

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática), organización que edita también la revista REICIS (Revista Española de Innovación, Calidad e Ingeniería del Software).

<<http://www.ati.es/novatica/>>
<<http://www.ati.es/reicis/>>

ATI es miembro fundador de CEPIS (Council of European Professional Informatics Societies) y es representante de España en IFIP (International Federation for Information Processing); tiene un acuerdo de colaboración con ACM (Association for Computing Machinery), así como acuerdos de vinculación o colaboración con Adaspan, AIZ, ASTIC, RITSI e Hispalinux, junto a la que participa en ProInnova.

Consejo Editorial
Guillermo Alsina González, Rafael Fernández Calvo (presidente del Consejo), Jaime Fernández Martínez, Luis Fernández Sanz, José Antonio Gutiérrez de Mesa, Silvia Leal Martín, Diego López Viles, Francisco Noguera Puig, Joan Antoni Pastor Collado, Andrés Pérez Payeras, Viktu Pons i Colomer, Moisés Robles Gero, Cristina Vigil Díaz, Juan Carlos Vigo López

Coordinación Editorial
Llorenç Pagés Casas <cpages@ati.es>

Composición y autoedición
Jorge Llácer Gil de Banales

Traducciones
Grupo de Lengua e Informática de ATI <<http://www.ati.es/ql/lengua-informatica/>>

Administración
Tomás Brunete, María José Fernández, Enric Camarero, Felicidad López

Secciones Técnicas - Coordinadores

Acceso y recuperación de la Información
José María Gómez Hidalgo (Optenet), <jingomez@yahoo.es>

Manuel J. María López (Universidad de Huelva), <manuel.maria@diehsia.uhu.es>

Administración Pública electrónica
Francisco López Crespo (MAE), <flo@ati.es>

Sebastià Justicia Pérez (Diputación de Barcelona) <sjusticia@ati.es>

Arquitecturas

Enrique F. Torres Moreno (Universidad de Zaragoza), <enrique.torres@unizar.es>

José Filch Cando (Universidad Politécnica de Valencia), <filch@disca.upv.es>

Auditoría SITIC

Marina Tourino Troilito, <marinatourino@marinatourino.com>

Sergio Gómez-Landero Pérez (Endesa), <sergio.gomezlandero@endesa.es>

Derecho y tecnologías

Isabel Hernando Collazos (Fac. Derecho de Donostia, UPV), <isabel.hernando@ehu.es>

Elena Davara Fernández de Marcos (Davara & Davara), <edavara@davara.com>

Enseñanza Universitaria de la Informática

Cristóbal Fariña Flores (DSIC-UCM), <cfari@dsic.ucm.es>

J. Angel Velázquez Iturbide (DLSI I, URJC), <angel.velazquez@urjc.es>

Entorno digital personal

Andrés Marín López (Univ. Carlos III), <amarin@it.uc3m.es>

Diego Gachet Páez (Universidad Europea de Madrid), <gachet@uem.es>

Estándares Web

Encarna Quesada Ruiz (Virati), <encarna.quesada@virati.com>

José Carlos del Arco Prieto (TCP Sistemas e Ingeniería), <jcarco@gmail.com>

Gestión del Conocimiento

Juan Baiget Solé (Cogni Semini Ernst & Young), <juan.baiget@ati.es>

Gobierno corporativo de las TI

Manuel Palao García-Suello (ATI), <manuel@palao.com>

Miguel García-Méndez (ITI), <mgarciamendez@ititrends.institute.org>

Informática y Filosofía

José Ángel Olivás Varela (Escuela Superior de Informática, UCLM), <josangel.olivas@uclm.es>

Roberto Feltrero Oreja (UNED), <rfeltrero@gmail.com>

Informática Gráfica

Miguel Chover Selles (Universitat Jaume I de Castellón), <chover@lsi.uji.es>

Roberto Vivó Hernando (Eurographics, sección española), <rvivo@dsic.upv.es>

Ingeniería del Software

Javier Dolado Cosín (DLSI-UPV), <dolado@si.ehu.es>

Daniel Rodríguez García (Universidad de Alcalá), <daniel.rodriguez@uah.es>

Inteligencia Artificial

Vicente Botti Navarro, Vicente Julián Inglada (DSIC-UPV), <vbotti.vinglada@dsic.upv.es>

Interacción Persona-Computador

Pedro M. Latorre Andrés (Universidad de Zaragoza, AIPD), <platorre@unizar.es>

Francisco L. Gutiérrez Vela (Universidad de Granada, AIPD), <fgutierrez@ugr.es>

Lengua e Informática

M. del Carmen Ugarte García (ATI), <cugarte@ati.es>

Lenguajes Informáticos

Oscar Belmonte Fernández (Univ. Jaime I de Castellón), <beltern@lsi.uji.es>

Inmaculada Coma Tatay (Univ. de Valencia), <inmaculada.coma@uv.es>

Lingüística computacional

Xavier Gómez Guinovart (Univ. de Vigo), <xgg@uvigo.es>

Manuel Palomar (Univ. de Alicante), <mpalomar@disi.ua.es>

Mundo estudiantil y jóvenes profesionales

Federico G. Mon Trotti (RITSI), <gnu.fede@gmail.com>

Mikel Salazar Peña (Asoc. de Jóvenes Profesionales, Junta de ATI Madrid), <mikelhxo_uni@yahoo.es>

Profesión Informática

Rafael Fernández Calvo (ATI), <rfcalvo@ati.es>

Miquel Sarries Gñó (ATI), <miquel@sarries.net>

Redes y servicios telemáticos

José Luis Marzo Lázaro (Univ. de Girona), <joseluis.marzo@udg.es>

Juan Carlos López López (UCLM), <juancharlos.lopez@uclm.es>

Robótica

José Cortés Arenas (Sopra Group), <joscorteras@gmail.com>

Juan González Gómez (Universidad CARLOS III), <juan@iearobotics.com>

Seguridad

Javier Arellano Bertolin (Univ. de Deusto), <jarellino@deusto.es>

Javier López Muñoz (ETSI Informática-UMA), <jlm@cc.uma.es>

Sistemas de Tiempo Real

Alejandro Alonso Muñoz, Juan Antonio de la Puente Altaro (DIT-UPM), <faalonso.jpunte@dit.upm.es>

Software Libre

Jesus M. González Barahona (GSYC-URJC), <jgib@gsyc.es>

Israel Herranz Tabernero (Universidad Politécnica de Madrid), <isra@herranz.org>

Tecnología de Objetos

Jesus Garcia Molina (DIS-UM), <jmolina@um.es>

Gustavo Rossi (LPIA-UNLP Argentina), <gustavo@sol.info.unlp.edu.ar>

Tecnologías para la Educación

Juan Manuel Dodero Beardo (UC3M), <ddodero@inf.uc3m.es>

César Pablo Córcoles Briongo (UOC), <ccorcoles@uoc.edu>

Tecnologías y Empresa

Didac López Vilas (Universitat de Girona), <didac.lopez@ati.es>

Alonso Álvarez García (TD), <aag@tid.es>

Tendencias tecnológicas

Gabriel Martí Fuentes (Interbits), <gabi@atinet.es>

Juan Carlos Vigo (ATI) <juancarlosvigo@atinet.es>

TIC y Turismo

Andrés Aguayo Maldonado, Antonio Guevara Plaza (Univ. de Málaga), <faguayo.guevara@lcc.uma.es>

Las opiniones expresadas por los autores son responsabilidad exclusiva de los mismos.

Novática permite la reproducción, sin ánimo de lucro, de todos los artículos, a menos que lo impida la modalidad de © o copyright elegida por el autor, debiéndose en todo caso citar su procedencia y enviar a **Novática** un ejemplar de la publicación.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid

Plaza de España 6, 2ª planta, 28008 Madrid

Tfno. 914029391; fax 913093685 <novatica@ati.es>

Composición, Edición y Redacción ATI Valencia

Av. del Penon de Valencia 23, 46005 Valencia

Tfno. 963740173 <novatica_valencia@ati.es>

Administración y Redacción ATI Cataluña

Calle Avila 50, 3a planta, local 9, 08005 Barcelona

Tfno. 934125235; fax 934127113 <secretgen@ati.es>

Redacción ATI Aragón

Lagasca 9, 3-B, 50006 Zaragoza

Tfno./fax 976235181 <secretara@ati.es>

Redacción ATI Andalucía <secretand@ati.es>

Redacción ATI Galicia <secretgal@ati.es>

Suscripción y Ventas <novatica.subscripciones@atinet.es>

Publicidad Plaza de España 6, 2ª planta, 28008 Madrid

Tfno. 914029391; fax 913093685 <novatica@ati.es>

Imprenta: Dierra S.A. Juan de Austria 6, 08005 Barcelona

Depósito legal: B 15.154-1975 -- ISSN: 0211-2124. CODEN NOVAC

Portada: "Heuristically" - Concha Arias Pérez / © ATI

Diseño: Fernando Agresta / © ATI 2003

editorial

La interfaz de usuario en el punto de mira en resumen > 02

Máquinas e interfaces adaptables y adaptadas para un mundo mejor > 02
Llorenç Pagés Casas

Actividades del sector informático

EXPOEARNING consolida su liderazgo y muestra las nuevas tendencias en aprendizaje online y RRHH > 03

monografía

Ingeniería de Sistemas Interactivos: diseño y evaluación

Editores invitados: Sandra Baldassarri, J. A. Macías Iglesias y Jaime Urquiza Fuentes

Presentación. Tendencias en el desarrollo de software interactivo > 05
Sandra Baldassarri, José Antonio Macías Iglesias, Jaime Urquiza Fuentes

Analizador de señales inerciales para tracking de pies y manos > 07
Ernesto de la Rubia Cuestas, Antonio Díaz-Estrella

Creación de un visor de fotografías inmersivo basado en una interfaz de usuario natural > 13

Iván González Díaz, Ana Isabel Molina Díaz

Toolkits de desarrollo de interfaces tangibles: criterios de calidad en uso > 19
Rosa Gil Iranzo, Javier Marco Rubio, José Luis González Sánchez, Eva Cerezo Bagdasari, Sandra Baldassarri

Generación de interfaces de usuario a partir de wireframes > 24
Oscar Sánchez Ramón, Jesús Sánchez Cuadrado, Jesús J. García Molina, Jean Vanderdonck

Diseño de sistemas interactivos para entornos de control > 30
David Díez Cebollero, Rosa Romero Gómez, Sara Tena García, Paloma Díaz Pérez

Viabilidad de la metodología de evaluación heurística adaptada e implementada mediante Open-HEREDEUX > 35
Llúcia Masip Ardévol, Francisco Jurado Monroy, Toni Granollers Saltiveri, Marta Oliva Solé

Order effect y presencia de erratas en estudios de usuarios con eye tracking > 39
Mari-Carmen Marcos Mora, Luz Rello Sánchez

Aplicaciones de VoIP para móviles: Propuesta de un instrumento de evaluación centrado en el usuario > 43
Roland Fermenal, Laura Godoy, Albert Ribelles-Cortés, Mari-Carmen Marcos Mora

secciones técnicas

Estándares web

Verificación dinámica de composiciones en la Internet de las Cosas usando procesamiento de eventos complejos > 47
Javier Cubo, Laura González, Antonio Brogi, Ernesto Pimentel, Raúl Ruggia

Ingeniería del Software

Guía de estilo completa para nombrar los elementos de un esquema conceptual en UML/OCL > 52
David Aguilera Moncusí, Cristina Gómez Seoane, Antoni Olivé Ramon

Redes y servicios telemáticos

Multicast óptico con protección contra falla de nodo: Un enfoque multi-objetivo basado en ACO > 59
Aditado Vázquez, Diego P. Pinto-Roa, Enrique Dávalos

Software libre

Un análisis de las herramientas de software libre para la gestión ágil de proyectos de TI > 65
Matías Martínez, Javier Garzás

Referencias autorizadas > 69

sociedad de la información

Programar es crear > 76
Día Juliano

Julio Javier Castillo, Diego Javier Serrano, Marina, Elizabeth Cárdenas

asuntos interiores

Coordinación editorial / Programación de Novática / Socios Institucionales > 77

Tema del próximo número:

"Eficiencia energética en centros de procesos de datos"

David Aguilera Moncusí,
Cristina Gómez Seoane,
Antoni Olivé Ramon
*Universitat Politècnica de Catalunya,
BarcelonaTech*

<daguilera@essi.upc.edu>,
<antoni.olive@upc.edu>,
<cristina@essi.upc.edu>

1. Introducción

La especificación de los requisitos funcionales constituye una de las tareas principales en el desarrollo de sistemas de información, independientemente del método de desarrollo (en cascada, ágil, etc.) usado. Esta especificación consta básicamente del esquema conceptual del sistema, que describe la estructura de los conceptos procesados por el sistema, y el comportamiento del sistema en respuesta a los eventos que se producen.

En la actualidad, el lenguaje de modelización conceptual más utilizado es el UML (*Unified Modeling Language*). Se trata de un lenguaje propuesto por OMG (*Object Management Group*) a finales de los años 90, de aplicación general en el desarrollo de software y que se ha convertido en un estándar de facto.

Los modelos definidos en UML deben tener la calidad mínima que se requiera en cada caso, y seguir las mejores prácticas profesionales. Naturalmente, esto también se aplica a los modelos conceptuales.

Se acostumbra a distinguir tres tipos de calidad en los esquemas conceptuales: sintáctica, semántica y pragmática. El objetivo de la calidad pragmática es que todas las partes interesadas comprendan las partes del esquema que les son relevantes. En este sentido, los nombres de los elementos de los esquemas son muy importantes, porque tienen una gran influencia en su comprensión. Los esquemas son más fáciles de comprender si se han dado nombres adecuados a sus elementos.

La guía que proponemos a continuación trata cada uno de los elementos de los esquemas conceptuales que pueden ser nombrados en UML¹. Para cada elemento, la guía proporciona una forma gramatical y una frase patrón. Un nombre de un elemento cumple con la guía si tiene la forma gramatical indicada, y si la frase generada a partir de la frase patrón y el nombre es sintácticamente correcta y semánticamente significativa.

El uso sistemático de la guía que se propone debería contribuir a mejorar la calidad de los esquemas conceptuales en UML.

2. Tipos de entidad

En esta sección ofrecemos una propuesta

Guía de estilo completa para nombrar los elementos de un esquema conceptual en UML/OCL

Resumen: Uno de los objetivos principales a la hora de especificar los requisitos funcionales de un sistema de información es el facilitar su comprensión por parte de los usuarios finales. Así, cuando se usan lenguajes de modelización conceptual como es el caso de UML, resulta muy importante usar sistemáticamente una nomenclatura apropiada, rigurosa y coherente para los distintos elementos de los esquemas. El propósito de este artículo es proporcionar una guía de estilo completa para denominar los distintos elementos que pueden aparecer en los esquemas conceptuales en UML.

Palabras clave: Calidad pragmática, calidad semántica, calidad sintáctica, comprensibilidad, esquema conceptual, forma gramatical, frase patrón, guía de estilo, modelización conceptual, nomenclatura, UML.

para nombrar los tipos de entidad en UML (ver **tabla 1**). La regla que presentamos sirve para clases UML, clases asociativas y tipos de datos (los cuales incluyen las enumeraciones).

Está claro que nombres como *Persona*, *Silla*, *Factura* o *Categoría* (clases), *Trabajo* o *Afiliación* (clases asociativas), *Fecha* o *CantidadDeDinero* (tipos de datos) y *Sexo* o *Color* (enumeraciones), siguen la norma que hemos presentado.

Un ejemplo más complejo de nombre que también cumple la guía es *ViaParaVehículo-sAMotor*. El núcleo del sintagma nominal (*Via*) es un nombre contable y singular, y la frase que se obtiene en aplicarlo a G_{Is} es:

Una instancia de este tipo de entidad es una vía para vehículos a motor.

la cual es gramaticalmente correcta y tiene sentido.

Fijense que la norma G_{If} establece que “el núcleo del sintagma nominal tiene que ser un nombre contable y singular”. Esto quiere decir que nombres como *Agua* o *Plata* son inválidos, ya que no son contables. De hecho, lo que suele pasar con instancias de estos conceptos es que, en realidad, son “porciones” o “cantidades”. Para estos casos atípicos, podríamos ofrecer una frase distinta que incluyese esta noción de “porción” o “cantidad”:

Una instancia de este tipo de entidad es una porción de *minúsculas(E)*.

pero creemos que es mejor que sea el modelador quien le asigne un prefijo que convierta el nombre en contable. Así, por ejemplo,

Tipos de entidad

Sea E un tipo de entidad denominado, precisamente, E , entonces

G_{If} : El nombre de un tipo de entidad E tiene que ser un sintagma nominal cuyo núcleo sea contable y singular. El nombre debe escribirse con la “forma Pascal”; es decir, la primera letra de cada palabra debe estar en mayúscula y el nombre no puede contener espacios.

G_{Is} : La siguiente frase tiene que ser gramaticalmente correcta y tener sentido:

Una instancia del tipo de entidad denominado E es [un | una] *minúsculas(E)*.

donde *minúsculas(E)* es una función que devuelve el nombre E en minúsculas, separando las palabras con un espacio.

Cuando el tipo de entidad es implícito por el contexto donde estamos, la frase es, sencillamente:

Una instancia de este tipo de entidad es [un | una] *minúsculas(E)*.

Tabla 1. Nomenclatura para tipos de entidad UML.

tendríamos *BotellaDeAgua* o *PorciónDePlata*. Esto es lo que hemos hecho antes con el concepto *Dinero*, el cual hemos nombrado como *CantidadDeDinero*.

Si el nombre del tipo de entidad contiene acrónimos, éstos tienen unas reglas específicas (ver **tabla 2**).

Algunos ejemplos del uso de acrónimos son (1) *bdActiva* y *cssPorDefecto*; (2) *LectorDeCD* y *UsuarioDeLaBD*; y (3) *PáginaHtml* y *DocumentoXml*.

3. Atributos

En esta sección presentamos las reglas para nombrar atributos. Éstas sirven para atributos de una clase, de una clase asociativa, o de un tipo de datos.

Sea $E::A:T [min..max]$ un atributo denominado A de tipo T de un tipo de entidad E y cuyas multiplicidades mínima y máxima son min y max . Para simplificar esta guía de estilo, supondremos que el mínimo es 0 ó 1 y el máximo es 1 ó * (sin límite), ya que el caso general es fácilmente derivable.

La guía G_2 que proponemos para los atributos distingue aquellos que son *booleanos* de los que no lo son. En ambos casos, la frase que generaremos tiene en cuenta las multiplicidades. Sea G_2 la regla para atributos no booleanos y G_2' la regla para los booleanos.

3.1. Regla para atributos no booleanos

Si T no es el tipo de datos *booleano*, entonces la regla que aplicaremos para el atributo es la expresada en la **tabla 3**.

El ejemplo de la **figura 1** muestra cuatro atributos no *booleanos*, los cuales siguen la anterior regla. Las frases que generan son:

- Una oferta tiene un precio.
- Una oferta puede tener una fecha de caducidad.
- Una oferta puede tener cero o más imágenes.
- Una cantidad de dinero tiene un valor.
- Un producto tiene uno o más colores.
- Una tienda tiene un nombre.

3.2. Regla para atributos booleanos

Si T es *booleano*, entonces suponemos que la multiplicidad mínima es 0 ó 1 y la máxima 1 (ver **tabla 4**).

El ejemplo de la **figura 1** muestra tres atributos *booleanos*, los cuales siguen la anterior norma. Las frases que generan son:

- Una tienda abre o no abre todo el día, o no se sabe.
- Una tienda es o no es económica.
- Un producto caduca o no caduca.

Uso de mayúsculas y minúsculas en acrónimos

1. Si el acrónimo es la primera palabra de un nombre y, además, el nombre tiene que seguir la “forma Camel” (es decir, la primera palabra se escribe en minúsculas, las demás empiezan con mayúscula y no puede contener espacios), entonces escriban el acrónimo todo en minúsculas.
2. Respetando la primera regla, escriban siempre en mayúsculas todas las letras de un acrónimo de sólo dos letras.
3. Respetando la primera regla, escriban sólo la primera letra en mayúscula para cualquier acrónimo de tres o más letras.

Tabla 2. Reglas específicas si el tipo de entidad contiene acrónimos.

Atributos no booleanos

Sea $E::A:T [min..max]$ un atributo denominado A de tipo T de un tipo de entidad E , entonces

- G_{2f} : El nombre A tiene que ser un sintagma nominal cuyo núcleo sea contable y singular, escrito utilizando la “forma Camel”.
- G_{2s} : Una de las siguientes frases tiene que ser gramaticalmente correcta y tener sentido:
- Si $min = 0$ y $max = 1$:
[Un|Una] *minúsculas(E)* puede tener [un|una] *minúsculas(A)*.
 - Si $min = 0$ y $max > 1$:
[Un|Una] *minúsculas(E)* puede tener cero o más *minúsculas(plural(A))*.
 - Si $min = max = 1$:
[Un|Una] *minúsculas(E)* tiene [un|una] *minúsculas(A)*.
 - Si $min > 0$ y $max > 1$:
[Un|Una] *minúsculas(E)* tiene [un|una] o más *minúsculas(plural(A))*, donde *plural(A)* es una función que devuelve la forma plural de A .

Tabla 3. Regla para atributos no booleanos.

Atributos booleanos

Sea $E::A:Boolean [min..max]$ un atributo booleano denominado A de un tipo de entidad E , entonces

- G_{2f} : El nombre A tiene que ser un sintagma verbal en tercera persona del singular, escrito con la “forma Camel”.
- G_{2s} : La siguiente frase tiene que ser gramaticalmente correcta y tener sentido:
- [Un|Una] *minúsculas(E)* *minúsculas(normalONegado(A))* [, o no se sabe].
- donde la última parte es opcional y se incluye si la multiplicidad mínima es cero.
- La función *normalONegado(A)* expande A añadiéndole su forma negada. Por ejemplo:

normalONegado(esDerivado) = esONoesDerivado
normalONegado(tieneHijos) = tieneONoTieneHijos

Tabla 4. Regla para atributos booleanos.

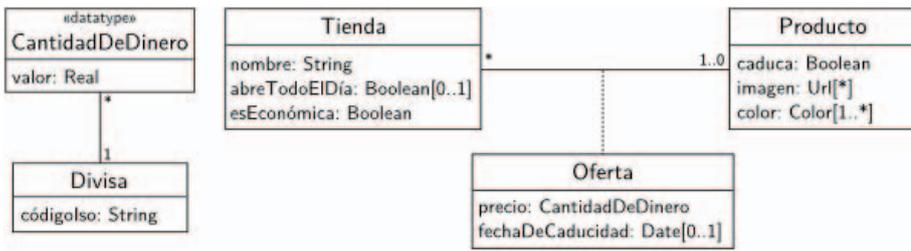


Figura 1. Ejemplos de atributos booleanos y no booleanos.

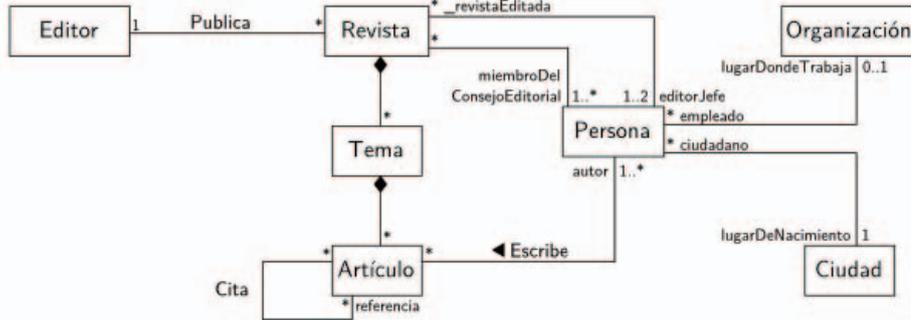


Figura 2. Ejemplos de asociaciones y roles.

Asociaciones binarias

Sea $R(p_1:E_1 [min_1, max_1], p_2:E_2 [min_2, max_2])$ una asociación binaria, entonces

G_{3f}^a : El nombre de la asociación R tiene que ser un sintagma verbal en tercera persona del singular, escrito utilizando la “forma Pascal”.

G_{3s}^a : Una de las siguientes frases tiene que ser gramaticalmente correcta y tener sentido:

- Si $min_2 = 0$ y $max_2 = 1$:
 $[Un|Una] \text{ minúsculas}(E_1) \text{ minúsculas}(R)$, como mucho, $[un|una] \text{ minúsculas}(E_2)$.
- Si $min_2 = 1$ y $max_2 = 1$:
 $[Un|Una] \text{ minúsculas}(E_1) \text{ minúsculas}(R) [un|una] \text{ minúsculas}(E_2)$.
- Si $min_2 = 0$ y $max_2 = *$:
 $[Un|Una] \text{ minúsculas}(E_1) \text{ minúsculas}(R)$ cero o más $\text{minúsculas}(\text{plural}(E_2))$.
- Si $min_2 = 1$ y $max_2 = *$:
 $[Un|Una] \text{ minúsculas}(E_1) \text{ minúsculas}(R) [un|una]$ o más $\text{minúsculas}(\text{plural}(E_2))$.

Tabla 5. Regla para las asociaciones binarias en UML.

4. Asociaciones

En esta sección presentamos cómo nombrar asociaciones en UML. Distinguimos entre asociaciones binarias y *n*-arias porque, como veremos, las respectivas reglas son bastante diferentes.

4.1. Asociaciones binarias

Sea $R(p_1:E_1 [min_1, max_1], p_2:E_2 [min_2, max_2])$ una asociación binaria entre dos tipos de entidad E_1 y E_2 , cuyos roles son p_1 y p_2 con multiplicidades $[min_1, max_1]$ y $[min_2, max_2]$, respectivamente. En UML, el orden de los participantes en la asociación se muestra con una pequeña flecha sólida al lado del nombre R . Cuando el orden es de arriba a abajo o de izquierda a derecha, puede omitirse.

UML permite tres nombres explícitos en una asociación, todos ellos opcionales: el nombre de la asociación R y los dos nombres de rol p_1 y p_2 . Los roles siempre tienen un nombre implícito cuando no se les da un nombre explícito, el cual coincide con el nombre del tipo de entidad al que están vinculados, pero empezando en minúscula.

En general, no hay nombres implícitos para las asociaciones. Sin embargo, UML incluye dos tipos especiales de asociación, las agregaciones y las composiciones, los cuales sí tienen nombres predefinidos y, por lo tanto, no debemos darles ninguno.

La regla G_3 que proponemos para las aso-

ciaciones binarias en UML tiene dos partes: una G_3^a para el nombre de la asociación R (ver tabla 5) y otra para los nombres de rol G_3^r (ver tabla 6).

Como ya hemos dicho, los nombres de rol pueden ser implícitos o explícitos. Distinguimos entre dos tipos de nombre de rol explícitos: los *externos* y los *internos*. En UML, hay casos en que tenemos que asignar un nombre rol a un miembro de una asociación y, por tanto, no queremos que se utilicen en la verbalización del esquema. Para esos casos diferenciamos entre ambos tipos de nombre y distinguimos los internos añadiendo un guión bajo como prefijo.

La regla que proponemos para los nombres de rol externos se basa en la idea que “un rol se puede ver como un atributo no booleano del tipo de entidad del otro extremo” y, por consiguiente, la regla que deberíamos aplicar es la misma que la que hemos descrito para esos atributos. En particular, si $R(p_1:E_1 [min_1, max_1], p_2:E_2 [min_2, max_2])$ es una asociación binaria, entonces los roles p_1 y p_2 se pueden ver como los siguientes atributos:

$$E_2::p_1:E_1 [min_1, max_1] \text{ y } E_1::p_2:E_2 [min_2, max_2]$$

y, así, tienen que seguir la regla G_2 . La regla en cuestión, adaptada a los nombres de rol, se muestra en la tabla 6.

Como ejemplos, consideremos las nueve asociaciones que se muestran en la figura 2. Hay dos asociaciones que son composiciones (*Revista–Tema* y *Tema–Artículo*) y que, por consiguiente, no necesitan nombre. Hay cuatro asociaciones con nombres explícitos que siguen la regla G_{3f}^a , generando las siguientes frases:

- Un editor puede publicar cero o más revistas.
- Un artículo puede citar cero o más artículos.
- Una persona puede escribir cero o más artículos.
- Una persona nació en una ciudad.

Hay ocho nombres de rol explícitos, los cuales siguen la regla G_{3f}^r :

- Un artículo tiene cero o más referencias.
- Un artículo tiene uno o más autores.
- Una ciudad tiene cero o más ciudadanos.
- Una persona tiene un lugar de nacimiento.
- Una revista tiene uno o más editores jefe.
- Una revista tiene uno o más miembros del consejo editorial.
- Una organización tiene cero o más empleados.
- Una persona tiene, como mucho, un lugar donde trabaja.

Fijense que el nombre de rol *_revistaEditada* es interno. En este ejemplo, necesitamos asignarle un nombre porque hay una restricción

Nombres de rol externos en asociaciones binarias

Sea $R(p_1:E_1 [min_1, max_1], p_2:E_2 [min_2, max_2])$ una asociación binaria, entonces

G_{3f} : Suponiendo que p_i es un nombre de rol externo, p_i tiene que ser sintagma nominal cuyo núcleo sea contable y singular, escrito utilizando la “forma Camel”.

G_{3s} : Una de las siguientes frases tiene que ser gramaticalmente correcta y tener sentido:

- Si $min_i = 0$ y $max_i = 1, i \neq j$:

[Un|Una] *minúsculas*(E_i) puede tener [un|una] *minúsculas*(p_j).

- Si $min_i = 0$ y $max_i > 1, i \neq j$:

[Un|Una] *minúsculas*(E_i) puede tener cero o más *minúsculas*(plural(p_j)).

- Si $min_i = max_i = 1, i \neq j$:

[Un|Una] *minúsculas*(E_i) tiene [un|una] *minúsculas*(p_j).

- Si $min_i > 0$ y $max_i > 1, i \neq j$:

[Un|Una] *minúsculas*(E_i) tiene [un|una] o más *minúsculas*(plural(p_j)).

plural(p_j) es una función que devuelve la forma plural de p_j .

Tabla 6. Regla para los nombres de rol de las asociaciones binarias en UML.

Asociaciones n-arias

Sea $R(p_1:E_1 [min_1, max_1], \dots, p_n:E_n [min_n, max_n])$ una asociación n -aria, entonces

G_{4r} : El nombre de la asociación R tiene que ser un sintagma verbal en tercera persona del singular, utilizando la “forma Pascal”. El nombre debe incluir $n-1$ subcadenas de texto de la forma $\langle p_i \rangle$, una por cada rol p_2, \dots, p_n , donde p_i es el nombre de rol explícito (o implícito si no tiene). Los nombres de rol se escriben usando la “forma Camel”.

En UML, los participantes de una asociación están ordenados, pero para las asociaciones n -arias este orden no se muestra gráficamente. Fíjense que el primer participante p_1 de la asociación es el que no aparece en R .

G_{4s} : La siguiente frase tiene que ser gramaticalmente correcta y tener sentido

[Un|Una] *minúsculas*(E_1) *minúsculas*(expanded(R))

La función *expanded*(R) devuelve R en minúsculas y utilizando el espacio como delimitador y substituye cada $\langle p_i \rangle$ por “[un|una] (E_i)”. Fíjense que los nombres de rol no aparecen en el resultado de *expanded*(R).

Tabla 7. Regla para los nombres de rol de las asociaciones binarias en UML.

en UML que impide que un tipo de entidad (como *Persona*) pueda participar en dos asociaciones tales que los demás participantes tengan el mismo nombre de rol.

4.2. Asociaciones n-arias

Sea $R(p_1:E_1 [min_1, max_1], \dots, p_n:E_n [min_n, max_n])$ una asociación n -aria ($n > 2$) entre los tipos de entidad E_1, \dots, E_n , cuyos roles son p_1, \dots, p_n , respectivamente. En UML hay hasta $n + 1$ nombres explícitos vinculados a esta asociación: el nombre de la asociación R y los nombres de rol p_1, \dots, p_n .

Los roles p_1, \dots, p_n siempre tienen un nombre implícito que se utiliza cuando no se les ha dado uno explícito. Este nombre implícito es el mismo nombre que el tipo de entidad al que están vinculados, empezando en minúscula. Cuando $E_i = E_j$, entonces p_i o p_j o ambos roles tienen que tener un nombre explícito. No hay nombres implícitos para las asociaciones n -arias. Dos asociaciones distintas pueden tener el mismo nombre.

La regla G_4 que proponemos para las asociaciones UML n -arias es la mostrada en la **tabla 7**.

Los ejemplos de la **figura 3** producen el siguiente resultado:

Suministra $\langle producto \rangle A \langle proyecto \rangle$ genera:

Un proveedor suministra un producto a un proyecto.

Vende $\langle producto \rangle De \langle proveedor \rangle En \langle país \rangle$ genera:

Un vendedor vende un producto de un proveedor en un país.

PuedeSustituir $\langle sustituto \rangle En \langle proyecto \rangle$ genera:

Un producto puede sustituir un producto en un proyecto.

5. Invariantes

En UML, un invariante es una restricción estática vinculada a un tipo de entidad. La restricción puede estar definida como una expresión OCL cuyo contexto es el tipo de entidad. Esta expresión debe ser cierta para todas las instancias de ese tipo. Los invariantes pueden tener nombre. Por ejemplo:

`context Departamento inv tieneSuficientesEmpleados: self.numeroDeEmpleados > 50`

es un invariante denominado *tieneSuficientesEmpleados*.

En UML, las restricciones *untargeted* deben definirse como invariantes cuyo contexto sea un tipo de entidad *singleton*. Por ejemplo, un sistema que gestione congresos podría tener una restricción indicando que “debe haber como mínimo una sala grande”, la cual podría expresarse como el siguiente invariante:

`context GestorDeCongresos inv tieneComoMinimoUnaSalaGrande: Sala.allInstances()->select(tamaño = Tamaño::grande)->size() > 0`

donde se supone que *GestorDeCongresos* es un tipo de entidad *singleton*, que *tamaño* es un atributo de *Sala* y que *Tamaño* es un tipo de datos enumerado.

La regla que proponemos para denominar un invariante I es la mostrada en la **tabla 8**.

Los siguientes ejemplos muestran la aplicación de nuestra regla:

`context Persona inv seIdentificaPorSuNombre:`

`context Matrimonio inv es una relación entre un hombre y una mujer:`

`context Seccion inv seIdentificaPorElCursoYElNombre:`

context Curso inv noPuedeSerSucesorDeSiMismo:

context Curso inv consta de secciones cuyos profesores son expertos en el tema del curso:

los cuales generan las siguientes frases:

- Una persona se identifica por su nombre.
- Un matrimonio es una relación entre un hombre y una mujer.
- Una sección se identifica por el curso y el nombre.
- Un curso no puede ser sucesor de sí mismo.
- Un curso consta de secciones cuyos profesores son expertos en el tema del curso.

6. Tipos de evento y operaciones de sistema

A continuación nos centraremos en la parte de comportamiento de los esquemas conceptuales. En general, un esquema de comportamiento se basa en un conjunto de tipos de evento u operaciones de sistema, y/o un conjunto de diagramas de transición de estados.

En esta sección veremos los tipos de evento (u operaciones de sistema) y, en la siguiente, los diagramas de transición de estados.

6.1. Tipos de evento

Sea *Ev* un tipo de evento. La regla que proponemos para nombrarlo es la mostrada en la **tabla 9**.

La **figura 4** muestra un ejemplo con tres tipos de evento: *NuevaVenta*, *NuevoItemDeVenta* y *FinDeVenta*, los cuales generan las siguientes frases:

- Una instancia de este tipo de evento es un evento de nueva venta.
- Una instancia de este tipo de evento es un evento de nuevo item de venta.
- Una instancia de este tipo de evento es un evento de fin de venta.

6.2. Operaciones de sistema

Sea *Op* una operación de sistema. La regla que proponemos para las operaciones de sistema es la mostrada en la **tabla 10**.

Fíjense que la frase incluye simplemente *Op*, en lugar de *minúsculas(Op)*. El motivo es que los nombres que se dan a la mayoría de operaciones son, en la práctica, descripciones compactas de lo que harán cuando se invoquen. Por lo tanto, no parece posible dar una regla general para su correcta verbalización.

La **figura 4** muestra también tres operaciones de sistema, denominadas *CreaNuevaVenta*, *CreaItemDeVenta* y *FinalizaVenta*, las cuales se corresponden a los tipos de evento que ya hemos comentado. Las frases generadas son:

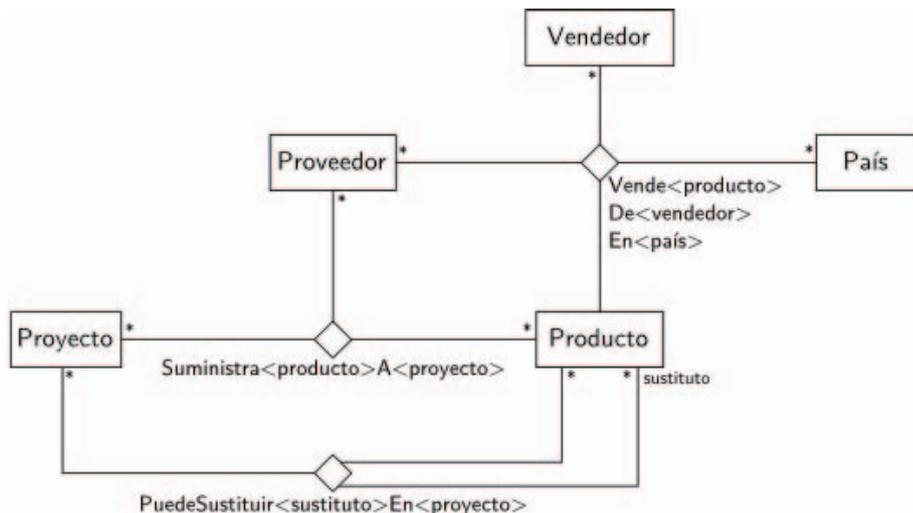


Figura 3. Ejemplos de asociaciones n-arias.

Invariantes

- G_{sf} : El nombre de un invariante tiene que ser un sintagma verbal en tercera persona del singular. El nombre tiene que estar escrito usando la "forma Camel" o, si no hay ninguna palabra que pueda generar ambigüedad con OCL, usando la forma normal (es decir, usando minúsculas y espacios).
- G_{ss} : Si *I* es el nombre del invariante y *E* su contexto, entonces la siguiente frase tiene que ser gramaticalmente correcta y tener sentido:
[Un | Una] *minúsculas(E) minúsculas(I)*.

Tabla 8. Regla para denominar un invariante.

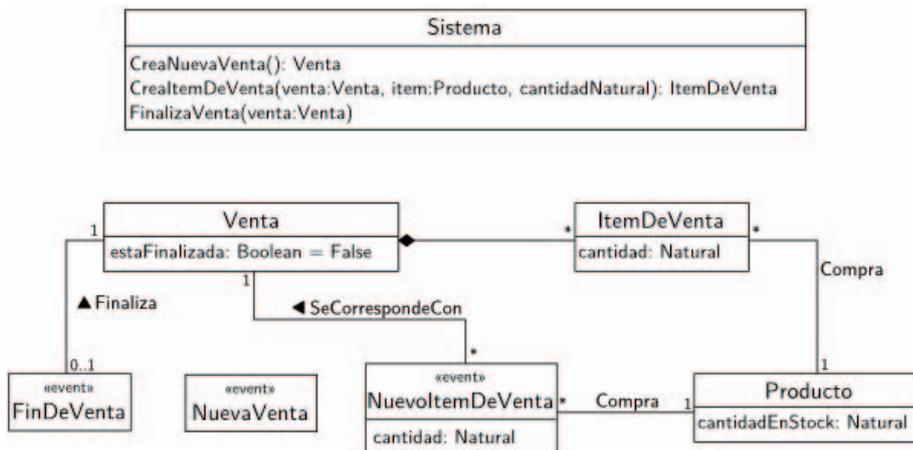


Figura 4. Fragmento de un esquema estructural y la especificación de su comportamiento utilizando tipos de evento y operaciones de sistema.

- Una invocación a esta operación pide al sistema que ejecute el comando *CreaNuevaVenta*.
- Una invocación a esta operación pide al sistema que ejecute el comando *CreaItemDeVenta*.
- Una invocación a esta operación pide al sistema que ejecute el comando *FinalizaVenta*.

6.3. Precondiciones

Una precondición es una condición que tienen que satisfacer las características del evento y los parámetros de la operación y/o

el sistema de información cuando el evento o la operación va a producirse. En UML, las precondiciones pueden tener un nombre y se especifican formalmente en OCL.

Sea *Pre* una precondición del tipo de evento *Ev* o de la operación de sistema *Op*. La regla que proponemos para darle nombre es la mostrada en la **tabla 11**.

En el ejemplo de la **figura 4**, el tipo de evento *NuevoItemDeVenta* (o la operación de siste-

Tipos de evento

G_{gr} : El nombre de un tipo de evento tiene que ser un sintagma nominal cuyo núcleo sea contable y singular. El nombre tiene que escribirse usando la “forma Pascal”. Fíjense que esta regla es igual a la propuesta para los tipos de entidad (G_p).

G_{es} : Si Ev es el nombre de un tipo de evento, entonces la siguiente frase tiene que ser gramaticalmente correcta y tener sentido:

Una instancia del tipo de evento denominado Ev es un evento de *minúsculas*(Ev).

Cuando el tipo de un evento es implícito por el contexto, la frase es simplemente:

Una instancia de este tipo de evento es un evento de *minúsculas*(Ev).

Tabla 9. Regla para nombrar tipos de evento.

Operaciones de sistema

G_{vr} : El nombre de una operación de sistema tiene que ser un sintagma verbal, en imperativo. El nombre debe escribirse usando la “forma Pascal”.

G_{vs} : Si Op es el nombre de la operación de sistema, entonces la siguiente frase tiene que ser gramaticalmente correcta y tener sentido:

Una invocación a la operación denominada Op pide al sistema que ejecute el comando “ Op ”.

Cuando la operación es implícita por el contexto, la frase es:

Una invocación a esta operación pide al sistema que ejecute el comando “ Op ”.

Tabla 10. Regla para denominar operaciones de sistema.

Precondiciones

G_{sv} : El nombre de una precondición tiene que ser un sintagma verbal. El nombre tiene que estar escrito en “forma Camel” o usando la forma “normal” (ver G_p).

G_{ss} : Si Pre es el nombre de la precondición, entonces la siguiente frase tiene que ser gramaticalmente correcta y tener sentido:

[Antes que suceda el evento de *minúsculas*(Ev) | Antes que se ejecute la operación Op], *minúsculas*(Pre).

Cuando el tipo de evento o la operación de sistema es implícito por el contexto, la frase es:

[Antes que suceda este evento | Antes que se ejecute esta operación], *minúsculas*(Pre).

Tabla 11. Regla para nombrar las precondiciones.

ma equivalente *CrearItemDeVenta*) tiene dos precondiciones con los nombres: “la venta no está finalizada” y “hay suficiente stock”:

```
context NuevoltemDeVenta::effect() pre la
venta no esta finalizada:
not venta.estaFinalizada pre hay
suficiente stock: cantidad >= producto.
cantidadEnStock
```

el cual genera las siguientes frases:

Antes que suceda este evento, la venta no está finalizada.

Antes que suceda este evento, hay suficiente stock.

6.4. Postcondiciones

El efecto de los eventos o las operaciones de sistema se describe mediante una o más postcondiciones.

Una postcondición es una condición que el

sistema de información tiene que satisfacer después que el evento haya sucedido (o que la operación se haya ejecutado). En general, una postcondición también puede incluir información sobre las salidas generadas (*queries* o *communications*). En UML, las postcondiciones pueden tener nombre y se especifican formalmente en OCL.

Sea *Post* una postcondición del tipo de evento Ev o de la operación de sistema Op . La regla que proponemos para denominar las postcondiciones se muestra en la **tabla 12**.

En el ejemplo de la **figura 4**, el tipo de evento *NuevaVenta* (o la operación de sistema equivalente *CreaNuevaVenta*) tiene una postcondición denominada “se ha creado una venta”:

```
context NuevaVenta::effect() post se ha
creado una venta:
venta.ocllsNew() and venta.
ocllsTypeOf(Venta)
```

la cual genera la siguiente frase:

Después que suceda este evento, se ha creado una venta.

De la misma forma, el tipo de evento *NuevoItemDeVenta* (o la operación de sistema equivalente *CrearItemDeVenta*) tiene dos postcondiciones denominadas: “se ha creado un nuevo ítem de venta” y “se ha reducido el stock del producto”, las cuales generan:

Después que suceda este evento, se ha creado un nuevo ítem de venta.

Después que suceda este evento, se ha reducido el stock del producto.

7. Estados

En UML, un estado es una “condición o situación durante la vida de un objeto en donde satisface una cierta condición, ejecuta una cierta actividad, o espera algún evento”.

Sea S un estado de un tipo de entidad E . La regla que proponemos para denominar S es la mostrada en la **tabla 13**.

A continuación tienen unos cuantos ejemplos:

- *Microondas: ListoParaCocinar, PuertaAbierta, Cocinando, ...*
- *Paciente: Entrando, Admitido, BajoObservación, DadoDeAlta.*
- *Llamada Telefónica: NoDisponible, SeñalDeLlamada, Llamando, ...*

Algunas de las frases que se generarían con los anteriores ejemplos son:

Un microondas está cocinando.

Un paciente está entrando.

Un paciente está admitido.

Postcondiciones

G_{9f} : El nombre de una postcondición tiene que ser un sintagma verbal. El nombre tiene que estar escrito en “forma Camel” o usando la forma “normal” (ver G_{7f}).

G_{9g} : Si *Post* es el nombre de la postcondición, entonces la siguiente frase tiene que ser gramaticalmente correcta y tener sentido:

[Después que suceda el evento de *minúsculas*(Ev) | Después que se invoque la operación Op], *minúsculas*(Post).

Cuando el tipo de evento o la operación de sistema es implícito por el contexto, la frase es:

[Después que suceda este evento | Después que se invoque esta operación], *minúsculas*(Post).

Tabla 12. Regla para denominar las postcondiciones.

Estados

G_{10f} : El nombre de un estado tiene que ser un adjetivo o complemento nominal, escrito utilizando la “forma Pascal”.

G_{10g} : Si *S* es el nombre de un estado de un tipo de entidad *E*, entonces una de las siguientes frases como mínimo tiene que ser gramaticalmente correcta y tener sentido:

[Un | Una] *minúsculas*(*E*) [es | está] *minúsculas*(*S*).

[Un | Una] *minúsculas*(*E*) está en el estado de *minúsculas*(*S*).

Tabla 13. Regla para denominar estados.

Nota

¹ El texto completo de la guía puede verse en el artículo de los mismos autores: *A complete set of guidelines for naming UML conceptual schema elements*. Data and Knowledge Engineering vol 88, pp. 60-74 (2013). Hay una versión de la guía para las lenguas catalana, castellana e inglesa.