'}$ IN

(SELECT $pk_{TableName'}$ FROM TableName' AS w WHERE E(w.a_1,..., w.a_n))

si E es una expresión lógica donde aparecen atributos de una asociación entre las tablas TableName y TableName', tal que estos atributos pertenecen a una instancia de TableName' y no a una colección; $fk_{TableName'}$ es la clave foránea de la tabla TableName hacia la tabla TableName', $pk_{TableName'}$ es la clave primaria de la tabla TableName', y a_1,...,a_n son atributos de TableName'; c es la variable auxiliar correspondiente a TableName que aparece en la cláusula FROM.

– Traducción(TableName, v.collectionRole->F(v)) = v.$pk_{TableName}$ IN

(SELECT $fk_{TableName'}$ FROM TableName' GROUP BY $fk_{TableName'}$

HAVING Traducción(TableName', F(v)))
donde TableName' corresponde al tipo de collectionRole, $pk_{TableName}$ es la clave primaria de la tabla TableName; $fk_{TableName'}$ es la clave foránea de la tabla TableName hacia la tabla TableName'.

– Traducción (contexto, (F)) = (Traducción (contexto, F))
– Traducción (contexto, not F) = NOT Traducción (contexto, F)
– Traducción (contexto, F_1 and F_2) = Traducción (contexto, F_1) AND Traducción(contexto, F_2)
– Traducción (contexto, F_1 or F_2) = Traducción (contexto, F_1) OR Traducción (contexto, F_2)
– Traducción(contexto, F_1 fc F_2) = Traducción (contexto, F_1) fc Traducción (contexto, F_2) donde fc es un conector difuso
– Traducción(TableName, exists(v | F(v)) = {FOR SOME | FOR ALL} c IN

  TableName : (Traducción(TableName, F(v)))
  donde TableName es la tabla correspondiente al tipo de la colección sobre la cual se aplica el exists o el forall; c es la variable auxiliar correspondiente a TableName que aparece en la cláusula FROM. Este tipo de condición sólo puede aparecer en el WHERE o en el HAVING.
– Traducción(TableName, select(v | F(v)) = {WHERE|AND} Traducción(TableName, F(v))
  donde se escribe una cláusula WHERE si en la traducción no se ha generado la misma, de lo contrario ya se generó una y se coloca un conector AND para agregar la traducción de la condición F(V).
– Traducción(TableName, fq(v | F(v)) = fq ARE (Traducción(TableName, F(v)))
  donde TableName es la tabla correspondiente al tipo de la collection. Note que el contexto que se pasa a la traducción de la condición del cuantificador difuso fq es TableName.

Caso de estudio

En esta sección se describe la aplicación de nuestro método en el desarrollo de un sistema Web de compra y venta de vehículos de acuerdo a las preferencias del usuario.

Análisis

Un usuario puede buscar y seleccionar vehículos de las distintas ofertas publicadas mediante la combinación de varios criterios sobre los elementos de datos que describen a un vehículo ofertado. Así, se han identificado cuatro requisitos iniciales:

FR1: ¿Cuáles son los vehículos de precio ***bajo***?

FR2: ¿Cuáles son los vehículos ***nuevos***?

FR3: ¿Cuáles son las marcas con la ***mayoría*** de vehículos de color ***claro***?

FR4: ¿Cuáles son las marcas ***preferidas*** de vehículos donde la ***mayoría*** de los vehículos son de precio ***bajo*** y color ***similar*** al gris?

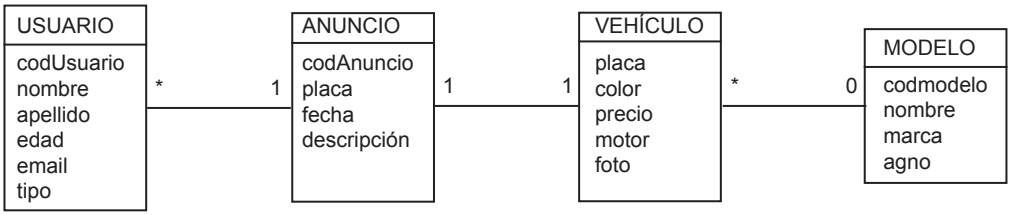
La especificación de los requisitos en lenguaje natural incluye términos difusos resaltados usando *itálicas* y *negritas*. Además, a cada requisito le es asignado un código ***FRi***, donde ***i*** es el número del requisito. Los requisitos de usuario especificados evidencian que el sistema posee las sietes características a ser modeladas usando lógica difusa, como se evidencia en la **Tabla 5**.

Adicionalmente, en el análisis se obtiene el modelo de dominio con sus objetos y sus primeras asociaciones. Un posible modelo del dominio, representado con un diagrama de clases UML, para el sistema de compra y venta de vehículos se presenta en la **Figura 8**.

Tabla 5
Análisis de características difusas

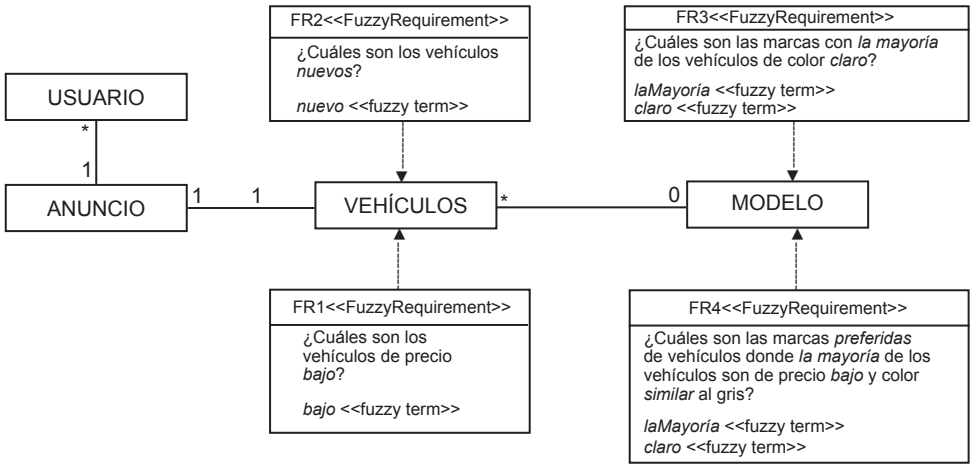
Característica	Descripción
Intuitividad	Cualidad del sistema de ser intuitivo para los usuarios. En las consultas se observan términos lingüísticos importantes que son muy cercanos al lenguaje natural más que valores, rangos o formulaciones complejas propias de lenguajes de consultas clásicas. Ellas deben ser fácilmente entendibles y manejables por el usuario final.
Flexibilidad	Capacidad del sistema de incluir en las respuestas valores en los bordes que pueden ser de interés para el usuario de acuerdo a su preferencia aunque no cumplan de una manera rígida con el ideal buscado. Es decir, el resultado de las consultas difusas puede incluir respuestas que serían rechazadas por una consulta precisa.
Vaguedad	Engloba la imprecisión o incertidumbre que en general está presente en los sistemas de conocimiento y de razonamiento humano. Se incluye cuando en los sistemas las consultas abarcan proposiciones que pueden ser parcialmente ciertas o parcialmente falsas. La información manejada puede estar caracterizada por la incertidumbre, lo cual incluye datos imprecisos o consultas vagas que reflejan preferencias de los usuarios.
Tolerabilidad	Capacidad de establecer niveles de tolerancia dentro de la imprecisión, estableciendo así rangos aceptables del grado de satisfacción de las respuestas obtenidas por el sistema. Esto es lo que en SQLf se conoce como la calibración, la cual limita por un cierto grado la generación de respuestas superpobladas y/o respuestas indeseadas por su bajo nivel de satisfacción o calidad.
Adaptabilidad	El manejo de las preferencias es parametrizable o adaptable por el usuario. Es decir, cada término lingüístico debe ser susceptible a cambios debido a percepciones de los diferentes usuarios o contextos.
Gradualidad	Es importante cuando las respuestas a los requerimientos de las aplicaciones son discriminadas por grados, de acuerdo con la satisfacción de las condiciones involucradas. Cada elemento o registro en el conjunto de respuestas resultantes de una consulta tiene asociado un valor que expresa su importancia o preferencia para el usuario de acuerdo al requerimiento expresado.
Gerencialidad	Cualidad de un sistema de dar soporte a la toma de decisiones. Aunque las consultas difusas pueden ser útiles para personas en cualquier nivel de la organización o para usuarios comunes, ellas toman mayor aplicabilidad e importancia en el entorno gerencial, donde los términos difusos aparecen con mayor frecuencia y tienen mayor necesidad de ser definidos. El principio de incompatibilidad de (Zadeh, 1975) lo describe así "en la medida que crece la complejidad de un sistema en esa misma medida decrece la capacidad de escribir enunciados precisos sobre su funcionalidad". Hemos notado que la lógica difusa es más aplicable cuando el rol del usuario final se encuentra en un nivel gerencial.

Figura 8
Modelo de dominio para el sistema de compra y venta de vehículos



Al modelo de dominio se le incluye los requisitos difusos como se muestra en la **Figura 9**. Cada requisito difuso descrito en lenguaje natural es modelado como una clase estereotipada "FuzzyRequirement", está asociado a las clases de las que depende. Por ejemplo, el requisito ***FR1*** depende únicamente de la clase VEHICULOS. Cada término difuso se identifica con un estereotipo "Fuzzy Term" y se coloca dentro de la clase del requisito.

Figura 9
Modelo de dominio usando el Perfil UML para el modelado de requisitos difusos

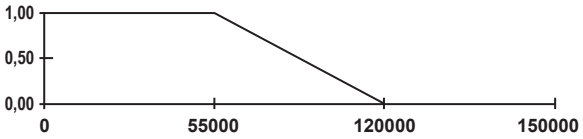


Diseño

El siguiente paso es definir cada término difuso resaltado durante la etapa de análisis. Esos términos serán usados en la especificación OCL de cada uno de los requisitos. Los términos ***bajo***, ***nuevos***, ***claro*** y ***preferidas*** son adjetivos calificativos de grado positivo, por lo que su modelación se haría como predicados difusos. Debido a que el universo del conjunto del término ***bajo*** es infinito, se empleará trapezoides para su modelación, lo cual se muestra en la **Figura 10**. Similarmente, se puede emplear un trapecio para modelar el término ***nuevos*** como se observa en la **Figura 11**. Dado que el universo del conjunto de los términos ***preferidas*** y ***claro*** es discreto y finito, se modelarán por extensión. Así, el conjunto difuso para el término ***preferidas*** es {1/fiat,1/chevrolet,1/toyota,1/ford,0.2/seat, 0.5/wolswagen} y para el término ***claro*** {0.0/negro, 0.2/verde, 0.4/rojo, 0.6/amarillo, 0.8/celeste, 1.0/blanco}.

Figura 10

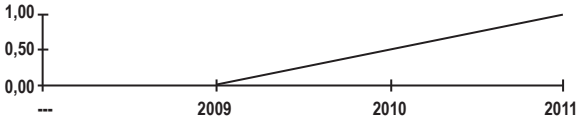
Función de membresía para el conjunto difuso del término *bajo*



El término ***similar*** es un adjetivo calificativo de grado comparativo, lo que indica un comparador difuso. En este caso, se decide modelar el término ***similar*** por extensión, enumerando cada par ordenado con su respectivo grado: {1/(negro/gris), 0.5/(negro,azul), 0/(negro,amarillo), 0/(negro/rojo), 0/(negro,blanco), 0.5/(gris,azul), 0/(gris,amarillo), 0/(gris,rojo), 0/(gris,blanco), 0/(azul,amarillo), 0/(azul/rojo), 0/(azul,blanco), 0/(amarillo, rojo), 0.5/(amarillo,blanco), 0/(rojo,blanco)}.

Figura 11

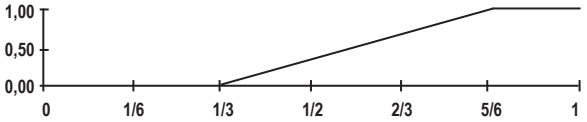
Función de membresía para el conjunto difuso del término *nuevos*



La expresión lingüística ***la mayoría*** es un adjetivo determinativo indefinido cuantitativo, por lo que corresponde a un cuantificador difuso, además, es proporcional pues describe una cantidad relativa a un todo. Su representación es dada en la **Figura 12**.

Figura 12

Función de membresía del conjunto difuso del término *la mayoría*

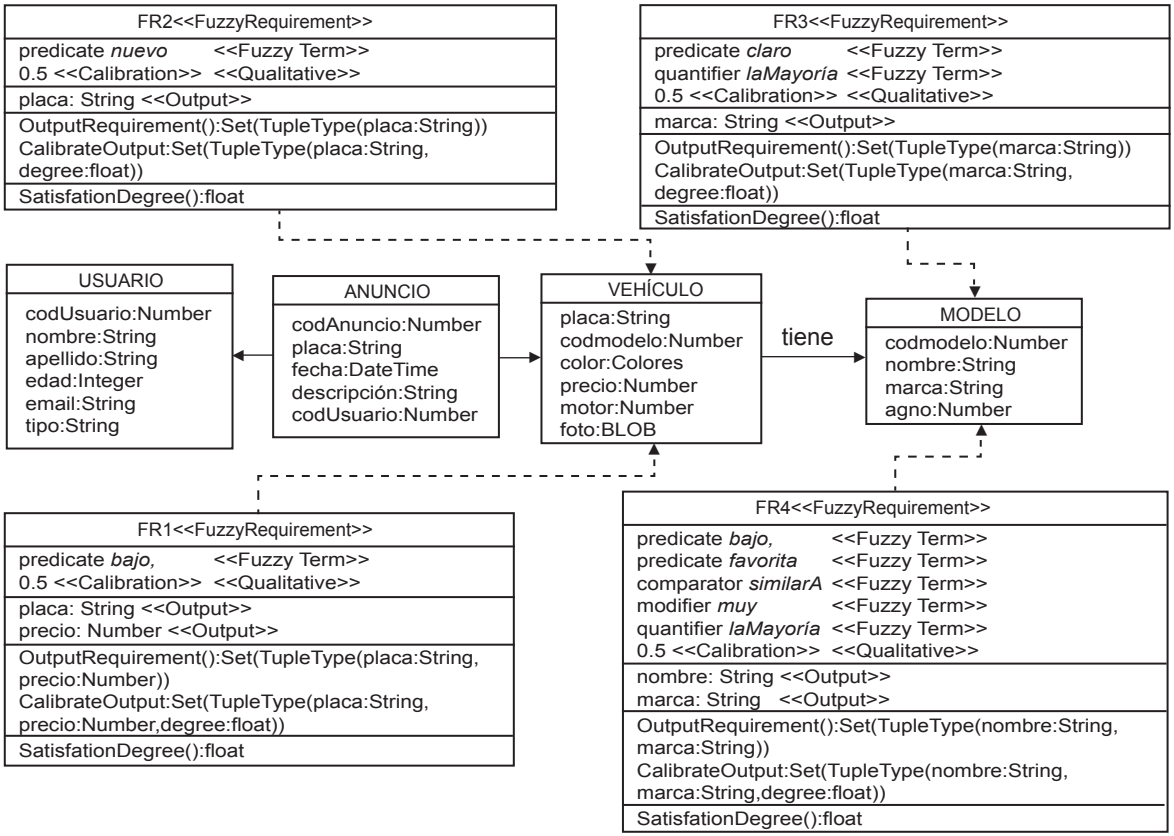


Una vez definidos los términos difusos, se deciden los tipos de datos. Se especifica el tipo de cada término difuso, el valor y tipo de calibración,

el conjunto de atributos de salida estereotipados como "Output" y la semántica del requisito a través de las meta-operaciones OutputRequirement y CalibrateOutput.

Todos los requisitos difusos tienen como operación SatisfationDegree que devuelve el grado de membresía a la consulta de cada tupla de salida. Entonces se obtiene un diagrama de clases como el de la **Figura 13**.

Figura 13
Modelo objeto del nivel de diseño



En este diagrama se observa sólo el encabezado de las meta-operaciones, en el cual aparecen los atributos que formarán parte de la respuesta (OutputRequirement), la cual se transforma en un conjunto difuso al agregarle el grado de membresía (CalibrateOutput). Este grado es accesible por medio de la operación SatisfationDegree. Como ejemplo, supóngase que se tiene el conjunto de instancias de la clase VEHÍCULO que se muestra en la **Tabla 6**. Si se evalúa la semántica del requisito difuso ***FR1***, la salida de esta operación serían las filas de la **Tabla 7**, que indican el grado de satisfacción de cada instancia de vehículo al requisito.

Tabla 6

Instancias de vehículos

Placa	Color	Precio	Motor	Foto
ALK679	BLANCO	55000	KHEQ8023NC	F1
FDR546	NEGRO	70000	ALWIOQ	F2
HYT156	ROJO	40000	HJU679H778	F3
JQW678	AMARILLO	105000	ÑFA902LA8	F4
MNB546	CELESTE	85000	LH9879KY7	F5
RDI987	VERDE	98000	KJÑIY7JUIO	F6

Tabla 7

Salida de la operación SatisfactionDegree() para el requisito *FR1*

Placa	Salida de la operación SatisfactionDegree()
HYT156	1
ALK679	0.85
FDR546	0.7
MNB546	0.5
RDI987	0
JQW678	0

Por último, se detalla la semántica de cada requisito difuso haciendo la especificación de su descripción en lenguaje natural al lenguaje OCL. En la **Tabla 8** se muestra el resultado de especificar la semántica de los requisitos difusos de la **Figura 13**.

Tabla 8

Especificación de cada requisito difuso en OCL

Requerimiento	Semántica en OCL
FR1	**context** FR1::OutputRequirement():**Set**(**TupleType**(placa: String,precio:Number)) **body**: VEHICULO.**allInstances**()->**select**(v | v.precio is *bajo*).**collect**(c | **Tuple**{placa=c.placa, precio=c.precio})
FR2	**context** FR2::OutputRequirement():**Set**(**TupleType**(placa:String)) **body**: VEHICULO.**allInstances**()->**select**(v | v.tiene.agno is *nuevo*).**collect**(c | **Tuple**{placa = c.placa})
FR3	**context** FR3::OutputRequirement():**Set**(**TupleType**(marca: String)) **body**: MODELO.**allInstances**()->**select**(m | m.tiene->*laMayoria*(v | v.color is *claro*)).collect(c | Tuple{marca = m.marca})
FR4	**context** FR4::OutputRequirement():**Set**(**TupleType**(nombre: String,marca:String)) **body**: MODELO.**allInstances**->**select**(m | m.marca is *favorita* **and** m.tiene->*laMayoria*(v | v.precio is *muy bajo* **and** v.color *similarA* gris)).**collect**(c| Tuple{ nombre = c.nombre,marca = c.marca })

Implementación

En esta etapa, los términos difusos y requisitos en OCL, identificados durante el análisis y definidos en el diseño, son traducidos a instrucciones en el lenguaje de consultas difusas SQLf-DDL como se muestra en la **Figura 14**.

Figura 14
Definición de términos difusos en SQLf

```
CREATE FUZZY PREDICATE bajo ON NUMBER AS (0, 0, 55000, 120000);
CREATE FUZZY PREDICATE nuevos ON DATE AS (2009, 2011, INFINITE, INFINITE);
CREATE FUZZY PREDICATE preferidas ON marca AS
                { 1/fiat, 1/chevrolet, 1/toyota, 1/ford,0.2/seat, 0.5/wolswagen);
CREATE FUZZY PREDICATE claro ON COLOR AS
                { 0.0/negro, 0.2/verde, 0.4/rojo, 0.6/amarillo, 0.8/celeste,1.0/blanco};
CREATE FUZZY COMPARATOR similar ON COLOR AS
                { 1/ (negro,gris), 0.5/ (negro,azul), 0/ negro,amarillo), 0/ (negro,rojo),
                  0/ (negro,blanco), 0.5/ (gris,azul), 0/ gris,amarillo), 0/ (gris,rojo),
                  0/ (gris,blanco), 0/ (azul,amarillo), 0/ azul,rojo), 0/ (azul,blanco),
                  0/ (amarillo,rojo), 0.5/ (amarillo,blanco), 0/ (rojo,blanco)};
CREATE FUZZY QUANTIFIER lamayoria AS (0.3333, 0.8333, 1.0, INFINITE);
```

Posteriormente, se traduce cada requisito difuso especificado en OCL a su correspondiente instrucción en SQLf. El resultado de la traducción se observa en la **Tabla 9**.

Tabla 9
Traducción de cada requisito difuso a SQLf

Requerimiento	Traducción en SQLf
FR1	**SELECT** placa,precio **FROM** VEHICULO **AS** c **WHERE** c.precio **is** *bajo*;
FR2	**SELECT** c.placa **FROM** VEHICULO **AS** c **WHERE** c.codmodelo **IN** (**SELECT** m.codmodelo **FROM** MODELO **AS** m **WHERE** m.agno **is** *nuevo*);
FR3	**SELECT** m.marca **FROM** MODELO **AS** m **WHERE** *laMayoria* codmodelo **ARE** (**SELECT** v.codmodelo **FROM** VEHICULO **AS** v **WHERE** v.color **is** *claro*);
FR4	**SELECT** m.marca **FROM** MODELO **AS** m **WHERE** m.marca **is** *favorita* **AND** m.codmodelo **IN** (**SELECT** codmodelo **FROM** VEHICULO **AS** v **GROUP BY** codmodelo **HAVING** *laMayoria* **ARE** v.precio **is** *muy bajo* **AND** v.color *similarA* gris);

El primer requisito ***FR1*** no incluye cuantificadores, por lo tanto la condición se coloca directamente en la cláusula WHERE, como lo especifica el primer caso de la función Traducción. Similar al requisito ***FR1***, la condición del requisito ***FR2*** no contiene cuantificadores. Sin embargo, en la condición aparece el acceso a un atributo de una asociación "v->tiene.agno", por lo cual se cae en el segundo caso de la función Traducción. Nótese que por medio de la clave foránea codmodelo de la tabla VEHÍCULO se accede al atributo "agno" de la tabla MODELO (ver **Figura 13**).

En el requisito ***FR3*** nuevamente aparece la asociación "tiene" pero para obtener una colección de vehículos. Por lo que se cae en el tercer caso de la función Traducción. La condición difusa posee un cuantificador ***lamayoria*** sobre la colección de vehículos, por lo que se utiliza el caso de la función Traducción para cuantificadores difusos. Por lo tanto, el cuantificador es aplicado sobre una sub-consulta que obtiene los vehículos *claros*. Finalmente, la condición del requisito ***FR4*** contie-

ne un conector (and) con dos condiciones difusas. Además, en la segunda condición un cuantificador que obtiene los vehículos de precio ***bajo*** y de color ***similar*** a gris. A través de la función Traducción se resuelve el conector de dos condiciones, una que cae en el primer caso de Traducción y otra por una asociación con el rol de colección, la cual se accede por el cuantificador difuso.

Conclusiones

En este trabajo, se ha propuesto un método que soporta la especificación, modelado e implementación de requisitos difusos en el desarrollo de software.

El análisis produce una lista de requisitos difusos en lenguaje natural. Cada uno es modelado visualmente y especificado formalmente en OCL extendido con lógica difusa. Cada especificación es traducida a SQLf para su implementación.

El mecanismo de traducción propuesto constituye la principal contribución de este trabajo. La sencillez y factibilidad de uso de este método se mostró mediante un caso de estudio real, un sistema flexible de compra y venta de vehículos que involucra requisitos con términos difusos. Así, se abre un abanico de posibilidades para una nueva generación de sistemas de información que requieran el soporte de requisitos difusos.

El método propuesto hace factible el desarrollo de una herramienta para especificar formal y visualmente los requisitos difusos, y generar código SQLf. Así, la construcción de aplicaciones con requisitos difusos puede ser automatizada eliminando o minimizando el problema de ambigüedad. Como trabajos futuros, se propone automatizar tanto la traducción del lenguaje natural al lenguaje de especificación formal OCL, como la traducción de OCL a SQLf. En el primer caso se puede emplear un buscador semántico que realice búsquedas basadas en el significado de un conjunto de palabras. Adicionalmente, la creación de un "framework" que facilite la construcción de sistemas orientados a datos que usen consultas difusas..

Bibliografía

Bordogna, G. y Psaila, G. (2008). *Customizable Flexible Querying for Classical Relation Databases*. Pensilvania, USA: Galindo, J (Ed.) Handbook of Research on Fuzzy Information Processing in Databases. Vol 1, (1), 191-217.

Bosc, P. y Pivert, O. (1995). SQLf: A Relational Database Language for Fuzzy Querying. *IEEE Transactions on Fuzzy Systems*. Vol 3, (1), 1-17.

Galindo, J., Urrutia, A. y Piattini, M. (2006). *Fuzzy Database Modeling, Design and Implementation*. Idea Group Publishing.

Goncalves, M., Rodríguez, R. y Tineo, L. (2009). Incorporando consultas difusas en el desarrollo de software. *Revista Avances en Sistemas e Informática*, Vol. 6, (3), 87-101.

Goncalves, M. y Tineo, L. (2001a) SQLf Flexible Querying Language Extension by means of the norm SQL2, *The 10th IEEE International Conference on Fuzzy Systems*, Vol. 1, 473-476.

Goncalves, M. y Tineo, L. (2001b) SQLf3: An extension of SQLf with SQL3 features, *The 10th IEEE International Conference on Fuzzy Systems*, Vol. 1, 477-480.

Goncalves, M. y Tineo, L. (2008). SQLfi y Sus Aplicaciones. *Revista Avances en Sistemas e Informática*, Vol. 5, (2), 33-40.

González, C. (2008). *Una propuesta para la integración y estandarización de SQLf y FSQL*. Trabajo Especial de Grado de Maestría, Universidad Simón Bolívar, Caracas, Venezuela.

González, C., Goncalves, M. y Tineo, L. (2009). A New Upgrade to SQLf: Towards an Standard in Fuzzy Databases. *20th International Workshop on Database and Expert Systems Application Proceeding*, pp. 442-446.

Kawash, J. (2004) Complex quantification in Structured Query Language (SQL): a tutorial using relational calculus. *Journal of Computer in Mathematics and Science Teaching*.

Object Management Group (2006). *Meta Object Facility (MOF) Core Specification OMG*. Recuperado el 9 de octubre de 2009, del sitio Web del Object Management Group: http://www.omg.org/spec/MOF/2.0/

Rodríguez, R. y Goncalves, M. (2009). Perfil UML para el modelado visual de requisitos difusos. *Enl@ce: Revista Venezolana de Información, Tecnología y Conocimiento*, Vol. 6, (3), 29-46.

Rodríguez, R. y Tineo, L. (2009). Elementos Gramaticales y Características que Determinan Aplicaciones con Requerimientos Difusos. *Revista Tekhne*, Vol. 12, 50-64. Universidad Católica Andrés Bello. Caracas, Venezuela

Tineo, L. (1998). *Interrogaciones Flexibles en Bases de Datos Relacionales*. Trabajo de Ascenso, Universidad Simón Bolívar, Caracas, Venezuela. No publicado.

Zadeh, L. (1965). Fuzzy Sets. *Information and Control*. Vol. 8, (3), 338-353.

Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Information Science*. Vol. 8, (1), 199-249.

Zadeh, L. (1977). PRUF – A Language for the Representation of Meaning in Natural Languages. *IJCAI*. 918.