Brazilian Journal of Development

Diseño Conceptual de Bases de Datos con UML

Conceptual Design of Databases with UML

Recebimento dos originais: 20/12/2018
Aceitação para publicação: 21/01/2019

Arturo Carlos Servetto
Licenciado en Informática por la Universidad Nacional de La Plata
Instituição: Universidad de Buenos Aires
Endereço: Av. Paseo Colón 850 –Ciudad Autónoma de Buenos Aires - Argentina
E-mail: aservetto@fi.uba.ar

RESUMEN
En atención a la preminencia actual de las metodologías ágiles en el ejercicio profesional, se plantea la importancia de adoptar métodos, técnicas, lenguajes y herramientas actuales de la fase de análisis de la ingeniería de software para el modelado conceptual de bases de datos, tanto en los currículos de carreras de informática, con los objetivos de mejorar la calidad de la enseñanza y concientizar a los futuros profesionales de la importancia del análisis y su documentación, como en el trabajo profesional, para mejorar la calidad de procesos y de productos. Se propone un estándar para modelar entidades y relaciones con UML y se analiza sus fortalezas y debilidades.

Palavras-chave: modelado conceptual de bases de datos, análisis de dominio en UML, estereotipos de clases entidad, asociaciones de identificación.

ABSTRACT
In consideration of the current primacy of agile methodologies in professional practice, the importance of adopting current methods, techniques, languages and tools of the analysis phase of software engineering for the conceptual modeling of databases, both in the curricula of computer science careers, with the objectives of improving the quality of teaching and raising awareness among future professionals of the importance of analysis and documentation, as well as in professional work, to improve the quality of processes and products. A standard is proposed for modelling entities and relationships with UML and its strengths and weaknesses are analyzed.

Keywords: conceptual modeling of databases, UML domain analysis, entity class stereotypes, identification associations.

1 INTRODUCCIÓN

En los planes de estudio de carreras de informática no suele faltar una o más asignaturas sobre bases de datos que generalmente comprenden contenidos relativos al diseño conceptual y lógico de datos, así como a la arquitectura de SGBD (Sistemas de Gestión de Bases de Datos o, en inglés,*DBMS–Data Base Management Systems*) [Connolly y Begg 2015], con modelos lógicos y de sistemas de gestión relacionales, por ser el paradigma relacional el dominante en el área de bases de datos. Por otra parte, tampoco falta una o más asignaturas sobre Ingeniería de Software, que generalmente comprenden análisis y diseño de datos con el modelo de objetos [Jacobson et al.

1999] [Booch et al. 2005] con *UML (UnifiedModelingLanguage)*[Debrauwer y Van der Heyde 2016], por ser el paradigma de objetos el que domina la ingeniería de software.

Para el modelado conceptual de datos es común que se utilice el modelo de Entidades y Relaciones (ER) del Dr. Peter Chen (1977). El Modelo ER ha sido la base para diversas metodologías sobre análisis y diseño de sistemas, herramientas de ingeniería de software asistida por computador *(CASE–ComputerAided Software Engineering)* y repositorios de sistemas, de ahí su persistente vigencia. Sin embargo, en el campo profesional es común que se saltee la etapa de diseño conceptual para producir directamente diagramas lógicos relacionales, lo mismo que en ingeniería de software, dada la preferencia por las metodologías ágiles [Beck et al. 2001], muy livianas en lo que respecta a la documentación, es común que se saltee la etapa de análisis propuesta por el Proceso Unificado de Desarrollo [Debrauwer y Van der Heyde 2016], precisamente la de modelado conceptual, para producir directamente diagramas de clases de diseño.

Tanto en asignaturas de bases de datos como de ingeniería de software se suelen abordar contenidos de modelado conceptual usando modelos distintos y prácticamente equivalentes. Cabe preguntarse en qué proporción el descarte del modelado conceptual de datos y del modelado de dominios con clases de análisis en la práctica profesional, productos precisamente de la etapa fundamental del desarrollo de software, se deben a la resistencia natural a duplicar esfuerzos para representar la misma información usando dos lenguajes distintos, respecto de otras causas como la simple “ley del menor esfuerzo”, la falta de tiempo y recursos, los costos de mantenimiento de la documentación, etc. Lo cierto es que esta falencia de documentación del análisis implica riesgos importantes en el desarrollo de sistemas, ya que la no documentación aumenta el riesgo de errores, y el costo de enmendar errores de análisis es muy alto.

He aquí entonces algunos interrogantes a considerar para el diseño curricular de carreras de informática: ¿Cómo se articulan las asignaturas de bases datos con las de ingeniería de software? ¿Se puede obviar en los contenidos de asignaturas de bases de datos el diseño conceptual de datos para centrarse exclusivamente en las operaciones y la arquitectura de los sistemas de gestión? ¿Si se adopta el modelo de clases de análisis como única herramienta para modelar datos a nivel conceptual, dónde se aborda la derivación de modelos de clases de análisis a modelos lógicos relacionales, si las bases de datos se continúan implementando con este último modelo?

2 DESAFÍOS EN EL MODELADO CONCEPTUAL DE DATOS CON UML

Ya se planteó como fortalezas del modelo ER de Chen su antigüedad y profusión de herramientas CASE que lo sustentan, pero pese a que también hay muchas herramientas que sustentan el modelado de clases en UML [Visual Paradigm 2018] [StarUML 2018][Enterprise

Architect 2018][ArgoUML 2018][Lucidchart 2018], éstas en general no suelen inducir a la separación entre modelos de análisis y modelos de diseño, es decir, no facilitan o favorecen la adopción de un proceso sistemático para el diseño de datos.

También se observa en textos sobre bases de datos [Connolly y Begg 2015], adaptaciones del UML para representar atributos compuestos, que tampoco son sustentadas por herramientas CASE ni por el mismo estándar de UML, así como el uso de primitivas propias de un nivel lógico, como agregaciones y composiciones, que no son propias de diagramas de análisis o de nivel conceptual.

Quizá un problema de fondo sea la no minimalidad del UML, o cierta laxitud normativa del Proceso Unificado de Desarrollo, que conspira en contra de la estandarización de su uso. Por ejemplo, para las clases de análisis, la especificación de tipos de atributos es optativa, entonces un atributo compuesto puede representarse definiendo una clase para asignársela como tipo, lo que por coherencia obligaría a definir tipos para todos los atributos, algo impropio del modelado conceptual, o, en el otro extremo, representar el atributo con una composición, pero que es propia de diagramas de diseño, es decir, de nivel de implementación en el paradigma de objetos.

Otro problema radica en los inconvenientes que suelen presentar las herramientas de modelado de clases para representar identificadores externos, mixtos o compuestos, que en la práctica obligan a concebir convenciones ad hoc para cada herramienta, o a superpoblar diagramas con notas aclaratorias. Para el modelado de datos es muy importante poder representar todos los identificadores posibles de una entidad, para luego distinguir claves candidatas en el modelo relacional.

El modelado conceptual de datos para un sistema, como todo problema de ingeniería, implica una resolución sistemática, disciplinada y cuantificable. Entonces, para que el modelado conceptual sea sistemático se debe respetar un proceso; para que sea disciplinado, en dicho proceso deben utilizarse instrumentos (lenguajes de modelado) y obtener productos (diagramas) con las reglas de la disciplina (la Informática), y para que sea cuantificable se deben usar métricas, también definidas en la disciplina, para evaluar la calidad o complejidad de los productos obtenidos.

3 REPRESENTACIÓN DE MODELOS DE ENTIDADES Y RELACIONES CON UML

En esta sección se describe la técnica y convenciones de representación en UML que se emplean en un curso de la asignatura Base de Datos, obligatoria para las carreras de Licenciatura en Análisis de Sistemas e Ingeniería en Informática, que se dicta en la Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA). Los contenidos de la asignatura incluyen diseño conceptual de datos, modelo relacional y diseño lógico de datos, lenguajes relacionales y diseño físico de

datos; seguridad, gestión de transacciones y procesamiento de consultas, e inteligencia de negocios. Como herramienta de modelado se sugiere el uso de una edición comunitaria (de uso libre y gratuito) de Visual Paradigm.

3.1 DIAGRAMAS DE ANÁLISIS DE DOMINIO

Se recomienda el uso de diagramas de dominio de UML correspondientes a la fase de análisis del Proceso Unificado de Desarrollo, para modelar entidades con clases de ese estereotipo *(entityclasses)* [Arlow y Neustadt 2005] [Farve 2003] [Kroll y Krutchen 2003]. En estos diagramas sólo aparecen clases entidad, que pueden participar en jerarquías, clases asociación para representar relaciones entre entidades con características propias, y asociaciones simples entre entidades para representar relaciones sin atributos. Para que estos diagramas resulten equivalentes a los de Chen, se propone no definir tipos de atributos, salvo para casos especiales de dominios definidos por extensión, que se pueden representar como clases estereotipadas como enumeraciones.

Para favorecer el proceso de análisis de la información y obtener diagramas fáciles de comprender, se definen nuevos estereotipos de clases entidad y de asociaciones. A continuación, se contextualizan los principales conceptos de modelado de entidades y relaciones [Batini, Ceri y Navathe 1994], empleando el UML como lenguaje de representación. Diseñar datos a nivel conceptual implica:

- Detectarlos en el sistema de información para el que se está desarrollando el sistema informático que los gestionará y especificar vínculos entre los mismos, en un proceso de abstracción que se denomina clasificación: se trata de identificar datos que representen cosas (clientes, productos, servicios, etc.), eventos (pedidos y ventas de productos, reservas y ventas de servicios, etc. –tienen fecha de ocurrencia) y procesos (trámites, órdenes de servicio –tienen un estado que cambia cronológicamente), así como de identificar vínculos entre ellos, y asignarles un conjunto o clase de pertenencia, que en el caso de que sean datos se denomina “entidad”, y en el caso de que sean vínculos, “relación”.

  Las entidades que agrupan representaciones de cosas se dicen “maestras”, y las que agrupan representaciones de eventos o de procesos, “transaccionales”, y para ayudar a la comprensión de diagramas se propone definir estereotipos <Cosa>, <Proceso> y <Evento> empleando colores de fondo distintivos para las representaciones.

  La clasificación de los vínculos ente datos comprende también determinar la cardinalidad de cada entidad en una relación, es decir, la cantidad mínima y máxima de datos de esta, otra u otras entidades que constituyen la relación con las que se puede vincular un dato individual de la entidad.

- Especificar las características o atributos de los datos y de los vínculos entre éstos que sean pertinentes para el sistema de información, en un proceso de abstracción que se denomina análisis: se determina las cualidades o estado de los datos y de los vínculos entre datos.

  La especificación de los atributos implica también determinar la cardinalidad de los valores que pueden tomar para un dato en particular, es decir la cantidad mínima y máxima de valores atómicos que puede tener cada atributo para un dato.

  Hay atributos que se pueden describir en términos de otros atributos: son llamados atributos compuestos. Tanto el atributo compuesto como sus componentes tienen su propia cardinalidad, y para representarlos en UML se propone definir un estereotipo <Atributo> con un color de fondo distintivo para su representación.

Hay atributos que pueden desconocerse al momento de registrar un dato o representar una característica que un dato particular puede no tener: son denominados **atributos opcionales**. Los atributos opcionales tienen cardinalidad mínima 0. Hay atributos que pueden describirse en términos de una lista de valores del mismo tipo o estructura: son conocidos como atributos polivalentes. Los atributos polivalentes tienen cardinalidad máxima fija (una constante) o indeterminada (*). Como la cardinalidad más común de los atributos es 1, es decir, la mayoría de los atributos son obligatorios y monovalentes, ésta se asume por defecto sin necesidad de denotarla. En UML, la denotación de cardinalidades se realiza usando calificadores [0..1] (opcional monovalente), [0..*] (opcional polivalente), [1..*] (obligatorio y polivalente). También, para optimizar operaciones de consulta, se puede considerar la definición de atributos derivados, que en UML se denotan con "/" como prefijo.

Cada representación de una cosa, evento o proceso perteneciente a una entidad, debe ser un elemento único e identificable de alguna manera: debe haber al menos un atributo o conjunto mínimo de ellos que identifique unívocamente al dato conformando lo que se llama un identificador de la entidad. Para descubrir un identificador, se toma un dato individual representativo de la misma, y se valida que el valor de algún atributo o la combinación de valores de un subconjunto mínimo de atributos, no se repita en el conjunto.

Los identificadores con un único atributo se clasifican como "simples", y si requieren dos o más atributos, como "compuestos". Todo atributo que integre un identificador debe ser obligatorio y monovalente, es decir, debe tener la cardinalidad por defecto, que es 1. Se propone como convención para denotar identificadores en UML, denotar los atributos que lo componen con el calificador {id}, o, para los simples, usar el calificador {unique}.

Una entidad puede tener más de un identificador, pero en UML sólo es posible denotar más de uno en el caso de que sean simples; entonces, a los efectos de denotar identificadores, se propone

Brazilian Journal of Development

como estándar denotar todos los que sean simples con el calificador {unique}, si hubiera más de uno, y en el caso de que hubiera sólo compuestos, denotar al más simple calificando a sus atributos componentes con {id}.

Hay datos que no se pueden identificar mediante un atributo ni un conjunto mínimo de atributos propios, porque su existencia depende de un dato de otra entidad; entonces la identificación de un dato individual es dependiente o relativa a la identificación del dato de la entidad de la que depende o, en caso de depender de más de una entidad, de los identificadores de un dato de cada una de dichas entidades. Los identificadores para estos datos se denominan **mixtos**, cuando intervienen atributos propios como discriminantes, o **externos**, cuando intervienen exclusivamente datos externos a la entidad dependiente. Cuando una entidad tiene un identificador mixto o externo, además de ser dependiente, se denomina "débil". Para poder denotar identificadores mixtos o externos, se propone estereotipar asociaciones como <id>.

En esta fase de diseño, no deben agregarse identificadores que no sean naturales en el sistema de información.

En la Figura 1 se ejemplifica la denotación de más de un identificador simple, de cardinalidades diversas de atributos, y de atributos compuestos. En la Figura 2, se proporciona un ejemplo de entidad débil y cómo denotar su identificador.

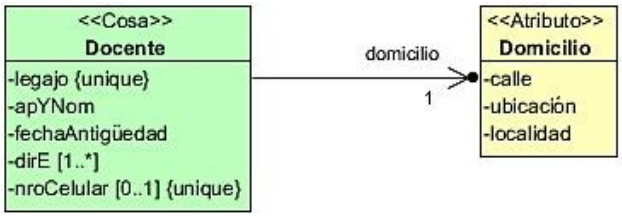

Figura 1. Identificadores simples y atributos opcionales, multivaluados y compuestos

Se utiliza como herramienta de modelado una edición comunitaria de Visual Paradigm, versión 15.0.

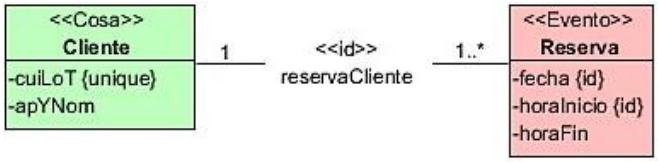

Figura 2. Identificador de entidad débil

3.2 CUALIDADES DE DIAGRAMAS

Los diagramas de entidades y relaciones producto del modelado conceptual involucran un proceso que además de sistemático y disciplinado, es esencialmente creativo, por lo que dependen en gran medida de la subjetividad y estilo de quien modela y de las cualidades que busque para sus diagramas: se puede buscar simplicidad, para una comprensión inmediata de la estructura estática

de un sistema, o se puede buscar expresividad, para representar en detalle propiedades y restricciones de los datos. Estas dos cualidades son incompatibles, ya que los modelos simples implican pasar a la especificación funcional del sistema las propiedades y restricciones no contempladas en el modelo estático, y los modelos muy expresivos, requerirán menos especificaciones funcionales, pero requerirán más esfuerzo para comprender la estructura estática de un sistema y distinguir entidades y relaciones importantes de otras que no lo sean.

Una cualidad siempre asequible para todo diagrama es la claridad, y para obtener diagramas claros se debe cuidar la disposición espacial de entidades y relaciones de manera que no se crucen líneas de relación y que las entidades que se relacionen se encuentren lo más próximas que sea posible en el diagrama; para esto, es posible que en el paso 1 o 2 del proceso de modelado, cuando la reproducción de diagramas no resulta demasiado costosa, sea necesario repetir intentos cambiando la disposición de las entidades. Asimismo, para obtener diagramas con esta cualidad también es necesario seguir criterios uniformes para nombrar entidades y relaciones.

3.3 MÉTRICAS PARA EVALUAR DIAGRAMAS

Las métricas de calidad implican la determinación de pesos de entidades, en función de atributos, relaciones y participación en jerarquías, y de pesos de relaciones, en función de atributos y cardinalidades de entidades participantes. Su objetivo es evaluar diagramas en busca de entidades y relaciones con pesos alejados del promedio y revisar su pertinencia (si se pueden fusionar o partir entidades o remodelar relaciones para homogeneizar pesos). También se puede calcular pesos de diagramas para revisar su homogeneidad.

Las métricas de complejidad implican sumas ponderadas de pesos de entidades y relaciones de un modelo completo (o de sus diagramas) con el objeto de graduar el esfuerzo requerido para un modelo o asignarle valor económico.

4 EL PROCESO DE MODELADO CONCEPTUAL

El modelado conceptual es un proceso evolutivo incremental, en el que se va resolviendo la complejidad mediante refinamientos sucesivos y modularización. Para ilustrar el proceso, se define a continuación un caso de estudio.

4.1 CASO DE ESTUDIO

Una concesionaria de automóviles vende unidades 0Km de una única marca. Un automóvil tipo de la marca se distingue por su modelo, caracterizado por un nombre único y un año de salida, por una terminación, caracterizada también por un nombre único y cantidad de puertas, y por una

Brazilian Journal of Development

motorización, que se distingue según el combustible, la cilindrada y la transmisión, por el precio de lista y por varios colores opcionales. Distintos autos tipo pueden tener igual modelo, o igual terminación o igual motorización. La concesionaria tiene en exhibición o en depósito unidades de autos tipo que se identifican tanto por su número de chasis como por su número de motor, y se caracterizan además por su color, fecha de ingreso y su disponibilidad para la venta.

Las ventas se inician una fecha determinada con el encargo de un auto tipo de determinado color por parte de un cliente, quien debe pagar un adelanto, del cual se registra su monto, forma de pago, descripción y número de recibo entregado al cliente. De las ventas debe registrarse también monto total, el saldo a cobrar, el empleado responsable, y la unidad que se vende, que si estuviera en depósito o exhibición se asociaría inmediatamente a la venta y se marcaría como vendido, o de lo contrario se registraría un número de orden a fábrica, y cuando se recibiese, se registraría su ingreso y se asociaría a la venta.

Las ventas se concretan cuando el cliente cubre el saldo a pagar, por lo que se registra un número de factura. El saldo se puede cobrar con uno o más pagos de los que se registra su fecha, monto, forma, descripción, y número de recibo que los identifica. Una forma de pago puede implicar la entrega de un vehículo usado, en cuyo caso debe registrarse su marca, modelo, tipo, patente y año de patentamiento. Tanto de empleados como de clientes se registran datos personales e información de contacto, y de empleados en particular, la fecha de ingreso y una fecha y motivo de baja opcionales.

4.2 REFINAMIENTOS SUCESIVOS

Se recomienda seguir los siguientes pasos de refinamiento en forma evolutiva:

1. Individualizar y representar entidades y relaciones, determinando cardinalidades de las entidades en las relaciones, subconjuntos de especialización que resulten evidentes, y coberturas de jerarquías (Figura 3).

Brazilian Journal of Development

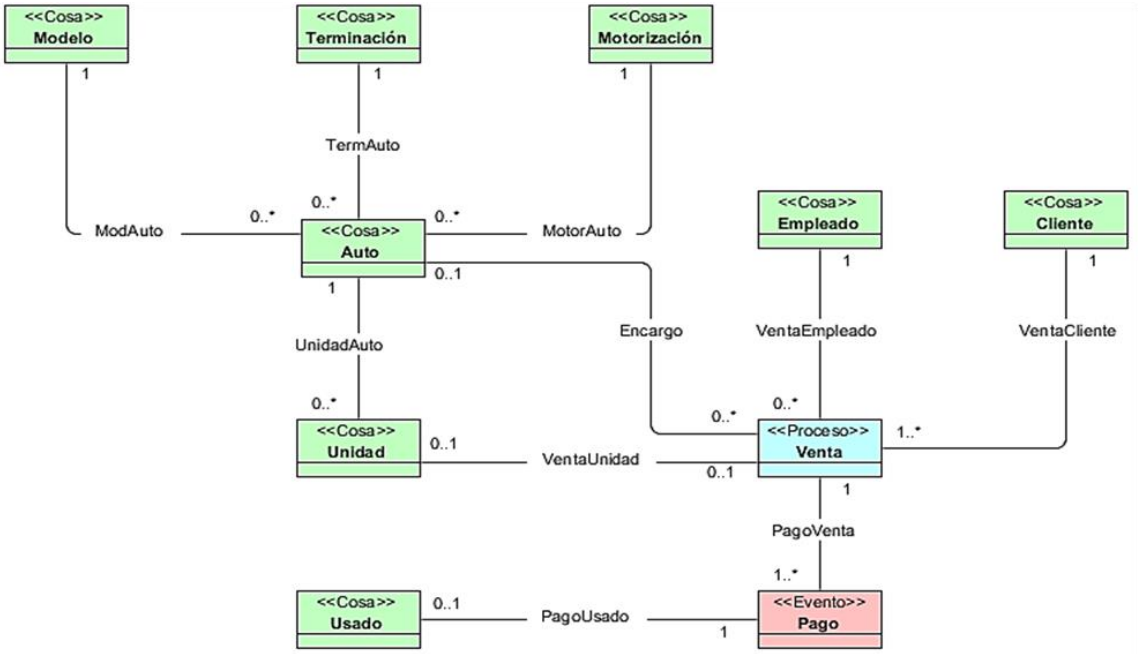

Figura 3. Clasificación de entidades y relaciones

2. Representar atributos que formen parte de identificadores, y verificar:
   - Que toda entidad tenga al menos un identificador, y que los subconjuntos de especialización hereden adecuadamente identificadores más allá de que puedan tener uno o más propios.
   - La consistencia de las cardinalidades de las entidades en las relaciones, cuidando que los vínculos que representen las relaciones se puedan identificar exclusivamente a partir de identificadores de las entidades.

3. Determinar los restantes atributos característicos de las entidades y de las relaciones que los requieran, denotando cardinalidades especiales (Figura 4), y analizar también
   - Si se detectan más identificadores (para completar el modelo).
   - Si se detectan más generalizaciones o especializaciones a partir del análisis de:
     - Atributos comunes en distintas entidades (para generalizar) o de atributos opcionales en una misma entidad (para especializar).
     - Entidades con más de una relación con cardinalidad mínima 0 (para especializar).
   - Si se puede agrupar atributos en atributos compuestos, si un subconjunto de atributos de una entidad representa una misma característica o propiedad (mejorar expresividad).

Brazilian Journal of Development

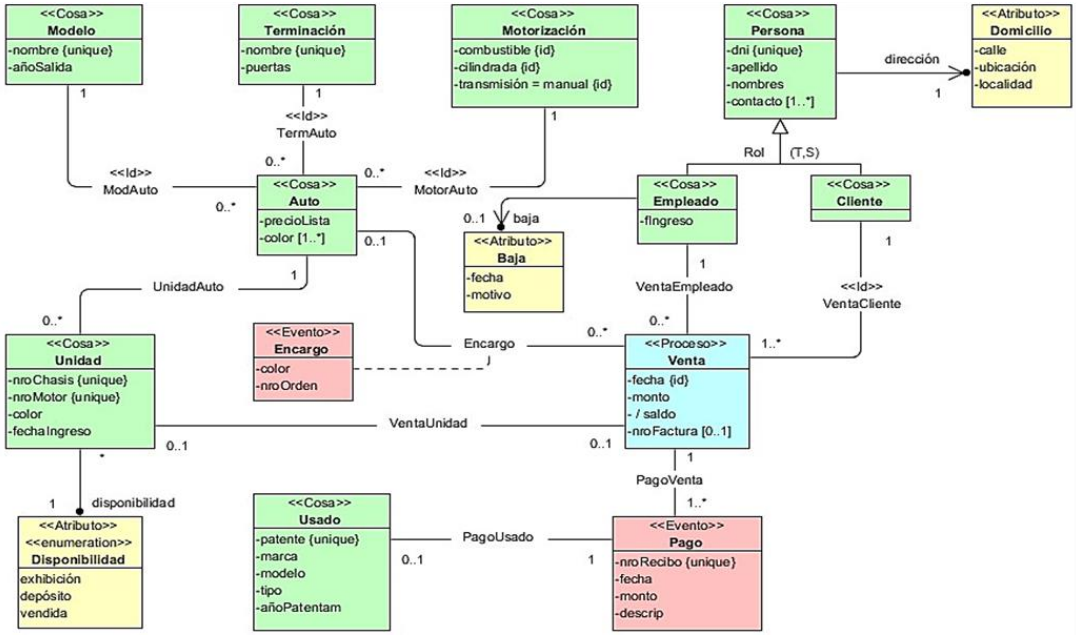

Figura 4. Análisis de entidades y relaciones

4. Evaluar la completitud y consistencia del diagrama obtenido:
   - Verificar que todas las entidades tengan al menos un identificador, aunque sea heredado, y que tengan todos los identificadores posibles.
   - Verificar que todas las entidades tengan denotada la cardinalidad en cada relación que participen.
   - Detectar y eliminar entidades colgantes: con sólo un atributo identificador y con una única relación con otra entidad.
   - Detectar y corregir redundancias: ninguna entidad puede tener como componentes atributos que sean propios de otra entidad existente en el modelo; esto sólo puede representarse mediante una relación con la otra entidad. Los únicos casos que no implican redundancia o que implican redundancia justificada son:
     - La copia de atributos de una entidad maestra a una relación transaccional o transacción dependiente de la misma (por ejemplo, el precio de productos o servicios: no hay redundancia porque el precio de un producto o servicio puede cambiar con el tiempo, pero el precio al que se vendió en una transacción no).
     - Los atributos derivados (por ejemplo, el total de una factura de venta: hay redundancia porque el precio puede obtenerse a partir de cantidades y precios de venta copiados al momento de facturar, pero puede justificarse si hay consultas frecuentes del total de facturas –redundancia justificada).

Brazilian Journal of Development

- Verificar que los subconjuntos de especialización de una jerarquía sean todos correspondientes a entidades del mismo tipo (todas deben representar conjuntos de cosas o conjuntos de eventos) y que no representen procesos en distinto estado (ya que esto significaría cambiar a un dato de conjunto cada vez que cambie su estado, y es funcionalmente ineficiente).
- Detectar y eliminar ciclos redundantes (entidades relacionadas conformando un ciclo tal que, por las cardinalidades de las relaciones, a partir de un dato de cualquiera de ellas se puede obtener el mismo dato transitivamente por las relaciones) quitándoles alguna relación.
- Para cada consulta u operación frecuente, listar entidades y relaciones involucradas en el orden en que deban recorrerse para resolverla, y constatar la consistencia del recorrido: que exista una relación, aunque sea transitiva, entre las entidades que se recorran y que las cardinalidades permitan obtener el resultado que se espera.

4.3 MODULARIZACIÓN

Cuando se enfrenta un sistema de información de gran complejidad, es decir, con gran número de entidades y relaciones, su abordaje suele efectuarse por divisiones propias del sistema o unidades de gestión, p. ej., una empresa puede estar organizada en departamentos o gerencias. Entonces el problema de modelar los datos de la empresa se descompone en el modelado de datos de cada departamento o gerencia, y si un departamento o gerencia resulta muy complejo por la cantidad de funciones u objetivos, se puede descomponer el problema de su modelado en el modelado de los datos involucrados en distintas funciones u objetivos. Como el resultado de esta descomposición será un conjunto de diagramas, posiblemente realizados por distintas personas, que pueden tener entidades y relaciones comunes (acoplamiento), se requiere un trabajo de integración consistente en:

- Detectar entidades sinónimas (con distinto nombre en distintos diagramas pero que representan al mismo conjunto de datos), y reducirlas a una única entidad con la unión de atributos (también con reducción de sinónimos), que servirá de acoplamiento entre los distintos diagramas que la compartan.
- Detectar relaciones sinónimas y también reducirlas.
- Detectar entidades homónimas (con el mismo nombre en distintos diagramas pero que representan distintos conjuntos de datos) y renombrar las necesarias para resolver el conflicto. Hacer lo mismo para relaciones homónimas.

Brazilian Journal of Development

5 CONCLUSIONES

Se puede comprobar la casi equivalencia expresiva del estándar de modelado conceptual de entidades y relaciones usando UML que se propone con el modelo de Chen. La única desventaja observada es la imposibilidad de denotar gráficamente más de un identificador para una entidad, cuando alguno de ellos es compuesto. Esta deficiencia se puede subsanar documentando de alguna forma las entidades, especificando identificadores adicionales al denotado en el diagrama donde aparece, de modo que sea posible exportar dicha información o visualizarla en conjunto al momento de derivar el modelo conceptual a un modelo lógico.

Por otra parte, la ventaja de adoptar este estándar radica en la unificación del lenguaje de modelado para el análisis de datos dentro de un proceso de desarrollo de software y el de modelado conceptual de datos para el diseño de bases de datos. Esto puede incidir positivamente para que los futuros profesionales se comprometan con un proceso de desarrollo que implique documentar efectivamente los sistemas, cuanto menos en la fase más crítica, que es la de análisis. A partir de un modelo de dominio de análisis, se pueden derivar en paralelo el modelo de diseño para un lenguaje de codificación orientado a objetos, y el modelo lógico relacional para implementar una base de datos.

REFERÊNCIAS

ArgoUML (2018) “Open source UML modeling tool”, http://argouml.tigris.org/.

Arlow, J. y Neustadt, I. (2005) “UML 2 and the Unified Process”, Addison-Wesley.

Batini, C., Ceri, S. y Navathe, S. (1994) “Diseño conceptual de bases de datos: un enfoque de entidades e interrelaciones”, Addison-Wesley Iberoamericana.

Beck, K. et al. (2001) “Manifesto for Agile Software Development”, www.agilemanifesto.org/.

Booch, G., Rumbaugh, J. y Jacobson, I. (2005) “The Unified Modeling Language User Guide”. 2a. ed., Addison-Wesley.

Debrauwer, L. y Van der Heyde, F. (2016) “UML 2.5. Iniciación, ejemplos y ejercicios corregidos”. 4a. ed., Ediciones ENI.

Chen, P. (1977) “The Entity-Relationship Approach to Logical Database Design”, QED Information Systems.

Connolly, T. y Begg, C. (2015) "Database Systems. A Practical Approach to Design, Implementation, and Management", Pearson Education Limited.

Enterprise Architect (2018) "UML modeling tools for Business, Software, Systems and Architecture", Sparx Systems, https://sparxsystems.com/.

Farve, L. (2003) "UML and the Unified Process", IRM Press.

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

Kroll, P. y Kruchten, P (2003) "The Rational Unified Process Made Easy", Addison-Wesley.

Lucidchart (2018) "Online Diagram Software & Visual Solutions", https://www.lucidchart.com/.

StarUML (2018) "StarUML 3. A sophisticated software modeler for agile and concise modeling", http://staruml.io/.

Visual Paradigm (2018) "Stay Competitive and Responsive to Change Faster & Better in the Digital World",https://www.visual-paradigm.com/.