LÉXICO ORIENTADO A MODELADO Y PATRONES:

UNA ESTRATEGIA PARA DERIVAR ELEMENTOS DEL MODELO CONCEPTUAL

Luis Oliverio Chaparro Lemus
Carmen Constanza Uribe Sandoval

Vigilada Mineducación
Universidad de Boyacá®
FACULTAD DE CIENCIAS E INGENIERÍA

Catalogación en la publicación – Biblioteca Nacional de Colombia

Chaparro Lemus, Luis Oliverio, autor
Léxico orientado a modelado y patrones: una estrategia para derivar elementos del modelo conceptual / Luis Oliverio Chaparro Lemus, Carmen Constanza Uribe Sandoval. -- Tunja: Universidad de Boyacá, 2023.
92 Páginas.

Incluye referencias bibliográficas.

ISBN 978-958-5120-42-6 -- 978-958-5120-47-1 (digital)

1. Arquitectura de software dirigida por modelos - Investigaciones 2. Patrones de diseño de software - Investigaciones 3. Desarrollo de software 4. Ingeniería de software I. Uribe Sandoval, Carmen Constanza, autora

CDD: 005.1 ed. 23 CO-BoBN– a1118943

LÉXICO ORIENTADO A MODELADO Y PATRONES:

UNA ESTRATEGIA PARA DERIVAR ELEMENTOS DEL MODELO CONCEPTUAL

Luis Oliverio Chaparro Lemus
Carmen Constanza Uribe Sandoval

Vigilada Mineducación
Universidad de Boyacá®
FACULTAD DE CIENCIAS E INGENIERÍA

Luis Oliverio Chaparro Lemus
Carmen Constanza Uribe Sandoval

Gestión editorial, diseño y diagramación
División de Publicaciones

Director División de Publicaciones
Ing. D.G. Mg. Johan Camilo Agudelo Solano

Diseño y diagramación
D.G. Rafael Alberto Cárdenas Estupiñan

Corrección de texto y estilo
Lit. Mg. Diva Marcela Piamba Tulcán

© **Ediciones Universidad de Boyacá**

Carrera 2ª. Este N° 64-169
Tels.: (8) 7452742 - 7450000 Ext. 5405
www.uniboyaca.edu.co
publicaciones@uniboyaca.edu.co

Tunja-Boyacá-Colombia

ISBN Físico: 978-958-5120-42-6
ISBN Digital: 978-958-5120-47-1

DOI: 10.24267/9789585120426

Resumen

La Arquitectura Dirigida por Modelos (MDA) y el Desarrollo de Software Dirigido por Modelos (MDSD) son iniciativas de gran importancia y proyección para el desarrollo de software. Sus principios combinados han generado herramientas de ciclo completo de desarrollo de software más potentes que las de desarrollo asistido por computadora. MDA es un conjunto de directrices para definir un proceso de desarrollo de software basado en modelos. MDSD es una tendencia de desarrollo de software que privilegia el uso de modelos y las transformaciones entre modelos, y se centra en construir técnicas, métodos y herramientas tecnológicas que hacen posible la generación de software con base en la definición de modelos.

Uno de los tres modelos descritos por MDA es el Modelo Independiente de la Computación (CIM), el cual describe el sistema desde el punto de vista de los expertos del dominio del problema, de forma que representa los requisitos del sistema. Muchos de los investigadores en desarrollo de software dirigido por modelos han dado poca importancia al CIM y se han concentrado en el modelo independiente de la plataforma y en las transformaciones que conducen a obtener el modelo específico de la plataforma. Con el fin de subsanar esta situación, en este libro se presentan las bases teóricas y conceptuales que permiten definir un modelo independiente de la computación, partiendo de la definición textual del sistema y de sus requisitos funcionales, y se definen los criterios que, a partir del CIM, conducen a obtener un Modelo Independiente de la Plataforma (PIM). Dado que para sentar las bases de construcción del CIM se ha tomado como referencia el método formal de modelado OO-Method, también se hace una descripción de OO-Method y de OLIVANOVA y se presentan los fundamentos teóricos de MDA, haciendo énfasis en el CIM.

Palabras Claves

Léxico orientado al modelado; arquitectura orientada al modelado; desarrollo de software conducido por modelos; transformaciones entre modelos; patrón de modelado; lenguaje acotado.

Abstract

Model Driven Architecture (MDA) and Model Driven Software Development (MDSD) are initiatives of great importance and projection for software development. Their combined principles have given rise to early full-cycle software development tools more powerful than outperform computer-aided development tools. MDA is a set of guidelines to define a model-based software development process. MDSD is a software development trend that favors the use of models and transformations between models and focuses on building techniques, methods and technological tools that make it possible to generate software based on the definition of models.

One of the three models described by MDA is the Computational Independent Model (CIM), which describes the system from the point of view of the experts in the problem domain in such a way that it represents the system requirements. Many of the researchers in model-driven software development have given little importance to the CIM and have concentrated on the platform-independent model and on the transformations that lead to obtaining the platform-specific model. In order to correct this situation, this book presents the theoretical and conceptual bases that allow defining an independent model of computation, starting from the textual definition of the system and its functional requirements, as well as defining the criteria that, from the CIM, lead to obtaining a Platform Independent Model (PIM). Given that the formal OO-Method modeling method has been taken as a reference to lay the foundations for the construction of the CIM, a description of both OO-Method and OLIVANOVA is also made and the theoretical foundations of MDA are presented in this book, emphasizing the CIM.

Keywords

Model Driven Lexicon; Model-Driven Architecture; Model-Driven Software Development; Transformations Between Models; Modeling Pattern; Bounded Language.

Presentación

El desarrollo de las tecnologías de la información y de la comunicaciones son un pilar clave del desarrollo sostenible. De acuerdo con la Agenda para el Desarrollo Sostenible, estas tecnologías son determinantes en la promoción de procesos de industrialización y de innovación sostenibles. Las Instituciones de Educación Superior son actores fundamentales en la consecución de estos propósitos, tanto más en el avance de la tecnología de cara a los desafíos que enfrenta el planeta social, económica y ambientalmente.

En coherencia con ello, nuestra Universidad ha proyectado su quehacer desde la visión de "La Universidad del futuro", como una reflexión colectiva que orienta su actuar a alcanzar, entre otros, objetivos tales como la implementación de la Universidad 4.0. Sin duda, este propósito está llamado a asentarse en funciones como la investigación, específicamente alrededor de aspectos como las tecnologías de la información y de la comunicación. "Léxico orientado a modelado y patrones: una estrategia para derivar elementos del modelo conceptual" es precisamente el resultado de los ejercicios de investigación a través de los cuales la Facultad de Ciencias e Ingeniería de la Universidad de Boyacá procura el avance del estado del conocimiento en las tecnologías de la comunicación y de la información.

Esta publicación hace un recorrido conceptual por una cuestión que cobra cada vez más fuerza: el desarrollo de software; dado el contexto de transformaciones que suponen tecnologías tales como la inteligencia artificial. Puntualmente, dos modelos de desarrollo de software ocupan los hallazgos a exponerse en las siguientes páginas: la Arquitectura Dirigida por Modelos (MDA) y el Desarrollo de Software Dirigido por Modelos (MDSD). No obstante, con un enfoque en particular, el Modelo Independiente de la Computación (CIM), un modelo del que se aprecian escasas contribuciones en la literatura.

La Universidad de Boyacá se complace en presentar a estudiantes, docentes, profesionales y comunidad en general, un recurso que no sólo permite un acercamiento conceptual al desarrollo de software, sino también una herramienta metodológica para su adopción. Igualmente, extendemos un saludo de reconocimiento a los autores, destacando no sólo la contribución que hace en sí misma esta publicación, sino el potencial que representa para los procesos de enseñanza en las aulas y todos aquellos que se surtan en los diferentes espacios a los que esperamos llegue a través de las manos del lector.

ANDRÉS CORREAL CUERVO

RECTOR

Contenido

Introducción

El desarrollo de software basado en la construcción de modelos es un conjunto de principios teórico-prácticos que busca potencializar en el ingeniero de software las habilidades fundamentales para el análisis y el diseño de software. Con este propósito, el Grupo de Gestión de Objetos (OMG) estableció un conjunto de directrices denominado Arquitectura Guiada por Modelos (MDA). De otro lado, el enfoque denominado Desarrollo de Software Guiado por Modelos (MDSD) se ocupa principalmente de los asuntos tecnológicos que conducen al desarrollo de software mediante modelos, para lo que trabaja con investigadores y profesionales de la ingeniería de software que se ocupan de formular técnicas, procedimientos y herramientas informáticas orientadas al desarrollo de software bajo este enfoque, como lo explica Molina y Pastor (2004).

MDA propone la representación del sistema en tres modelos diferenciados por su nivel de abstracción que, ordenados de mayor a menor nivel de abstracción, son: el Modelo Independiente de la Computación (CIM), el Modelo Independiente de la Plataforma (PIM) y el Modelo Específico de la plataforma (PSM) (Chaparro y Gómez, 2012). El CIM, que constituye el interés central de este libro, es el modelo de mayor abstracción, pues considera el estudio del sistema tal como funciona en su entorno real. Sin embargo, en la comunidad MDSD son pocos los trabajos que se ocupan del CIM, pues la mayor atención de investigación se ha centrado en la conformación de PIM y en las transformaciones que conducen a obtener el PSM. Para ayudar a solventar esta situación, en este libro se presenta un conjunto de patrones obtenidos a partir del Léxico Orientado al Modelado (LOM), como una técnica (Chaparro, 2009) que persigue sentar las bases teórico-conceptuales para definir un CIM, partiendo de la definición textual del sistema y de sus requisitos funcionales, además de definir las correspondencias para obtener un PIM.

Al respecto, dos problemas fundamentales se presentan cuando se quiere construir un CIM: cómo establecer los requisitos reales del sistema (lo que el cliente necesita del sistema) y cómo representar tales requisitos como parte de un modelo independiente de la perspectiva computacional. Entonces, para afrontar el primer problema, en este libro se presenta el LOM, que es una estrategia gramatical basada en el léxico del dominio y en la acotación del lenguaje que describe los términos del léxico. Para atender el segundo problema se ofrecen un conjunto de patrones y un procedimiento para obtener, por ejemplo, el diagrama de clases y el diagrama de transición de estados.

La precisión de los requisitos del sistema depende prioritariamente de que el ingeniero de software y los interesados (***stakeholders***) establezcan una comunicación de calidad. Esta se puede mejorar estableciendo un lenguaje común que contenga los términos relevantes del dominio del problema (Chaparro, 2009). Este lenguaje debe ser factible de refinar mediante el LOM para identificar elementos del CIM en un entorno orientado a objetos.

Aquí se presentan las bases teóricas y prácticas para formalizar y automatizar la construcción del CIM, el LOM, los patrones de requisitos, una taxonomía de ítems de términos LOM y el lenguaje natural acotado, hacen parte de los resultados visibles de un conjunto de actividades de investigación realizadas durante varios años. Adicionalmente, las aplicaciones prácticas de la investigación y las teorías mencionadas se representarán en una herramienta MDA que es objeto de un trabajo de investigación complementario.

Las ideas que se exponen en los próximos capítulos, además de hacer un aporte al estado del conocimiento, permitirán crear conciencia en la comunidad de ingeniería de software sobre la importancia de formular procesos de desarrollo de software sistemáticos y centrados en los modelos. Así, se ponen a disposición del ingeniero de software herramientas conceptuales y de trabajo, y se le alienta para que ponga en práctica algunas de sus responsabilidades centrales: el análisis, el diseño y el control sobre las soluciones proyectadas, actividades que conforman los pilares de la ingeniería de software.

Este libro consta de nueve capítulos. En el primer capítulo se presentan los elementos que motivaron la investigación. En el segundo capítulo se presenta un marco teórico conceptual centrado en MDA y MDSD. En el tercer capítulo se describen las primitivas o elementos que fundamentan todo el trabajo. El cuarto capítulo, que constituye uno de los capítulos centrales, presenta el Léxico Orientado al Modelado. En el quinto capítulo se presenta una descripción OO-METHOD y OLIVANOVA que permite encausar el trabajo y determinar los elementos de modelado que se pueden obtener del LOM. El capítulo seis describe los patrones de modelado, estrategia que permite identificar los elementos de modelado con base en el LOM. En el capítulo siete se presenta un proceso, paso a paso, previsto para generar el modelo conceptual de OO-METHOD/OLIVANOVA. En el capítulo ocho se presentan los trabajos relacionados que anteceden o son paralelos al presente trabajo y, finalmente, en el capítulo nueve se presentan las consideraciones finales que permiten dar una conclusión al trabajo socializado y ofrecen un panorama para las investigaciones que permitan seguir fortaleciendo el área de conocimiento aquí presentada.

1. Directrices que Motivaron la Investigación

Aunque el Modelo Independiente de la Computación (CIM) es vital para el proceso de desarrollo de software dirigido por modelos, la comunidad investigadora no le ha prestado la atención que merece a su desarrollo y a la conformación de herramientas de apoyo. La relevancia del CIM radica en que este modelo le permite al ingeniero de software entender el sistema que se va a construir, partiendo del sistema existente (sea manual o no). Este modelo favorece el conocimiento de dominio del problema y permite establecer un lenguaje de común entendimiento entre los usuarios y el ingeniero de software.

En este libro se presentan las bases para conformar un CIM a partir de la descripción del sistema y de sus requisitos funcionales en lenguaje natural, abordando los siguientes interrogantes: cómo establecer los requisitos funcionales reales y cómo representar tales requisitos en un modelo independiente de la perspectiva computacional (CIM).

Para resolver este asunto se plantearon las siguientes tres hipótesis, complementarias entre sí:

- Una técnica gramatical centrada en la identificación de términos relevantes del sistema (LOM) ayuda efectivamente a determinar los requisitos funcionales.

- Los refinamientos de tales términos conducen a la identificación de elementos de un modelo conceptual inicial (que en este caso tiene que ver con un CIM).

- El modelo así obtenido constituye la base para generar el modelo Conceptual o PIM de OO-METHOD.

En este sentido, la idea fue conformar un modelo gramatical de términos relevantes del sistema, que permitiera derivar parcialmente un CIM para OO-Method. Esto incluye un método gramatical de análisis de requisitos, con base en los términos relevantes extraídos de requisitos funcionales descritos en lenguaje natural.

La conformación del modelo y su método gramatical implicó formular el LOM, es decir, formular sus principios teóricos y sus estrategias de análisis, para estructurarlo; también, determinar los elementos del Modelo Conceptual (PIM de OO-Method) que se pueden obtener con apoyo del LOM; formular un conjunto de patrones gramaticales que, a partir del LOM, ayudaran a identificar potenciales elementos del Modelo Conceptual de OO-Method; proponer un proceso que condujera a generar parte del modelo conceptual a partir de la descripción del dominio

del negocio y de los requisitos funcionales de usuario (incluidas las reglas generales de transformación CIM-PIM); y especificar un lenguaje acotado que permitiera describir los términos relevantes del sistema de manera precisa.

2. Focalización Teórica

Los siguientes son los fundamentos teóricos conceptuales.

2.1 Abstracción

Proceso de separar las características más relevantes de un dominio específico de acuerdo con un determinado enfoque. El nivel de abstracción puede variar desde niveles altos hasta niveles bajos. Entre mayor sea el nivel de abstracción, mayor será la cantidad de detalles que ocultan; y entre más bajo sea este nivel, mayor será la cantidad de detalles que se muestran.

2.2 Modelo

Es un "conjunto de declaraciones acerca de un sistema bajo estudio" (Seidewitz, 2003, p. 27). Un modelo se describe mediante un formalismo o un lenguaje que define su sintaxis o notación y su semántica o su significado. Un modelo abstrae la realidad mediante uno o más de los mecanismos de reducción, de clasificación o de generalización (Metzger, 2005), así:

- Abstracción por reducción: selecciona las propiedades relevantes y desecha las propiedades irrelevantes.

- Abstracción por clasificación: basada en un proceso de identificación de tipos o conceptos. Esta es la abstracción básica del modelado orientado a objetos.

- Abstracción por generalización: ignora las diferencias entre elementos similares para conformar una entidad que enfatiza en las similitudes.

El modelo gramatical de términos relevantes del sistema (CIM) que aquí se propone está conformado por un conjunto de términos que describen los elementos más relevantes de un sistema de información según el análisis y las relaciones entre tales elementos. Tanto los términos como sus relaciones, que son declaraciones que describen la estructura global del sistema, se pueden llegar a representar mediante un diagrama de clases (DC) preliminar. Los términos se describen utilizando

plantillas preestablecidas que son equiparables al formalismo o al lenguaje usado para implementar la descripción de un sistema.

La estructura sintáctica se visualiza mediante las plantillas de especificación de los términos, y la configuración de los ítems de definición de términos se implementa mediante un lenguaje acotado.

En la construcción del LOM se utilizan dos estrategias de abstracción: reducción, para identificar los elementos relevantes, y clasificación, para agrupar los términos y los nexos en sus respectivos tipos. También se utiliza la clasificación cuando se definen las reglas (patrones y heurísticas) para obtener el diagrama de clases y, posiblemente, el diagrama de estados.

2.3 Sistema

Conjunto de elementos que operan de manera coordinada y cooperativa para lograr una meta. En el caso de los sistemas de información, estos están conformados por elementos de información activos y pasivos en los que los elementos activos inciden sobre los elementos pasivos para conseguir las metas.

2.4 Transformación

Proceso cuyo propósito es convertir entidades de un determinado tipo en entidades de otro tipo preestablecido. Si la transformación es vertical, el nivel de abstracción de las entidades varía. Si la transformación es horizontal, se mantiene el nivel de abstracción, pero varía la perfección de su descripción. La transformación también puede tener el propósito de cambiar su representación, en cuyo caso lo que cambia es la notación, es decir, los elementos sintácticos.

3. Arquitectura y Desarrollo Dirigidos por Modelos

3.1 Arquitectura Dirigida por Modelos (MDA)

Los lenguajes de desarrollo de software han tendido a elevar su abstracción desde su creación. Los lenguajes cercanos al lenguaje de máquina (binario y ensamblador) evolucionaron en lenguajes de tercera generación, con estructuras lingüísticas próximas

al lenguaje humano; luego, los entornos de desarrollo adicionaron facilidades de edición, compilación y depuración lógica. Mientras tanto, los sistemas de gestión de bases de datos crearon SQL, un lenguaje de cuarta generación para administrar los sistemas de bases de datos. Se espera que el siguiente escalón ascendente en el nivel de abstracción se enfoque en el modelado del sistema. Los modelos del sistema generan elementos de construcción de software que se aproximan a la manera de pensar de los ingenieros de software y ocultan detalles tecnológicos innecesarios acerca de los actuales lenguajes de programación.

El fundamento teórico-práctico para este nuevo nivel de abstracción lo proponen dos grandes áreas de investigación: MDA y MDSD. MDA establece los lineamientos teóricos y conceptuales, mientras que MDSD establece las acciones prácticas para desarrollar el modelamiento, usando como guía los modelos (Brown et al., 2005); es decir, MDA es la base de MDSD.

Los aspectos y principios más sobresalientes de MDA que han sido recopilados por Chaparro y Gómez (2012) son:

- Cambia el foco de desarrollo de software, de la escritura de código a la construcción de modelos.

- El asunto principal de MDA es separar la especificación de la funcionalidad del sistema, de los detalles de su implementación en una plataforma específica.

- Su principal meta es lograr portabilidad, interoperabilidad y reutilización, mediante la separación de los aspectos de arquitectura.

- Considera que todos los artefactos son modelos, es decir especificación de requisitos, las descripciones de arquitectura, las descripciones de diseño y el código.

- Es un enfoque conducido por modelos que guían el curso del entendimiento, el diseño, la construcción, el despliegue, la operación, el mantenimiento y la modificación.

- La transformación automática es una de las nociones centrales de MDA. Ella describe cómo un modelo en un lenguaje fuente (modelo fuente) se puede transformar en uno o más modelos en un lenguaje objeto (modelo objeto).

- Los modelos bien definidos (sintáctica y semánticamente) son el aspecto más relevante para comprender el sistema y para implementar soluciones empresariales escalables.

- La construcción de un sistema se puede hacer mediante la elaboración de modelos, las transformaciones entre modelos y la organización del trabajo en una arquitectura de niveles y transformaciones.

- Los metamodelos constituyen la estructura formal que facilita la transformación y la integración entre modelos, y son fundamentales para la automatización mediante herramientas.

- Para aceptar y adoptar el desarrollo centrado en modelos se requiere la formulación de estándares industriales para brindar facilidades a los clientes y habilitar una clara competencia entre vendedores.

- Uno de los aspectos centrales del marco de trabajo de MDA es la noción de transformación automática. Esta especifica cómo transformar un modelo escrito en un lenguaje fuente (modelo fuente) a un modelo en lenguaje objeto (modelo objeto).

Como ya se había introducido, MDA propone la representación del sistema en tres modelos diferenciados por su nivel de abstracción. Estos se describen en Debnath et al. (2008) de la siguiente manera:

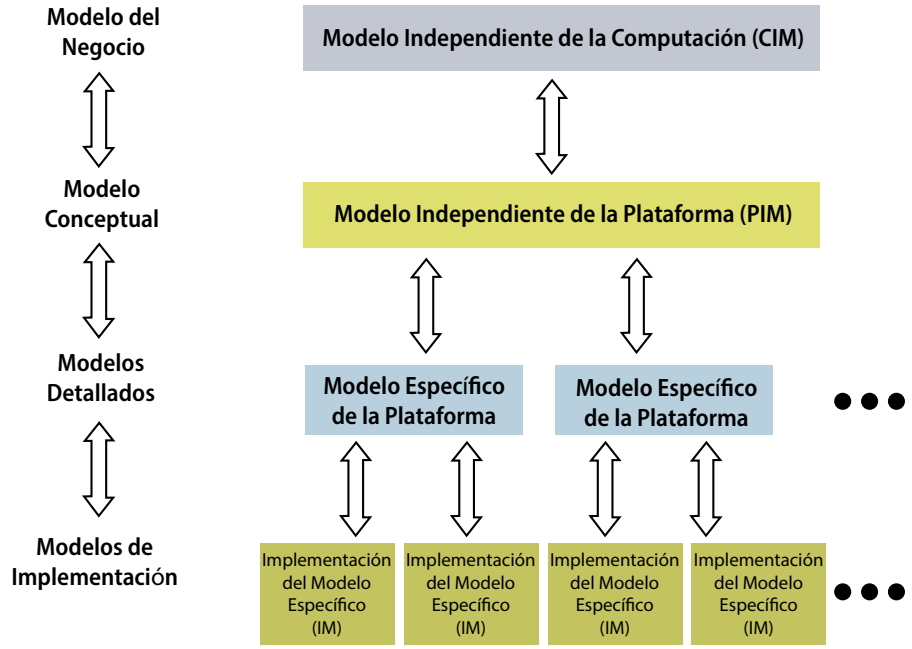
- Modelo Independiente de la Computación (CIM). Este modelo oculta los detalles de la estructura del sistema software. Dado que usa un lenguaje cercano a los profesionales del dominio que se esté tratando (es decir, quienes conocen el negocio), también se le denomina Modelo del Dominio.

- Modelo Independiente de la Plataforma (PIM). Como su nombre lo indica, es un modelo que proporciona una vista del sistema independiente de la plataforma, de forma que es el ideal para usarse en varias de ellas.

- Modelo Específico de Plataforma (PSM). Es una vista del sistema que, junto con las especificaciones del PIM, describe cómo el sistema usará una plataforma determinada.

Los niveles de abstracción de estos modelos se representan en la figura 1.

Figura 1. Modelos por niveles y transformaciones MDA



Nota: Adaptado de Brown et al. (2005).

Debnath et al. (2008) proponen los siguientes pasos para el desarrollo de una MDA:

- Se captura el conocimiento y los requisitos del negocio en un CIM.

- Los desarrolladores crean el modelo del sistema que satisface los requisitos del negocio, es decir, el PIM que no tiene en cuenta una plataforma tecnológica específica.

- Un entorno de desarrollo conducido por modelos (MDSD) transforma el PIM en uno o más PSM, mediante un conjunto de reglas.

- La obtención del código se hace desde el PSM al utilizar un segundo conjunto de reglas de transformación.

Aunque en estas etapas la transformación del CIM en un PIM se hace de forma manual, es necesario establecer un procedimiento que, cuando menos de manera semiautomática, ayude a construir el PIM a partir del CIM, con la ayuda de un entorno MDSD. Este sería un conjunto de reglas que orienten esa transformación y un lenguaje de alto nivel que las implemente en los modelos.

El éxito de MDA, entonces, depende de la definición de lenguajes de escritura de modelos y de herramientas que impacten significativamente en el proceso de ingeniería hacia adelante.

3.2 Desarrollo de Software Dirigido por Modelos - MDSD

Para comenzar, es preciso destacar que MDSD formula recomendaciones generales y flexibles para construir herramientas basadas en modelos, de manera que cada herramienta es libre de seguir las directrices que considere convenientes. De otro lado, MDSD prefiere que no se la identifique con MDA, inclusive, algunas tendencias de MDSD afirman que MDA es una iniciativa de estandarización de OMG inspirado en MDSD y que MDSD es independiente de la madurez de los estándares MDA (Brown et al., 2005). De todas maneras, el MDSD se constituye en un hito en la evolución de las prácticas de programación y desarrollo. Solo hay que dar un vistazo a la evolución de la manera de programar para darse cuenta de que se ha caracterizado por el continuo aumento del nivel de abstracción de los lenguajes de programación. Se inició con la codificación binaria de unos y ceros, que evolucionó a los lenguajes ensambladores; luego la programación con lenguajes ensambladores dio lugar a la programación con lenguajes de alto nivel en ambientes textuales; y de estos últimos se evolucionó a los ambientes de programación integrados y a los 4GL. Actualmente, con el advenimiento del Lenguaje Unificado de Modelado - UML y MDA, el paso natural de la evolución es la programación usando lenguajes de modelado.

El papel de los modelos en la construcción de software se puede percibir desde dos perspectivas diferentes: desarrollo de software ***basado en modelos*** y desarrollo de software ***dirigido por modelos***. El primero utiliza los modelos como una concepción del dominio del sistema, mientras que para el segundo los modelos tienen un rol central activo, tan importante como el código fuente. Aquí los modelos se van transformando hasta convertirse en el código del sistema (Pons et al., 2010).

Chaparro y Gómez (2012) resumen las implicaciones de los términos "basado" y "dirigido", explicando que, dado que los sistemas son dinámicos, especialmente en las primeras etapas del ciclo de vida, la documentación se convierte en una tarea compleja que necesita ser ***adaptada continuamente***, lo que puede hacerla inconsistente. Cuando los modelos se aplican solo como documentación, se reduce su efecto a interpretar el dominio del sistema con el fin de apoyar la obtención manual del código. En contraste, en el desarrollo dirigido por modelos, el modelo representa código de alto nivel, que después de algunas transformaciones automáticas genera código fuente en uno de los lenguajes de programación de tercera generación.

MDSD propone, en primer lugar, definir procesos de producción automática y, en segundo lugar, establecer abstracciones específicas del dominio para facilitar dicha automatización con alta productividad, calidad y facilidad de mantenimiento. Debido a esto, se facilita la comunicación entre el ingeniero de software y los expertos del dominio.

Una condición inicial para el desarrollo conducido por modelos es formular un "modelo específico del dominio". En esta dirección, Chaparro y Gómez (2012) presentan los siguientes requerimientos para aplicar exitosamente el concepto de "modelo específico del dominio":

- Lenguajes específicos del dominio que permitan la formulación de modelos reales.

- Lenguajes que puedan expresar las transformaciones requeridas de modelos a código.

- Compiladores, generadores o transformadores que puedan ejecutar las transformaciones para generar código ejecutable en diversas plataformas.

De esta manera, MDSD establece que la tarea de construir modelos no debe quedar reducida solo a la documentación del sistema, sino que debe ser parte integral del código, de manera que sea posible generar código fuente en lenguajes de programación tradicionales.

Finalmente, cabe aclarar que, por lo general, estos modelos no son exclusivamente gráficos, sino que se requiere que algunas de sus partes se expresen en forma textual y que el modelo, en su conjunto, se pueda trasladar a código fuente en lenguajes de programación.

3.3 Modelo independiente de la computación (CIM)

Ya se ha dicho que el CIM es el modelo de mayor abstracción de MDA, pues hace una representación desde el punto de vista del trabajador del conocimiento, ya que oculta los detalles relacionados con la solución. Este es un modelo independiente de la solución.

El CIM modela los requisitos del sistema y representa al sistema en el entorno en el que funcionará, de manera que ayuda a describir lo que se espera que el sistema

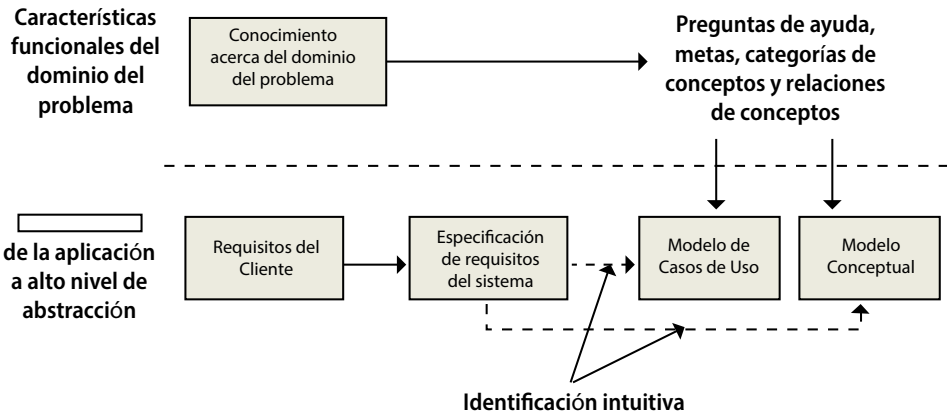
haga. Su utilidad se fundamenta en que, además de ayudar a entender el problema, es también la fuente del vocabulario que se comparte con los demás modelos. El CIM debe ser trazable con el PIM y el PSM (Object Management Group - OMG., 2003, 2014). Además, el CIM puede estar conformado por varios modelos (diagramas) que describen los requisitos del sistema, como el modelo de procesos de negocio, el modelo de objetos y el modelo del entorno en el que operará el sistema. La especificación de requisitos se puede hacer en UML o en cualquier otro lenguaje de modelado, para lo cual se utiliza un lenguaje propio de los especialistas en el dominio del negocio (Osis et al., 2007). Por esto, al CIM se le considera con frecuencia un modelo del dominio o modelo del negocio.

En MDA los modelos se construyen utilizando diversos niveles de abstracción y refinamiento, de forma que tales modelos se puedan obtener por transformaciones: PIM-a-PIM, PIM-a-PSM, PSM-a-PSM y PSM-a-PIM. Sin embargo, se ha subestimado la importancia del CIM, la trazabilidad en su interior y la trazabilidad entre el CIM y el PIM. Esta situación no favorece la disminución de la brecha entre los dominios del negocio y de la aplicación (Osis et al., 2007).

Ahora bien, Osis et al. (2007) afirman que en el desarrollo de software existen dos aspectos fundamentales: el ***análisis***, que define lo que una aplicación tiene que hacer con un dominio de problema para satisfacer los requisitos del cliente, y el ***diseño***, que define cómo debe ser construida la aplicación. El análisis implica examinar la situación actual del sistema del cliente en el mayor detalle posible y recolectar los requisitos del sistema que se planea construir. El análisis de requisitos busca averiguar lo que el cliente quiere del software, pero es preciso determinar lo que realmente el cliente necesita de él (en la mayoría de los casos son diferentes). Para asegurarlo, los requisitos deben estar en concordancia con las metas del cliente y con el medio en el que el software debe funcionar.

Una estrategia utilizada actualmente para construir el CIM, y que concreta la anterior afirmación, se muestra en la figura 2. Esta implica determinar las características funcionales del dominio del problema y las características de la aplicación en un alto nivel de abstracción. Las características funcionales se obtienen del conocimiento del dominio del problema que poseen los especialistas y se representan mediante el modelo de casos de uso y el modelo conceptual. Esto se puede hacer con ayuda de una o más de las siguientes estrategias: preguntas de ayuda, metas, categorías de conceptos y relaciones de conceptos. Para establecer las características de la aplicación es necesario recolectar los requisitos del cliente, especificarlos y representarlos mediante el modelo de casos de uso y el modelo conceptual.

Figura 2. Estado de la creación del CIM en análisis OO



Nota: Adaptado de Osis et al. (2007)

En conclusión, el CIM, además de ser el modelo del sistema de mayor nivel de abstracción, posee una gran influencia en el éxito del desarrollo del sistema. De este modelo depende que el desarrollador haga una interpretación correcta del sistema que se quiere desarrollar. Sin embargo, el desarrollador debe ser muy objetivo, pues puede encontrar dificultades que se deben a conflictos de intereses entre los usuarios y, en muchos casos, a que ellos mismos pueden no saber a ciencia cierta lo que quieren del sistema. En tal caso, el modelo de negocio y el modelo de requisitos son una herramienta que puede ayudar a los usuarios indecisos a esclarecer sus dudas y colaborar efectivamente en determinar lo que necesitan del sistema, por encima de lo que ellos quisieran. En este propósito puede ayudar mucho la determinación de las metas de negocio para cada usuario y el modelo que describa cómo operará el sistema en el entorno real de trabajo.

3.4 Transformaciones

La transformación de modelo y el modelo constituyen el núcleo de MDA. Así mismo, tienen gran importancia en el desarrollo del presente escrito. En forma general, en el contexto de MDA y MDSD, una transformación, como lo presenta Metzger (2005), se puede percibir como una 4-tupla de la forma <MF, MO, R, T>, en la que:

- MF es el modelo fuente que se quiere transformar,

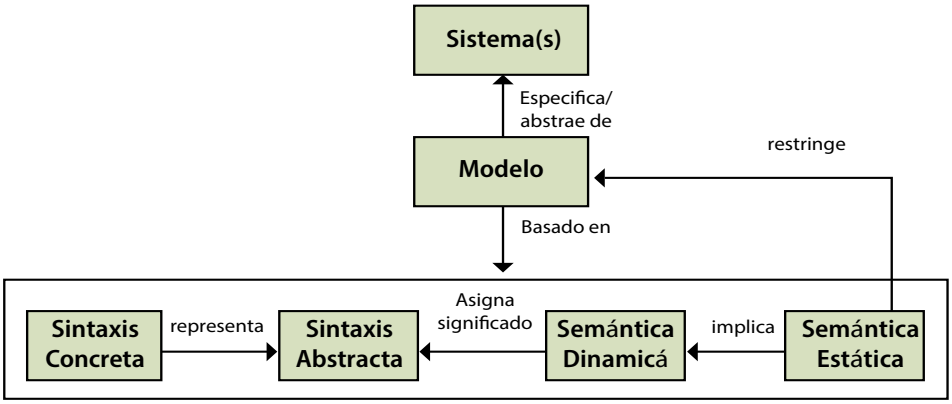
- MO es el modelo objeto resultado de la transformación,

- R es un conjunto de reglas que definen las correspondencias entre MF y MO,

- T es una operación que transforma especificaciones de MF en MO, usando las reglas de transformación.

Sin embargo, para describir más formal y precisamente el concepto de transformación, es necesario tener claros los conceptos de sistema, modelo y formalismo. Un *sistema* de información está conformado por diversos artefactos (elementos) de información del negocio que trabajan colaborativamente para lograr un objetivo empresarial; mientras que "un *modelo* es un conjunto de declaraciones de un cierto *sistema bajo estudio*" (Seidewitz, 2003, pág. 27). Cada tipo de modelo abstrae características de un cierto tipo, que les permiten a sus usuarios concentrar la atención en determinados aspectos significativos del sistema. Cada modelo se basa en un *formalismo* (o lenguaje) que define la sintaxis del modelo (o notación) y su semántica (o significado). Como se muestra en la figura 3, la sintaxis de un formalismo se compone de la sintaxis concreta y la sintaxis abstracta. La sintaxis concreta especifica la representación legible de los elementos abstractos de notación, puede ser estática o dinámica y representa la sintaxis abstracta (Metzger, 2005).

Figura 3. Relación entre sistema, modelo y formalismo



Nota: Adaptado de Metzger (2005).

La semántica estática está conformada por un conjunto de reglas bien formadas que representan las restricciones aplicables a una colección de modelos válidos y está implicada en la semántica dinámica, como se ve en la figura 3. Estas se pueden expresar usando el formalismo subyacente (Metzger, 2005).

Las definiciones de sistema, modelo y formalismo son el fundamento para describir las transformaciones que pueden ocurrir durante el desarrollo del software. Para ello se utiliza una modificación de un formalismo introducido por Caplat y Sourrouille (2002), donde ***M*** es un modelo de un sistema, ***S*** es un sistema y ***F*** es el formalismo en el cual el modelo se describe. Una transformación ***t*** puede ser formulada como se ve en la fórmula (1), que define una transformación en función del modelo, el sistema y el formalismo, donde M1 es el modelo fuente y M2 es el modelo objetivo de la transformación, S1 y S2 representan el mismo sistema con algunas variantes y los formalismos F1 y F2 por lo general son diferentes.

t: $M_1(S_1) \mid F_1 \Rightarrow M_2(S_2) \mid F_2$ (que se leería M_1 de S_1 según F_1) (1)

Sin embargo, en la transformación ***t*** el modelo fuente puede mantenerse inalterado o no al generarse el programa objeto. Si el modelo fuente no se modifica, se diría que la transformación es de tipo consulta porque no produce efectos colaterales (Metzger, 2005).

3.5 Tipos de transformaciones de modelo

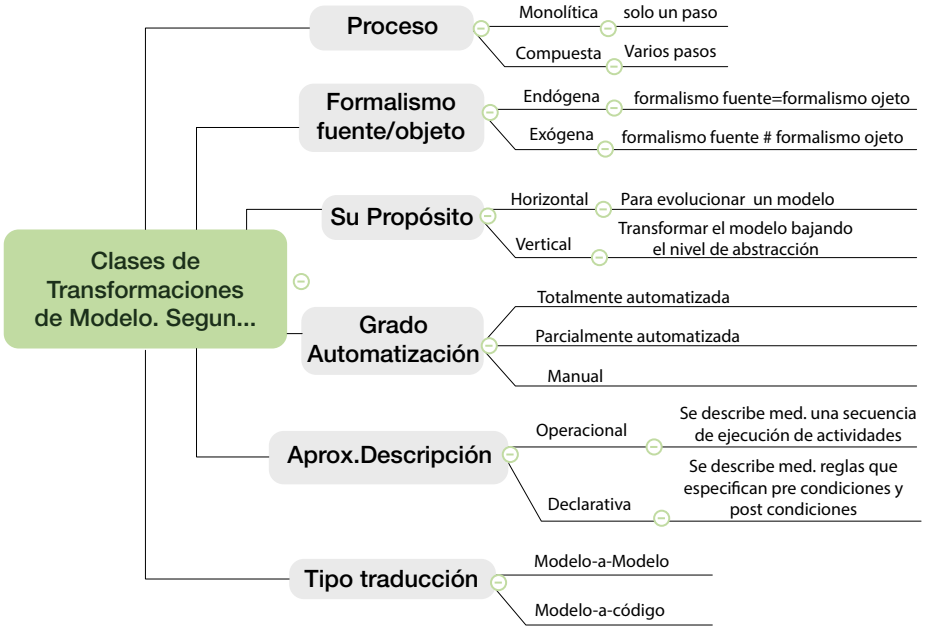
Como se ha dicho anteriormente, las transformaciones de modelo, junto con los modelos, conforman el núcleo de MDA. En la literatura se consideran diversos tipos de transformaciones de acuerdo con sus características.

En un estudio sobre diversos aspectos de transformaciones de modelos, Metzger (2005) ha determinado los tipos que se resumen en la figura 4 y se mencionan posteriormente.

Según su propósito, la transformación puede ser vertical u horizontal. Una ***transformación es vertical*** si acerca el nivel de abstracción del modelo a la plataforma objetivo o plataforma de ejecución. Una ***transformación es horizontal*** si su propósito es hacer evolucionar un modelo sin que cambie su nivel de abstracción; es decir, si su propósito es refinar el modelo. Si el propósito de una transformación vertical es una implementación real, la transformación se denomina ***generación de código***. Además, aunque las transformaciones verticales son transformaciones exógenas, no todas las transformaciones exógenas tienen que ser transformaciones verticales (Metzger, 2005).

De acuerdo con la complejidad del proceso de transformación, esta puede ser monolítica (atómica) o compuesta. En una ***transformación monolítica*** solo es posible reconocer un paso diferenciado, mientras que una ***transformación compuesta*** implica

Figura 4. Tipos de transformaciones en MDA



Fuente: Los autores

varios pasos bien diferenciados, lo que sugiere que la transformación sea progresiva (Metzger, 2005).

De acuerdo con los formalismos, los modelos fuente y el objeto, una transformación puede ser endógena o exógena. La *transformación* es *endógena* cuando el formalismo del modelo fuente es el mismo que el formalismo del modelo objeto (F1 = F2). La *transformación* es *exógena* cuando los formalismos del modelo fuente y el modelo objeto son diferentes (F1 ≠ F2) (Metzger, 2005).

De acuerdo con el grado de *automatización*, las transformaciones pueden ser totalmente automatizadas, parcialmente automatizadas o manuales. Para que una *transformación* sea *totalmente automatizada,* se requiere que los lenguajes de escritura (sintaxis), tanto del modelo fuente como del modelo objeto, sean completamente formales. En otros casos, la *transformación parcialmente automatizada* puede requerir algunas tareas manuales para su ejecución. Ahora, si los lenguajes de escritura de los modelos son poco formales, la *transformación* se tendrá que hacer *manualmente*. Sin embargo, el grado de automatización de las transformaciones también depende del grado de desarrollo de las herramientas usadas en el proceso de transformación (Metzger, 2005).

De acuerdo con la aproximación descriptiva de una transformación, esta puede ser operacional o declarativa. La ***transformación operacional*** se describe como una secuencia de actividades. La ***transformación declarativa*** se describe mediante reglas que se especifican mediante pre y postcondiciones. La precondición describe el estado del modelo antes de ser ejecutada la transformación. La transformación determina el estado después de que la transformación ha tenido éxito. Como muchos modelos o especificaciones se representan de manera gráfica, muchas de las aproximaciones declarativas que se usan actualmente se derivan de transformaciones gráficas. Las transformaciones declarativas se alcanzan con frecuencia mediante mapeo entre elementos del modelo fuente y el modelo objeto (Metzger, 2005).

Según el tipo de traducción, las transformaciones pueden ser ***modelo-a-modelo*** o ***modelo-a-código*** (Metzger, 2005). Este tipo de transformaciones resalta las diferencias entre transformaciones con base en la representación (sintaxis) de las reglas de transformación o con base en la forma de escribir lo que el lenguaje de la regla ofrece.

Otros tipos de transformaciones se pueden encontrar en Caplat y Sourrouille (2002), relacionadas con la granularidad de la transformación y el sistema; también en Favre (2004) se hace una distinción entre función de transformación e instancia de transformación, que puede ser interesante de examinar para estudios concretos de transformaciones.

4. Léxico Orientado al Modelado

Dos asuntos se plantean cuando se quiere conformar un Modelo Independiente de la Computación. Estos son: cómo establecer los requisitos reales del sistema a partir de las necesidades del usuario y cómo representar esos requisitos en el CIM. Para resolver estos asuntos se ha ideado el Léxico Orientado al Modelado (LOM), que es una estrategia gramatical basada en el léxico del dominio del problema y en la acotación del lenguaje del dominio para describir de manera formal los términos de este léxico. Lo anterior implica que el modelo independiente de la computación CIM que se presenta en este libro, esté constituido por las especificaciones LOM semiformales de los diferentes términos que definen el sistema.

El LOM se inspira en el LEL, que es una propuesta de Julio Leite (Kaplan et al., 2000). El LEL es un método gramatical para establecer el léxico del dominio del problema, cuyo propósito es disminuir la brecha de comunicación entre el ingeniero de software y los usuarios o los interesados (***stakeholders***) en el proceso de identificación de requisitos. Sin embargo, a diferencia del LEL, con el LOM se persigue

conformar una base de términos bien definidos, con características que contribuyan a representar el punto de vista independiente de la computación. Los términos LOM se pueden obtener por refinamiento del LEL o con base en los requisitos funcionales, y se especifican de tal manera que a partir de ellos es posible determinar una gran parte del Modelo Independiente de la Plataforma (PIM).

Para elaborar la propuesta de este CIM y así extender OO-Method, se tomó como referencia de PIM el modelo conceptual de OO-Method-ON que se explicará en el capítulo 5. Esta herramienta adopta el estándar de MDA y la mayoría de las recomendaciones de MDSD y, además, es una de las más avanzadas en el ámbito de MDA-MDSD; tanto que puede llegar a generar hasta el 98 % del código fuente de una aplicación a partir de modelos conceptuales bien definidos. La herramienta comercial de OO-Method-ON es Oliva Nova Model Execution, actualmente más conocida como Integranova.

En esta obra se define el LOM como un método que facilita la traducción de especificaciones funcionales a elementos del modelo CIM del sistema. Estructuralmente, el LOM está conformado por un conjunto de conceptos o términos relevantes del dominio del problema, denominados términos LOM. Los términos LOM se clasifican en cinco tipos y se especifican utilizando una plantilla similar a la prevista por Leite (Kaplan et al., 2000) para los símbolos del LEL. De hecho, su taxonomía y estructura se pueden considerar una evolución del LEL. Sin embargo, el LOM posee una estructura más elaborada y su propósito es diferente: apoyar la conformación del CIM, de forma que este facilite la generación del PIM.

Para establecer la estructura del LOM y sus términos se proponen los siguientes conceptos básicos: término LOM, ítem de especificación de término LOM, patrón de modelado, tipo de término LOM, tipo de ítem de especificación y elemento de modelado.

Un ***término LOM*** es una especificación estructurada y semiformal de un concepto relevante del dominio del problema. El conjunto de especificaciones del sistema conduce a conformar el punto de vista independiente de la computación (CIM) y posee características que se requieren para inferir elementos de modelado del PIM.

Un ***ítem** de especificación de término LOM* es un texto breve que obedece a una construcción gramatical predefinida y que hace parte de la especificación de un término LOM. Los ítems se rigen por una taxonomía preestablecida que define su función concreta dentro de la especificación del término LOM; esto se amplía en la sección 4.5.

Un *tipo de término* agrupa términos con características comunes. Aquí se consideran objetos, acciones, sujetos, estados y atributos.

Un *tipo de ítem* agrupa ítems de especificación de términos con características comunes, tales como nexo entre entidades, nexo sujeto-acción y transición entre estados. Su taxonomía se puede ver en la tabla 1.

Un *patrón de modelado* es una construcción conceptual destinada a facilitar la identificación de los potenciales elementos de modelado PIM contenidos en la especificación de un término LOM. Un patrón se caracteriza y especifica mediante una estructura gramatical predefinida y un conjunto de heurísticas asociadas.

Un *elemento de modelado* es una pieza útil para la construcción del diagrama del modelo conceptual de Oliva Nova - ON (ver sección 5.2). Se consideran dos tipos: elementos simples y elementos complejos. Los elementos simples del modelo conceptual son clases, actores, estados, servicios y atributos. Algunos ejemplos de elementos complejos son asociaciones entre clases y transiciones entre estados.

4.1 Caso de estudio – Gestión del proceso de escrituración

Para ilustrar el proceso de conformación del LOM se ha seleccionado un caso de estudio real. Se trata de un sistema de gestión de trámites de escrituras en una empresa administradora que asesora y lleva a cabo el trámite completo de escrituración de negocios de bienes inmuebles a petición de sus clientes. La descripción del dominio que se presenta a continuación se obtuvo a partir del diálogo directo con los usuarios del sistema:

La empresa que, como cliente, requiere el sistema, es una administradora de diversos tipos de bienes raíces. Su necesidad informática es un sistema informático para gestionar el proceso de escrituración para diversos conceptos de escrituras, en el que cada concepto equivale a un tipo de negocio sobre un bien. A petición de sus clientes, la administradora realiza el proceso de escrituración del negocio. Algunos de los tipos de negocios que esta incluye en su portafolio son compra-venta, hipotecas, asignaciones patrimoniales y cesiones. Para que la administradora esté habilitada para realizar los trámites de escrituración se requiere que el cliente la autorice mediante un poder, con el cual se transfieren a la administradora los derechos de tramitación de la escritura y demás asuntos asociados, como pago de impuestos y registros en las entidades públicas.

El proceso inicia cuando el cliente, quien puede o no ser el titular (dueño) del bien, le solicita a la empresa adelantar los trámites de escrituración y complementarios de un negocio de finca raíz, y termina cuando se ha cumplido el trámite de actualización del registro del bien ante la oficina de registro competente.

El proceso de escrituración es complejo: consta de una colección de trámites y actividades, internos y externos a la empresa, muchos de los cuales deben ser controlados (registro y procesamiento) por la aplicación solicitada por el cliente. Procurando seguir el orden de ejecución de las actividades del proceso, a continuación se describen las más importantes:

- El cliente, mediante un poder, autoriza a la empresa administradora para llevar a cabo el proceso de escrituración. Para esto, diligencia una solicitud y entrega una determinada suma de dinero en calidad de adelanto, para cubrir, cuando menos, los gastos correspondientes a los trámites iniciales. En cualquier momento del proceso el cliente puede hacer depósitos de dinero que se ingresan a una provisión de fondos a cuenta del cliente.

- Se asigna uno de los funcionarios (administrativos) para realizar los trámites de cada escritura, quien estudia la solicitud y elabora la programación de actividades requeridas.

- El administrativo recopila los datos y los documentos necesarios para que el notario elabore el texto de la escritura.

- El administrativo entrega los datos y los documentos necesarios al notario, quien elabora el texto de la escritura y lo devuelve para que sea revisado por el administrativo encargado y por los participantes en el negocio.

- Cuando el texto de la escritura ha sido perfeccionado, el administrativo cita a los participantes para firmar el documento.

- El administrativo paga los respectivos honorarios al notario y retira la escritura para continuar los trámites.

- Con base en la información de la escritura y el valor del negocio, se liquidan y pagan los impuestos correspondientes (rete-fuente, beneficencia, etc.). Para el pago de impuestos es necesario rellenar un formulario o modelo de liquidación, en el que se escriben los datos correspondientes al bien y al negocio que se está realizando.

- Se lleva la escritura a la oficina de registro correspondiente para actualizar el registro del bien negociado y se pagan los derechos a que haya lugar.

- Cuando ha finalizado el trámite de registro, el administrador recoge la escritura y la adjunta a la carpeta de documentación respectiva.

- El administrativo calcula los gastos generados por el proceso de escrituración y los descuenta del total de las sumas entregadas a la administradora en calidad de adelanto y aportes parciales, para determinar si el cliente debe algún dinero adicional o si, por el contrario, se le debe devolver algún excedente.

Para tener mayor claridad del dominio del negocio, se precisaron con el cliente los siguientes conceptos.

Una ***escritura*** es un documento público que formaliza un negocio de un bien (finca raíz) entre dos o más participantes. En un negocio de compraventa, por ejemplo, algunos participantes son compradores y otros son vendedores. Un participante puede ser natural, residente o no residente en el país, lo que puede marcar algunas diferencias en los trámites.

El proceso de escrituración genera ***gastos*** por diferentes conceptos, tales como honorarios de notaría, pagos de impuestos, derechos de registro y honorarios de la empresa.

Los ***negocios***, también denominados ***conceptos*** o ***conceptos de negocios***, pueden ser diversos; entre los más representativos se tienen compraventa, hipoteca, ratificación, subsanación y transmisión patrimonial, entre otros.

Los ***impuestos*** son derechos que se pagan a entidades del Estado cuando se realiza un negocio. Cada impuesto cubre unos determinados derechos y, para su pago, se requiere el diligenciamiento de un modelo de liquidación.

Las principales entidades que participan en el proceso de escrituración son el notario, el centro gestor y la oficina de registro.

Un ***notario*** que despacha en una notaría es una persona jurídica investida por el Estado para realizar y dar fe de un conjunto de operaciones públicas y negociaciones, como la escrituración.

Un ***centro gestor*** es una entidad estatal donde se administran las operaciones relacionadas con los impuestos de los contribuyentes (en este caso del titular o del cliente).

La *oficina de registro* es una entidad estatal que controla la bitácora o la historia de las negociaciones asociadas a un determinado bien.

Los *bienes* pueden ser de diversos tipos, como casas, lotes, apartamentos, bungalós, locales, etc.

Un *concepto de negocio* en una escritura puede ser por compra-venta, hipoteca, etc.

Una persona puede ser incluida en una escritura como *participante*, como *interviniente* o como *testigo*.

El *proceso de escrituración* consiste en una colección de *fases* conformadas por *tareas* concretas que deben ser controladas por la aplicación.

Los *requisitos funcionales* más relevantes que expresaron los usuarios se describen brevemente a continuación:

La aplicación debe administrar toda la información relacionada con los trámites de las escrituras que solicitan los clientes a la administradora. Esto implica gestionar:

- La información correspondiente a cada escritura. Esto incluye mantener actualizadas las escrituras y los soportes que se anexan para su procesamiento.

- La información de las personas que participan en el negocio que describe la escritura, entre quienes se encuentran los participantes, los intervinientes y los testigos.

- La información de los clientes, quienes son las personas que autorizan a la administradora para realizar el proceso de escrituración.

- Los depósitos que el cliente hace a la cuenta del trámite de una o más escrituras.

- El control de los gastos que implica el trámite y el cruce de cuentas con el fondo de cada cliente. Aquí se incluyen los impuestos, los derechos notariales, los derechos de registro y los demás que puedan surgir.

- El control del proceso del trámite de cada escritura (*workflow*), de manera que sea posible establecer la fase y la tarea en la que está actualmente y las tareas que faltan por terminar. El sistema debe avisar cuando una escritura está atrasada en su proceso y la causa del atraso.

- La información de los bienes objeto de negocio, así como su tipo y destinación.

- La información de las entidades relacionadas con el proceso: notarios, centros gestores y oficinas de registro.

- La información complementaria requerida para el buen funcionamiento del sistema: empresa del cliente, poblaciones, países y detalles de IVA.

Con base en la descripción anterior, se pueden identificar los términos relevantes que, una vez sometidos a los criterios de selección propuestos en la sección 7.3, conforman una lista como la que se muestra en la tabla 1, que se ha tomado como ilustración para desarrollar el resto de este capítulo.

4.2 Lista de términos LOM

Como se dijo anteriormente, la identificación de los términos relevantes del sistema o términos LOM se puede hacer a partir del LEL o a partir de la descripción del dominio del negocio del sistema y los requisitos funcionales. Por agilidad del proceso, se ha elegido hacerlo con base en la segunda opción.

Así, los términos LOM se identifican de acuerdo con las definiciones correspondientes a cada uno de los tipos, como se puede ver en la sección 4.4, es decir, objetos, atributos, acciones, estados y sujetos. En la medida en que se van identificando los términos, se agregan a una tabla como la tabla 1, en la que la primera columna muestra los consecutivos de los términos por tipo, la segunda columna muestra los nombres de los términos junto con sus alias (sinónimos) y la tercera columna muestra el tipo al que pertenece cada término. El número consecutivo de cada término debe coincidir con el número asignado a la especificación (Cs) del término más adelante.

Tabla 1. Lista de Términos Simples LOM

Cs	Término/Alias	Tipo
	Objetos	
1	Escritura	Ob
2	Fase / Fase de escrituración	Ob
3	Tarea / Tarea de una Fase	Ob
4	Concepto / Concepto de Escritura / Negocio	Ob
5	Categoría de concepto / Tipo de negocio	Ob
6	Tipo de participante / Calidad compareciente	Ob

Cs	Término/Alias	Tipo
	Objetos	
7	Bien / Objeto de negocio	Ob
8	Modelo de impuesto / Modelo	Ob
9	Tipo de bien	Ob
10	Honorario	Ob
11	Tipo de honorario	Ob
12	Provisión de fondos / Entrega en cuenta	Ob
13	Registro / Oficina de registro	Ob
14	Notario	Ob
15	Modelo de Liquidación / Formulario de liquidación	Ob
16	Gasto	Ob
17	Impuesto	Ob
18	Participante / Participante en escritura / Compareciente	Ob
19	Interviniente / Testigo	Ob
20	Cliente	Ob
21	Centro Gestor / Oficina de Recaudo	Ob
22	Minuta	Ob
	Acciones	
1	Liquidar gasto / Calcular gastos	A
2	Registrar escritura / Registrar Bien	A
3	Solicitar trámite / Solicitar trámite escritura	A
4	Liquidar Impuesto / Liquidación Impuesto	A
5	Firmar escritura / Firmar	A
6	Pagar Impuesto	A
7	Editar escritura	A
8	Catalogar escritura	A
	Sujetos	
1	Usuario	Su
2	Funcionario / Administrativo	Su
3	Administrador	Su
	Estados	
1	Escritura firmada	E
2	Escritura registrada	E
3	Escritura ratificada	E
4	Escritura en notaría / Escritura en edición	E
5	Escritura en registro	E
	Atributos	
1	Tarifa (de honorario)	At

Cs	Término/Alias	Tipo
	Atributos	
2	Valor de impuesto	At
3	Importe / Valor negocio	At
4	Fecha de petición	At
5	Número / Número de escritura	At
6	Nombre usuario / Nombre	At
7	Código usuario / Código	At

4.3 Estructura de un término LOM

Para especificar los términos LOM se ha ideado una plantilla que facilita la posterior identificación de los elementos de modelo conceptual de OO-Method (PIM). Esta plantilla es una evolución mejorada de la plantilla utilizada para especificar los símbolos LEL. Su estructura es la siguiente:

TÉRMINO (No.): (Nombre, Término y Número)
PLURAL: (Nombre del Término en Plural)
ALIAS: (Pueden ser cero o más)
TIPO: (Ob, A, At, Su, E)
NOCIÓN: (Uno o más ítems)
IMPACTO: (Uno o más ítems)

Donde ***término*** indica que a continuación se escribe el nombre del término; ***No.*** se remplaza por tipo de la lista de términos y el consecutivo correspondiente; ***plural*** se usa para escribir la forma plural del término; ***alias*** corresponde a sinónimos del término u otras formas de denominación del término por parte del usuario; la ***noción*** está constituida por uno o más ítems que definen las características estáticas, propias del término; y el ***impacto*** está conformado por uno o más ítems que definen las características dinámicas del término, y describen cómo este afecta a otros términos o cómo es afectado por otros términos.

Como se muestra en el ejemplo 1, la noción y el impacto están constituidos por ítems y cada ítem está conformado por expresiones externas e internas al LOM. Las expresiones externas son construcciones del lenguaje natural acotado (sección 7.5) y las expresiones internas son otros términos LOM previamente registrados en la lista de términos, que son usados para especificar ítems de términos. Por otro lado, cada ítem se ha dotado con una estructura particular que facilita su construcción, como se muestra más adelante.

Ejemplo 1: Especificación del término – Escritura

TÉRMINO (Ob.1): Escritura
PLURAL: Escrituras
ALIAS: Escritura pública
TIPO: Ob

NOCIÓN:

Documento público que formaliza y legaliza uno o más negocio/s (Ob.4) pactados entre dos o más participante/s (Ob.18).

Una escritura (Ob.1) tiene una fecha de petición (At.4)
Una escritura (Ob.1) tiene un número (At.5)
Una escritura (Ob.1) se asigna a un notario (Ob.14)
Una escritura (Ob.1) es solicitada por (solicitud) de un cliente (Ob.20)
Una escritura (Ob.1) incluye uno o más negocio/s (Ob.4).

IMPACTO:

Una escritura (Ob.1) es registrada (registrar escritura (A.2)) por un administrativo (Su.2)
Una escritura (Ob.1) es firmada (firmar escritura (A.5)) por un participante (Ob.18)
Una escritura (Ob.1) es editada (editar escritura (A.5)).

En este ejemplo se evidencian las siguientes características y convenciones:

- Número de término al inicio (Ob.1). Ob denota el tipo del término y el número indica el consecutivo en una lista LOM.

- Términos subrayados. Se corresponden al nombre, plural o alias de un término que debe estar incluido en la lista LOM.

- Identificación de los términos subrayados. La parte textual hasta el punto denota el tipo del término y el número indica el consecutivo en una lista LOM.

- Barra inclinada. Se usa para expresar el singular-plural.

4.4 Tipos de términos LOM

Una vez identificados los términos relevantes del sistema consignados en la lista de términos (tabla 1), se procede a describir cada término con la ayuda de los usuarios del sistema. Este ejercicio permite obtener un mejor conocimiento del dominio del problema y, de forma más precisa, de las necesidades del usuario con respecto a la aplicación.

En la concepción del LOM, la especificación funcional de un sistema y su descripción en lenguaje natural pueden contener términos de cinco tipos diferentes. De acuerdo con las entidades o conceptos que estos representan, tales tipos son: objeto, atributo, acción, sujeto y estado. Cada tipo de término se especifica mediante ítems de noción e ítems de impacto.

En todos los casos, las nociones de todos los tipos de términos incluyen un ítem de tipo definición. Los demás ítems se describen junto con los tipos de términos a continuación.

Objeto. Es una entidad que desde el punto de vista del modelado OO del sistema se percibe como un objeto que pudiera ser clasificable y que se puede diferenciar de otros objetos de la misma clase; mediante una característica particular se le puede asignar una identificación única. El ejemplo 1 representa la especificación del término tipo objeto escritura. Un objeto escritura pertenece a la clase escritura y se diferencia de otros objetos de la misma clase mediante un número único.

Atributo. Es una entidad que desde el punto de vista del modelado OO del sistema se percibe como una característica o propiedad de un objeto. Como tal, en un determinado momento podrá asumir un valor dentro de un determinado dominio de valores (ver el ejemplo 2).

Ejemplo 2: Especificación del término Valor Impuesto

TÉRMINO (At.2): Valor Impuesto
PLURAL:
ALIAS:
TIPO: A

NOCIÓN:

Representa el monto de dinero que se debe pagar por un determinado impuesto (Ob17). El valor impuesto (At2) no puede superar el 10 % del valor negocio (At3).

IMPACTO:

Liquidar impuesto (A7) reemplaza el valor impuesto (At2) con el valor calculado.

Acción. Representa una actividad concreta realizada por algún sujeto del sistema y que puede afectar a uno o más objetos. Al determinar el objeto afectado es posible obtener un servicio de una clase (ver ejemplo 3).

Ejemplo 3: Especificación del término Registrar Escritura

TÉRMINO (A2): Registrar Escritura
PLURAL:
ALIAS: Registrar / Registrar Bien
TIPO: A

NOCIÓN:

Representa un acto legal por el que se actualiza la propiedad de un bien (Ob7) en la oficina de registro (Ob13) y del que queda constancia en la escritura (Ob1). La acción de registrar escritura (A2) es efectuada por un administrativo (Su1).

IMPACTO:

Registrar escritura (A2) cambia el estado de la escritura (Ob1) de escritura firmada (E1) a escritura registrada (E2). Para registrar escritura (A2) es obligatorio que todos los impuestos/s (Ob17) estén pagados. Registrar escritura (A2) se debe hacer dentro de los 30 días después de haber realizado la acción firmar escritura (A2).

Sujeto. Representa personas o sistemas que comúnmente interactúan con el sistema y que ejecutan directamente una o más acciones (ver ejemplo 4). Desde el punto de vista de modelado se percibe como un actor del sistema y, como tal, se le pueden asociar un conjunto de acciones que son activadas por tal actor en el sistema.

Ejemplo 4: Especificación del término Funcionario

TÉRMINO (Su2): Funcionario
PLURAL:
ALIAS: Administrativo
TIPO: Su

NOCIÓN:

El funcionario (Su2) es un empleado de la administradora que está encargado de tramitar las escritura/s (Ob1)

Un funcionario (su2) tiene nombre (At1)
Un funcionario (su2) tiene código (At1)
Un funcionario (su2) es un usuario (Su1).

IMPACTO:

El funcionario (Su2) efectúa la acción de registrar escritura (A2)
El funcionario (Su2) efectúa la acción de liquidar impuesto (A7)
El funcionario (Su2) efectúa la acción de liquidar gasto (A1)
El funcionario (Su2) efectúa la acción de pagar impuesto (A6)
El funcionario (Su2) registra en el sistema la acción de firmar escritura (A5)
El funcionario (Su2) registra en el sistema la acción de solicitar trámite (A9).

Estado. Representa una de varias situaciones posibles por las que puede pasar un objeto o un sujeto (ver ejemplo 5). Esta situación se entiende como una circunstancia relevante o hito del objeto, en la que este puede ejecutar ciertas acciones, pero no otras.

Ejemplo 5: Especificación del término Escritura Firmada

TÉRMINO (E1): Escritura Firmada
PLURAL:
ALIAS:
TIPO: Estado

NOCIÓN:

Indica que el documento de escritura (Ob1) realizado por el notario (Ob14) está perfeccionado y todos los participantes en el negocio (Ob4) han efectuado la acción firmar escritura (Ob5). Escritura firmada (E1) es un estado de una escritura (Ob1).

IMPACTO:

Firmar escritura (A5) conduce a escritura firmada (E1)
Escritura registrada (E1) ocurre a partir de registrar escritura (A2)
Escritura firmada (E1) ocurre a partir de catalogar escritura (A8).

4.5 Ítems de especificación

Como se expresó en la sección anterior, la noción y el impacto de un término LOM se especifican mediante uno o más ítems denominados ítems de especificación. La introducción de este concepto ayuda a estructurar la especificación de los diversos tipos de términos.

4.5.1 Términos LOM e ítems de especificación

Un ítem de término LOM es una unidad estructurada que, de acuerdo con su tipo, cumple una función concreta de especificación en la noción o impacto de un término LOM. Esta estructura obedece al formato gramatical previamente establecido (ver estructura general al inicio de la sección 4.3 y los ejemplos siguientes).

Al analizar el ejemplo 1 de la sección 4.3, que especifica el término ***escritura***, se observa que en este término la noción se describe mediante cinco ítems y el impacto mediante tres ítems. En la especificación de cada ítem se aplica el principio de ***circularidad,*** dado que la especificación de cualquier ítem se hace en función de términos LOM previamente identificados. Adicionalmente, se aplica el principio de ***minimalidad*** en la medida en que en las especificaciones se utiliza un lenguaje externo acotado, expresado principalmente por frases claves que enlazan los términos LOM de cada ítem. Estos principios también se utilizan en la especificación de todos los demás tipos de términos catalogados, de forma que su aplicación garantiza la consistencia y la eficiencia de las especificaciones.

Por otro lado, cada ítem expresa algún tipo de relación entre pares de términos o una restricción sobre algún término. Tanto los tipos de relaciones como los tipos de restricciones dependen de la naturaleza de los términos participantes en la especificación.

Como consecuencia de la diversidad de relaciones o restricciones expresadas, los ítems de especificación de términos pueden ser clasificados en varios grupos catalogados y definidos en la siguiente sección.

4.5.2 Taxonomía de ítems de especificación

Con base en el análisis de las definiciones de términos de los ejemplos citados en la sección 4 y en las especificaciones, de muchos de los términos que aparecen en la lista no se muestra su definición, pero se determinaron los tipos de ítems que se listan y describen en lo que queda de esta sección.

La tabla 1 muestra los elementos del modelo conceptual que se pueden obtener de los términos LOM. Con base en la especificación de estos términos se infiere que cada tipo de ítem puede dar lugar a uno o más elementos complejos de modelado. Así, en la tabla 2, para cada conjunto de elementos se muestra la referencia cruzada entre término e ítem, ya sea de noción o de impacto, pero no se muestra la referencia para los ítems de noción (definición) por no tener una estructura patrón, lo que impide anticipar si generan o no un elemento de modelado. Por lo tanto, se dice que estos ítems no generan elementos de modelado, lo que se identifica como NGEM.

Tabla 2. Elementos complejos de modelado obtenidos de ítems de especificación

TIPO ÍTEM	TÉRMINO				
	Objeto	**Atributo**	**Acción**	**Sujeto**	**Estado**
1 **Definición**	Noción: NGEM	Noción: NGEM	Noción: NGEM	Noción: NGEM	Noción: NGEM
2 **Nexo entre objetos**	Noción: - Asociación - Agregación - Herencia, entre Clases			Noción: - Herencia entre Actores	
3 **Nexo Atributo - Objeto/ Sujeto**		Noción: - Pertenencia		Noción: - Pertenencia	
4 **Nexo Acción-Objeto**	Impacto: - Servicios de una Clase / servicios de un Actor				
5 **Nexo Sujeto-Acción**			Noción: - Relación Actor - Servicio	Impacto: - Relación Actor - Servicio	
6 **Nexo Estado- Estado (transición)**			Impacto: -Transición entre Estados (servicio)		

TIPO ÍTEM	TÉRMINO				
	Objeto	**Atributo**	**Acción**	**Sujeto**	**Estado**
7 **Restricción-Acción**			Impacto: - Precondición - Restricción sobreservicio		
8 **Nexo** **Acción-Atributo**		Impacto: - Servicio afecta el valor de Atributo			
9 **Restricción de Atributo**		Noción: - Restricción sobre atributo			
10 **Nexo Estado-Entidad**					Noción: - Estado de entidad (O/Su)
11 **Nexo Estado-Acción**					Impacto: - Servicios posibles en este estado
12 **Nexo** **Acción-Estado**					Impacto: - Servicio que conduce a un estado

Ítem de definición (1). Expresa una breve descripción del término desde el punto de vista del usuario del sistema. No se rige por un formato particular y es independiente del tipo de término.

Un ejemplo de ítem de definición de un término de tipo sujeto es:

El administrativo (Su2) es un empleado de la administradora que está encargado de tramitar las escritura/s (Ob1)

Todos los tipos de términos incluyen un ítem de definición como primer ítem de la noción. Por flexibilidad, los ítems de definición no obedecen a un formato gramatical específico, sino que se adaptan a las particularidades del término que define. Sin embargo, pueden incluir términos previamente identificados, como se pudo apreciar en los ejemplos del 1 al 5, de la sección 4.4.

Ítem de nexo entre objetos (2). Si la entidad es un objeto, el ítem puede sugerir diversas relaciones entre objetos, lo que conduce a inferir relaciones de asociación, agregación y herencia entre clases de objetos. Si la entidad es un sujeto, permite descubrir relaciones de herencia entre actores.

Los siguientes son ejemplos de ítems de nexo entre entidades de las nociones de los términos ***escritura*** y ***funcionario***.

Una escritura (Ob.1) se asigna (su elaboración) a un notario (Ob.14)
Una escritura (Ob.1) incluye uno o más negocio/s (Ob.4)
Un funcionario (su2) es un usuario (Su1)

Los anteriores ítems establecen relaciones entre dos objetos o entre dos sujetos. Una frase verbal establece el nexo o la relación entre los objetos. El primer ítem expresa una asociación, el segundo una agregación y el tercero una herencia entre sujetos.

Ítem de nexo atributo-objeto/sujeto (3). Se presenta en la definición de términos tipo objeto o sujeto y expresan la pertenencia de un atributo a un objeto o a un sujeto.

Los siguientes son ejemplos de ítems de nexo entre entidades de las nociones de los términos ***escritura*** y ***funcionario***.

Una escritura (Ob.1) tiene una fecha de petición (At.1)
Una escritura (Ob.1) tiene un número (At.2)

Los anteriores ítems establecen relaciones entre atributos y objetos. Una frase verbal establece el nexo de pertenencia.

Ítem de nexo acción-objeto-[sujeto] (4). Indica qué acciones recaen sobre un objeto y, opcionalmente, también podría señalar el sujeto responsable de tal acción. Conduce a inferir algunos servicios o métodos de una clase de objetos y puede incluir un nexo con un sujeto.

Los siguientes son ejemplos de ítems de nexo acción-objeto correspondientes al impacto del término tipo objeto ***escritura***:

La escritura (Ob.1) es registrada (registrar escritura (A.2)) por un administrativo (Su.1)
La escritura (Ob.1) es firmada (firmar escritura (A.5))

En los dos ejemplos anteriores la escritura es afectada por dos acciones: registrar-escritura y firmar-escritura. En ambos casos se puede considerar que las acciones son propias del objeto escritura. Por otro lado, en el primer ejemplo se hace explícito el sujeto que efectúa la acción, de manera que, además, se establece que el actor que inicia tal actividad es un *administrativo*.

Ítem de nexo sujeto-acción (5). Establece las acciones que ejecuta un determinado sujeto. Este tipo de ítem conduce a definir relaciones actor-servicio.

Los siguientes son ejemplos de ítems de nexo sujeto-acción correspondientes al impacto del término tipo sujeto *funcionario*:

El funcionario (Su2) efectúa la acción liquidar impuesto (A7)
El funcionario (Su2) efectúa la acción liquidar gasto (A1)
El funcionario (Su2) registra en el sistema la acción firmar escritura (A5)

En estos casos, cada ítem especifica una acción que el sujeto (funcionario) lleva a cabo. Las acciones ejecutadas por el sujeto recaen sobre diferentes objetos o sujetos, de manera que sugieren qué actor es responsable de iniciar tales acciones.

Ítem de nexo estado-estado (transición) (6). Para un determinado objeto o sujeto, especifica el cambio de estado de un objeto o de un sujeto producido por una acción, y especifica los estados origen y destino. Ayuda a construir Diagramas de Transición entre Estados (DTE) si tal acción se transforma en un servicio que ocasiona que un objeto o un sujeto pase de uno a otro estado. El DTE representa, mediante un gráfico basado en una máquina de estados finitos, los principales estados o situaciones por los que un objeto puede pasar durante su vida en el sistema, y la forma como ocurren las transiciones entre estados.

El siguiente es un ejemplo de ítem de transición entre estados correspondiente al impacto del término *Registrar Escritura*:

Registrar escritura (A2) cambia el estado de la escritura (Ob1) de escritura firmada (E1) a escritura registrada (E2)

Este ítem especifica que *Registrar Escritura* es la acción que genera la transición *escritura firmada – escritura registrada*.

Ítem de Restricción de Acción (7). Describe una condición que debe cumplirse para poder ejecutar una acción, o una restricción que limita el entorno

de tal acción. El primer caso conduce a definir una precondición sobre un servicio y el segundo caso conduce a definir una restricción estática.

Los siguientes son ejemplos de ítems de restricción de acción correspondientes al impacto del término tipo acción ***Registrar Escritura***:

Para registrar escritura (A2) es obligatorio que todos los impuestos/s (Ob17) estén pagados.

Registrar escritura (A2) se debe hacer dentro de los 30 días después de haber realizado la acción firmar escritura (A2)

El primer ítem establece una precondición sobre la acción ***Registrar Escritura,*** y el segundo ítem determina una restricción estática sobre tal acción (vencimiento de pago).

Ítem de nexo acción-atributo (8). Indica, para un atributo, una acción que afecta su valor. Es útil para formular el modelo funcional del sistema que establece la manera como un servicio afecta el valor de un atributo.

El siguiente es un ejemplo de ítem de nexo acción-atributo correspondiente al impacto del término tipo atributo ***Valor Impuesto***:

Liquidar impuesto (A7) reemplaza el valor impuesto (At2) con el valor calculado.

El anterior ítem señala que una acción (***liquidar impuesto***) afecta el valor de un atributo (***valor impuesto***) y sugiere que es un atributo derivado.

Ítem de restricción de atributo (9). Indica, para un atributo, una restricción sobre su valor. Es útil para formular una restricción sobre el valor del atributo actual.

El siguiente es un ejemplo de ítem de ***Restricción de Atributo*** correspondiente a la noción del término ***Valor Impuesto***:

El valor impuesto (At2) no puede superar el 10 % de valor negocio (At3)

El anterior ítem establece una restricción de valor máximo que puede asumir el atributo ***valor impuesto***.

Ítem de nexo estado-entidad (10). Señala, para el actual estado, a qué objeto o sujeto pertenece. Ayuda a construir DTE, indicando a qué entidad (clase de objetos/actor) pertenece el estado.

El siguiente es un ejemplo de ítem de nexo estado-entidad correspondiente a la noción del término tipo estado *Escritura Firmada*:

Escritura firmada (E1) es un estado de una escritura (Ob1)

El anterior ítem permite asociar un estado con su objeto.

Los nexos estado-acción y acción-estado son de mucha ayuda para conformar el diagrama de transición entre estados, pero aún no es claro si se pueden obtener o no de la descripción de los términos.

5. OO-Method y OlivaNova Model Execution

5.1 OO-Method

Como se expresó en la introducción de este libro, el objetivo central de este documento es conformar bases de un CIM para extender OO-Method y su herramienta de implementación Oliva Nova Model Execution (ONME). En este capítulo se describen los dos y se hace precisión en los elementos del modelo conceptual de OO-Method que tienen posibilidades de ser obtenidos desde el CIM.

Desde el punto de vista de MDA, ONME automatiza la mayor parte del ciclo de desarrollo de software. Con base en el modelo conceptual (PIM de ONME), el modelo de ejecución define e implementa las reglas de transformación para generar código fuente en diversos entornos de programación como Java y C Sharp. En esta sección se describe OO-Method y se presta atención a los elementos del modelo conceptual que se pueden obtener a partir de un modelo más abstracto, en este caso del CIM que se propone aquí.

Las características más relevantes de OO-Method, de acuerdo con Molina y Pastor (2004), son las siguientes:

- Separa claramente el espacio del problema (el QUÉ) del espacio de la solución (el CÓMO). Esta característica establece un punto común con uno de los principios

fundamentales de MDA: la separación de la lógica de las aplicaciones de sus posibles implementaciones. Se diría que, en este sentido, OO-Method es ***compliant*** con MDA.

- OO-Method está conformado por dos componentes generales: el modelo conceptual y el modelo de ejecución. La definición del problema se representa mediante el modelo conceptual, que no es otra cosa que la descripción abstracta de una aplicación.

- El modelo conceptual describe el problema desde cuatro vistas complementarias: el modelo de objetos, el modelo dinámico, el modelo funcional y el modelo de presentación.

- Dado que cada una de las vistas del modelo conceptual están conformadas por patrones con semántica bien definida, es posible describir las características funcionales de una aplicación a partir de ellas.

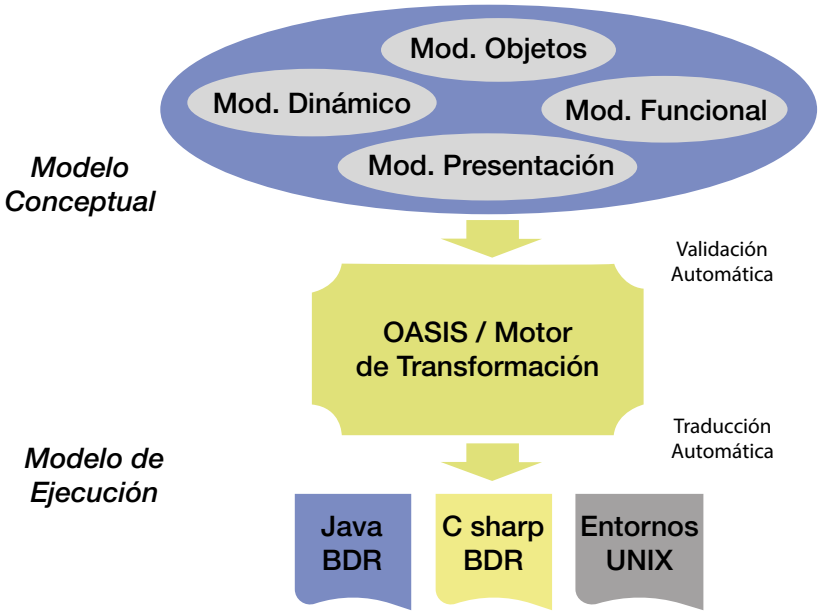
- La notación gráfica de la mayor parte de los patrones conceptuales está basada en UML. Este hecho favorece el ocultamiento de la complejidad propia del sistema y de su especificación formal OASIS[1] subyacente.

- El modelo de ejecución define las reglas de transformación de un modelo conceptual en su representación en código fuente. Mediante este mecanismo se conserva la semántica de los elementos del modelo conceptual y asegura la equivalencia funcional entre la aplicación resultante y el mismo modelo.

OO-Method entonces es una estrategia metodológica que, al igual que MDSD, centra su atención en el modelado del sistema, y está conformada por dos componentes globales: el modelo conceptual y el modelo de ejecución. Estos modelos respectivamente dan cuenta del dominio del problema y del dominio de la solución, como se muestra en la figura 5. Por su parte, el modelo de ejecución convierte las especificaciones del modelo conceptual en código de un lenguaje de programación, mientras el modelo conceptual se transforma en especificaciones formales OASIS y estas, a su vez, se transforman en el código de la aplicación, siguiendo un conjunto de reglas de transformación específicas de la plataforma en la que se va a ejecutar.

[1] Lenguaje de especificación formal fundamentado en varios tipos de Lógicas matemáticas.

Figura 5. Estrategia metodológica OO-Method implementada en ONME



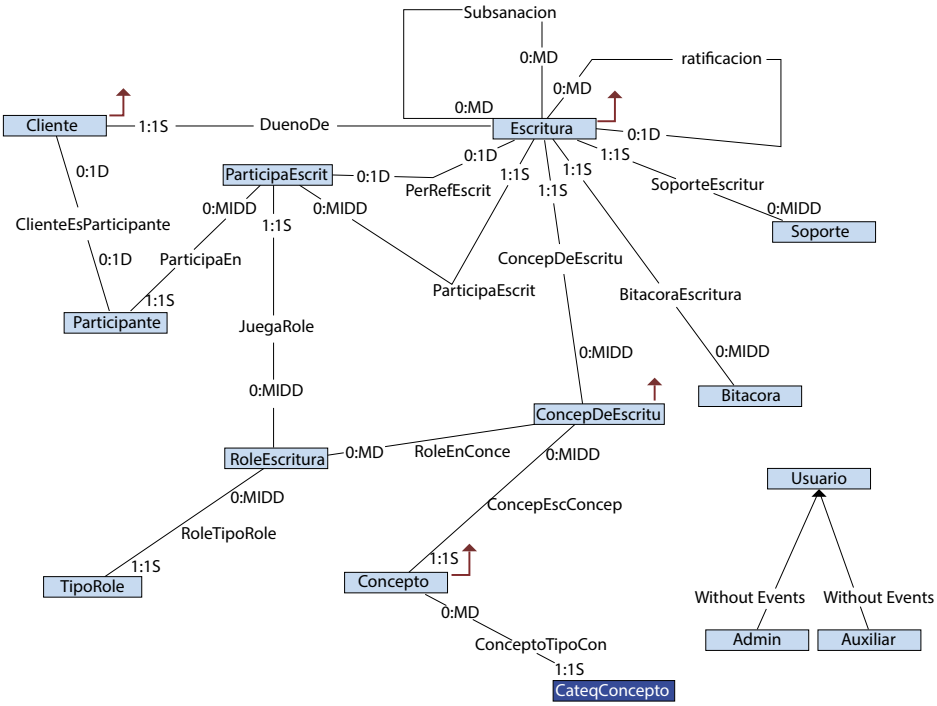
Nota: Tomado de Molina (2003).

Dada la importancia de las cuatro vistas que el modelo conceptual tiene para la comprensión del tema, estas se detallan en las siguientes secciones.

5.1.1 Modelo de objetos

El modelo de objetos de OO-Method representa la parte estructural del sistema. Está constituido por el diagrama de clases (ver figura 6) y un conjunto amplio de elementos adicionales que permiten recopilar especificaciones detalladas del sistema en construcción. El diagrama de clases, además de representar las clases que conforman el sistema y sus interrelaciones, expresa características detalladas propias de los atributos, los servicios y las relaciones entre clases. En OO-Method, además de las clases de objetos, existen clases para representar los actores (usuarios) del sistema.

Figura 6. Diagrama de clases OO-Method implementado en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*

Las relaciones entre clases pueden ser agregaciones, herencia y agente. Las dos primeras son relaciones estáticas; la tercera representa el aspecto cliente/servidor de la sociedad de objetos del sistema (Insfran, et al., 2002).

En la figura 6 se muestra un diagrama de clases y en él se destacan las clases y sus relaciones. La clase escritura se percibe como una agregación inclusiva de las clases concepto, soporte y bitácora. Entre las clases escritura y cliente existe una agregación referencial. Las clases admin y auxiliar representan actores del sistema y se generalizan mediante herencia permanente en la clase usuario.

Los elementos más relevantes del modelo de objetos y que se espera que se puedan obtener a partir del LOM son clase, actor, agregación, herencia, agente, atributo, precondición y servicio.

Una *clase* de objetos es una abstracción que permite agrupar objetos del mismo tipo. Una clase puede representar entidades reales como el *cliente,* o abstractas como *tipo de negocio*.

Un *actor* es una clase especial que representa a un determinado tipo de usuarios del sistema. Un actor puede iniciar una acción, puede ver determinados atributos y puede navegar entre objetos del sistema. Tanto el administrador como el auxiliar de la figura 6 son actores.

Una relación de *agregación inclusiva* indica que un objeto de una clase está conformado física o lógicamente por objetos de otra u otras clases. Por ejemplo, una escritura está conformada (lógicamente) por anexos y negocios. En la literatura común, la agregación inclusiva simplemente se denomina *agregación*.

Una relación de *agregación referencial* indica la existencia de un nexo entre los objetos de una o más clases. Un ejemplo es el nexo existente entre un vehículo y la persona dueña de este. En la literatura común, la agregación referencial de OO-Method equivale a la asociación entre clases (ej. UML).

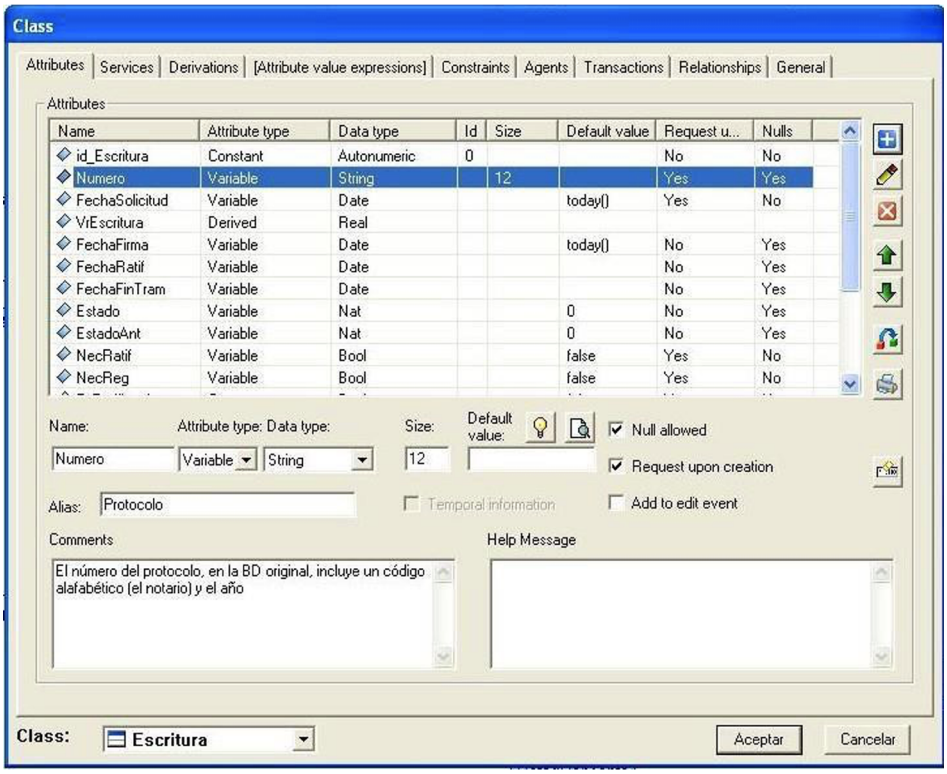
Una relación de *herencia* especifica que un grupo de objetos (la clase hija) hace parte de un tipo más amplio (la clase padre). La herencia se puede percibir como especialización cuando las clases hijas se obtienen como una subcategoría de la clase padre, o como una generalización cuando la clase padre se abstrae de las clases hijas. Un ejemplo de especialización es que las clases administrador y funcionario son subcategorías de la clase usuario y, en consecuencia, heredan los atributos y acciones de la clase usuario.

Adicionalmente, una clase OO-Method posee elementos relevantes para este estudio como atributos, servicios y precondiciones.

Un *atributo* es una característica relevante compartida por todos los objetos de una clase y de la que se requiere que el sistema guarde su valor para cada objeto. En la figura 7 se muestran los atributos de la clase escritura. Los atributos pueden ser constante, variable o derivado. Un atributo constante es aquel que, una vez creado el objeto, su valor no varía; como, por ejemplo, el número de la escritura (id_Escritura). Un atributo variable es ese cuyo valor puede cambiar durante la vida del objeto; por ejemplo, el estado de la escritura (Estado), el valor de la escritura (vrEscritura) o la fecha de solicitud de la escritura (FechaSolicitud). Un atributo derivado es similar a un atributo variable, pero su valor se obtiene a partir de una fórmula que puede incluir otros atributos; por ejemplo, el número de soportes de una escritura

se obtiene contando los soportes de una determinada escritura (mediante la agregación escritura-soporte que se ve en la figura 6). En condiciones normales, a partir de la descripción de un sistema o a partir del LOM se pueden identificar atributos de objetos, pero es difícil decidir su tipo u otros detalles.

Figura 7. Especificación de los atributos de la clase escritura implementada en ONME

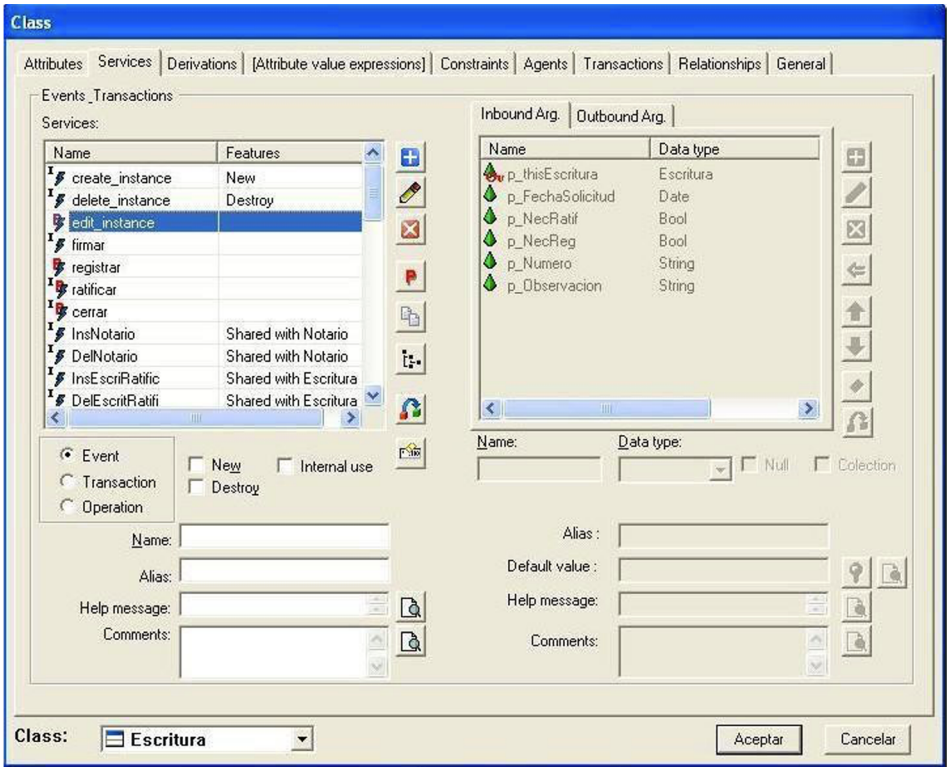


Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

Un *servicio* es una acción que puede ejecutar cualquier objeto en determinadas condiciones y se corresponde con el concepto de método de la programación orientada a objetos. En OO-Method, un servicio puede ser evento, transacción u operación. Un evento es una acción puntual que actúa sobre uno o más atributos de una clase. Una transacción es un servicio compuesto por otros servicios (eventos, transacciones, operaciones) y que exige que se ejecuten todos sus servicios componentes totalmente, para que su acción sea válida. Una operación es similar a una transacción, pero sin la restricción de ejecución total. En la figura 8 se muestran los servicios

correspondientes a la clase *Escritura,* en la que las transacciones y las operaciones se muestran en mayúscula, especificando su tipo.

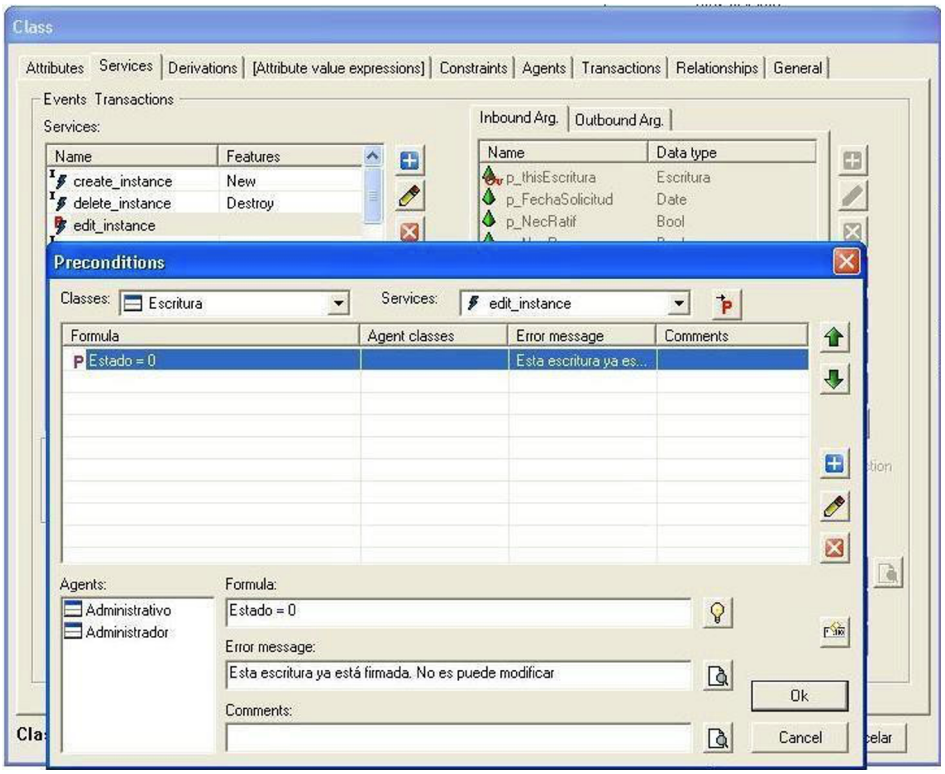
Figura 8. Especificación de servicios de la clase Escritura implementada en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement.*

Una *precondición* es una restricción que se representa como una condición y que se debe cumplir para poder ejecutar un servicio. Una precondición se asocia directamente con un servicio y se expresa mediante una fórmula que incluye la condición y un mensaje de error que, en caso de que no se cumpla la condición, advierte por qué no se puede ejecutar el servicio. En la figura 8 se presentan tres servicios con precondiciones asociadas, las cuales se distinguen porque tienen una P roja a la izquierda; y en la figura 9 se muestra la especificación de la precondición, incluyendo su condición y mensaje de error.

Figura 9. Especificación de una precondición asociada a un servicio de la clase escritura implementada en ONME

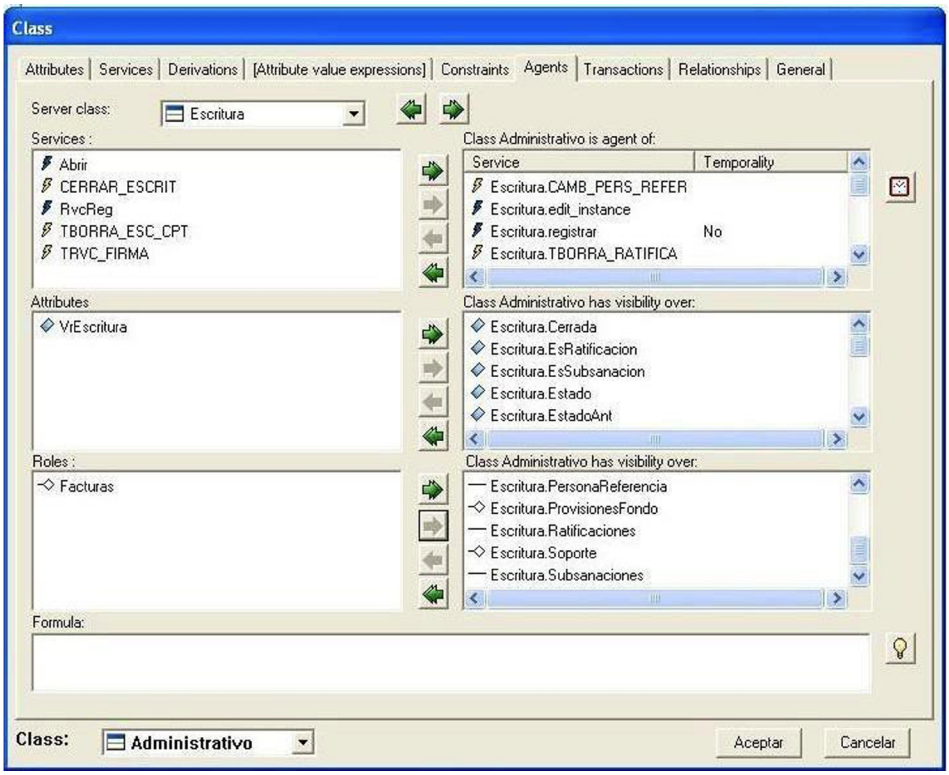


Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

Un *Agente* (también denominado *Actor*) es una clase de objeto especial que representa a un tipo de usuario del sistema. En OO_METHOD un agente puede activar servicios, visualizar atributos y navegar por el sistema. Un actor, por lo general, es una persona que interactúa con el sistema de diferentes formas, pero también pueden ser otros sistemas o un dispositivo que demanda algún servicio. De todas formas, un actor siempre se representará en el diagrama de clases como una clase, que por lo general no posee nexos estructurales con las demás clases del sistema.

La figura 10 muestra en la columna de la izquierda los privilegios que el actor administrativo (*Class*) tiene sobre la clase escritura (*Server Class*), y en la columna de la derecha los privilegios denegados para ese mismo actor.

Figura 10. Asignación de privilegios al agente administrativo sobre la clase escritura implementada en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

En el diagrama de clases (figura 6) se observaron los actores que pueden tener nexos con los servicios y los atributos de una clase, y con la navegación desde la clase actual hacia otras clases. Los privilegios correspondientes se muestran en la figura 10 para el agente administrativo.

En OLIVA NOVA, los privilegios de un actor se pueden asignar y quitar a consideración del desarrollador.

De acuerdo con lo expuesto antes, un nexo agente-servicio se determina a partir de un nexo sujeto-acción (LOM). En el contexto de ONME, un nexo agente-servicio especifica que un agente inicia un servicio perteneciente a una clase del sistema. En el contexto de LOM, de forma similar, un nexo sujeto-acción expresa que un sujeto ejecuta una acción en el sistema. Así que un nexo agente-servicio (ONME) se obtiene

de un nexo sujeto-acción (del LOM). Nótese, adicionalmente, que una acción puede ser activada por varios sujetos y que un sujeto puede activar varias acciones, por lo que este nexo puede tener múltiples instancias.

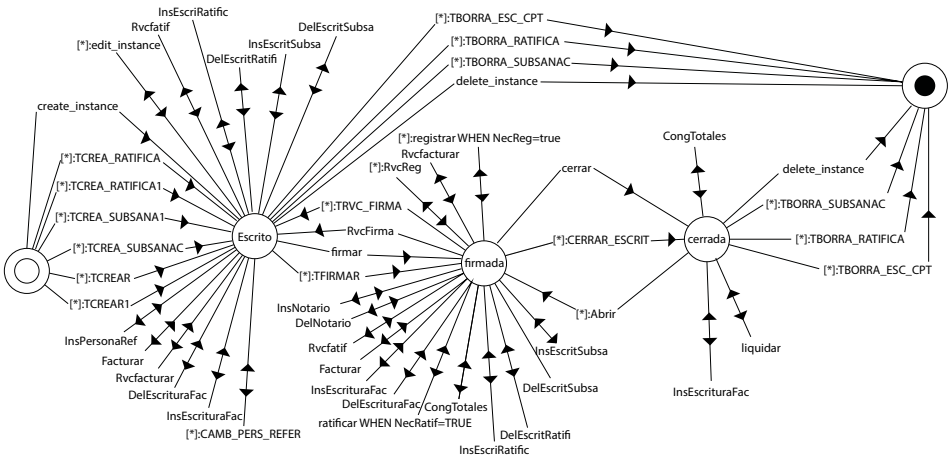
Se ha determinado que un término tipo sujeto del LOM también puede conducir a definir un agente del sistema, aunque no todos los sujetos generen agentes; por ejemplo, un cliente, a quien en la descripción del sistema se le atribuyen acciones, puede que no realice tales acciones en el sistema automatizado (a menos que el sistema lo señale explícitamente - web). Adicionalmente, solo se puede confirmar un agente si en su descripción se reporta que puede activar al menos una acción.

5.1.2 Modelo dinámico

El modelo dinámico de OO-Method expresa el comportamiento de los objetos del sistema. Se representa mediante dos diagramas: el diagrama de transición de estados y el diagrama de interacción entre objetos.

En la figura 11 se muestra el DTE para la clase escritura. Nótese que varias de las transiciones mostradas no generan un cambio de estado, por lo que decimos que es una transición reflexiva. Por lo tanto, como se sabe que las transiciones son causadas por servicios, es posible determinar qué servicios se le permiten ejecutar a un objeto en cada uno de los estados de su ciclo de vida.

Figura 11. Diagrama de transición de estados de la clase escritura implementada en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

Por tanto, un *estado* se define como una situación relevante en la vida de un objeto. En tal situación, a un objeto se le permite ejecutar algunos servicios y se le prohíbe la ejecución de otros. En la figura 11 se observan tres estados relevantes para los objetos de la clase escritura: *escrito, firmada y cerrada*, y dos estados comunes a todo objeto. Los estados comunes son el estado previo a la creación del objeto, o estado de entrada, que se simboliza como un doble círculo, y el estado posterior a su destrucción que se simboliza mediante un círculo relleno circunscrito dentro de una circunferencia, el cual representa el fin de la vida de un objeto.

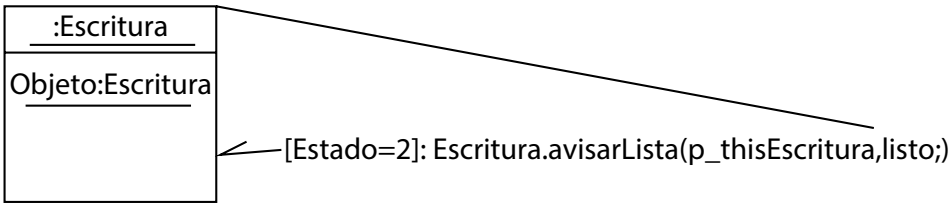
Los estados de un objeto se obtienen de términos LOM de tipo estado, pero no todos los términos LOM de tipo estado se convierten en estados en el DTE de un objeto.

Una *transición entre estados* se define como el paso de un objeto de un estado a otro, causado por algún servicio y posiblemente bajo algunas condiciones. Sin embargo, como se dijo anteriormente, algunas transiciones no generan un cambio de estado, sino que son reflexivas. En la figura 11, por ejemplo, el servicio cerrar (escritura) hace que una escritura pase del estado *firmada* al estado *cerrada*, mientras que el servicio facturar o el servicio congTotales (congelar totales) no causan un cambio de estado efectivo.

Las transiciones entre estados se representan con arcos dirigidos. Los arcos que salen de un estado y llegan a otro estado representan una transición efectiva (cambio de estado), mientras que los arcos que entran y salen del mismo estado (con doble flecha) no representan transiciones efectivas. Los servicios que rotulan los arcos que salen de un estado, indistintamente de si vuelven o no al mismo estado, representan los servicios que un objeto puede ejecutar estando en tal estado.

Un *disparador* en OO-Method representa una acción que se ejecuta de manera automática (sin intervención de un actor) cuando una condición predeterminada se cumple. En la figura 12 se ilustra un disparador de un objeto escritura que especifica que se debe mostrar un mensaje (servicio avisarLista) si se cumple la condición (Estado = 2). Dicho mensaje indica que la escritura ha sido firmada y está lista para ser recogida para continuar su trámite. Aunque actualmente no se ha previsto la identificación de disparadores con base en el LOM, es posible que posteriormente se pueda incluir.

Figura 12. Disparador que activa un mensaje cuando la Escritura llega al estado 2, implementado en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

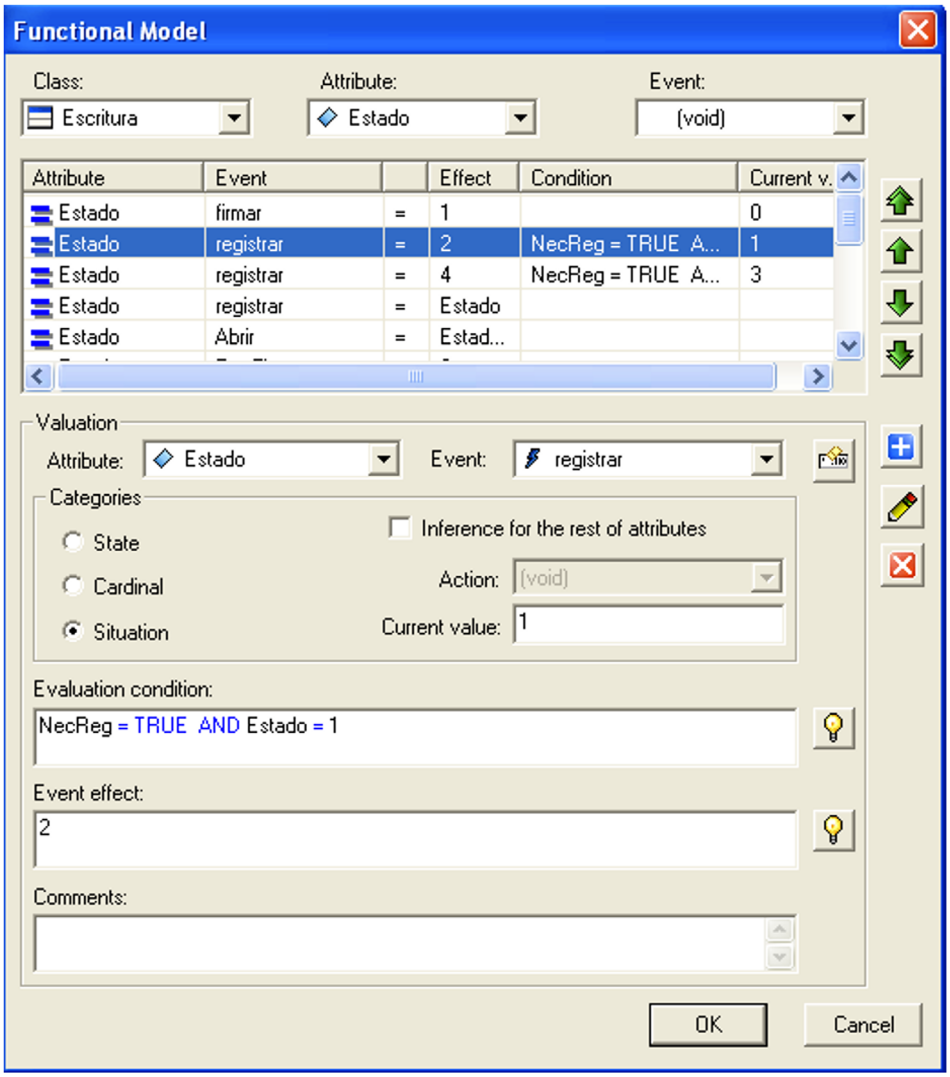
5.1.3 Modelo funcional

El modelo funcional de OO_METHOD especifica la manera como determinados servicios afectan el valor de un atributo. Esto se representa mediante una matriz, como se muestra en la figura 13; en cada fila se muestra la afectación del valor de un atributo por efecto de la acción de un servicio, junto con la condición requerida para que el servicio se lleve a cabo.

Para describir mejor la acción de un servicio sobre un atributo, en OO-Method los atributos se han clasificado en tres tipos: cardinal, estado y situación. El valor de un atributo cardinal se incrementa o se disminuye por la acción de un servicio; el valor de un atributo de estado cambia independientemente de su valor actual y toma sus valores de un dominio limitado; y el valor de un atributo de situación cambia su valor en función de su valor actual. Nótese que este tipo de elementos de modelado son de difícil obtención a partir del LOM, aunque sí se pueden inferir de algunas reglas de negocios particulares.

En la figura 13 se muestra cómo el valor del atributo estado (de categoría situación) es afectado por los servicios firmar, registrar y abrir, dependiendo de su valor actual y de algunas condiciones. La afectación del servicio registrar junto con una de sus condiciones se discrimina en la parte inferior de la ventana.

Figura 13. Especificación del modelo funcional para la clase escritura, implementado en ONME



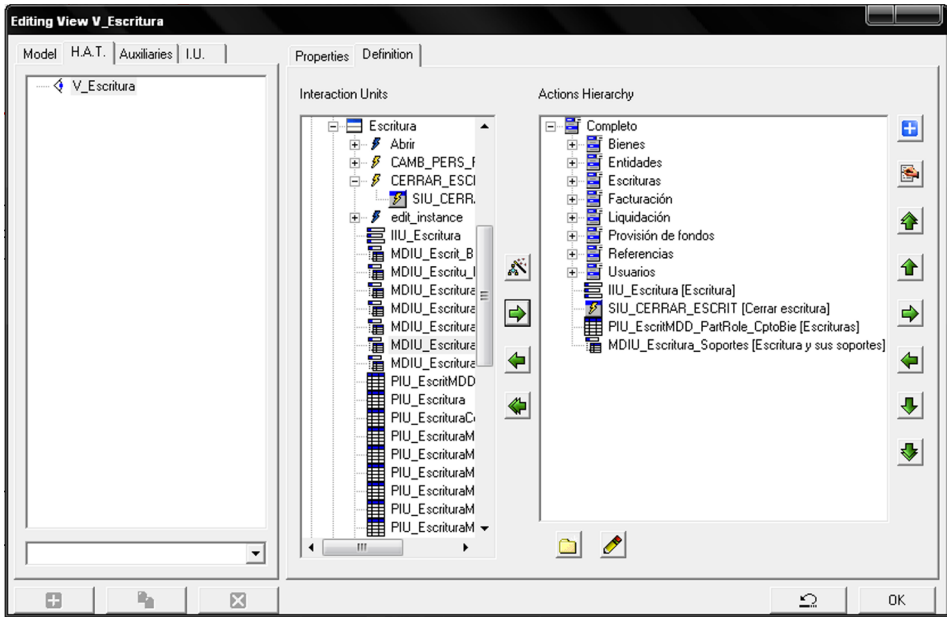
Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*.

5.1.4 Modelo de presentación

El modelo de presentación de OO_METHOD permite definir un conjunto de elementos gráficos (patrones) que contribuyen a construir la interfaz del sistema

con el usuario. Estos elementos se definen con base en el diagrama de clases y en las necesidades de interacción con el sistema que el usuario final requiere. En la figura 14 se muestran los elementos gráficos que constituyen el modelo de presentación asociados con la clase escritura.

Figura 14. Especificación del modelo de presentación relativo a la clase escritura, implementado en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*

Este modelo permite especificar la forma como el usuario va a interactuar con el sistema y provee facilidades para definir elementos que conformarán la interfaz gráfica. Estos elementos se clasifican en cuatro grandes grupos: unidades de interacción de servicios (SIU), unidades de interacción de población (PIU), unidades de interacción de instancia (IIU) y unidades de interacción maestro - detalle (MDIU), como se muestra en la figura 14. Todos los elementos de cada grupo se construyen con un conjunto de elementos (patrones) auxiliares, que también provee OO-Method como se describe a continuación (Molina, 2003).

- Una unidad de interacción de servicio (SIU) utiliza los siguientes elementos auxiliares: introducción, selección definida, agrupación de argumentos, recuperación de estado, dependencia e información suplementaria.

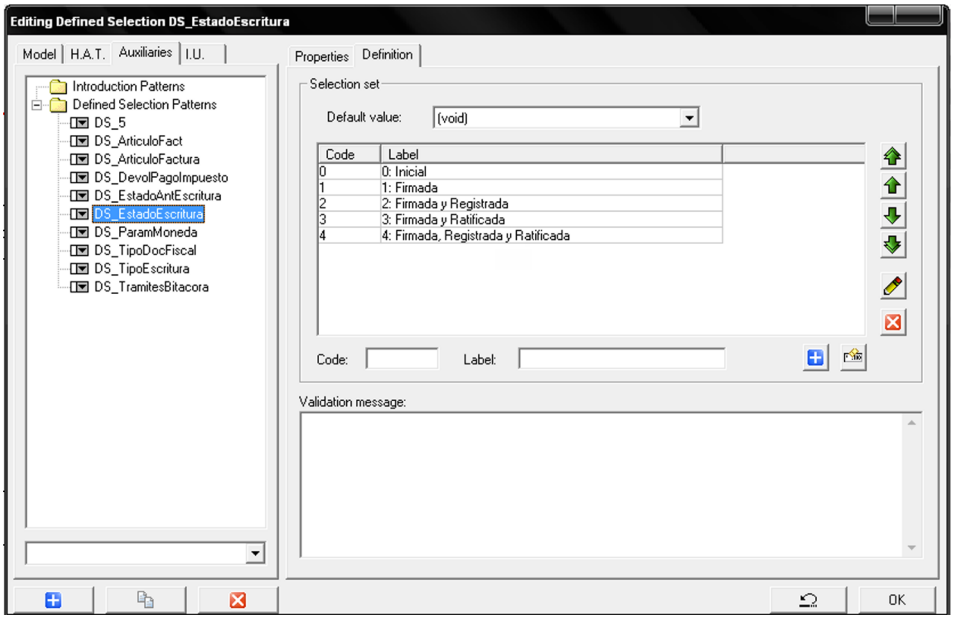
- Una unidad de interacción de población (PIU) utiliza los siguientes elementos auxiliares: filtro, criterio de ordenamiento, conjunto de despliegue, acciones y navegaciones.

- Una unidad de interacción de instancia (IIU) utiliza los elementos auxiliares: conjunto de despliegue, acción y navegación.

- Una unidad de interacción maestro - detalle (MDIU) utiliza los elementos auxiliares: unidad de interacción maestro y unidad de interacción detalle.

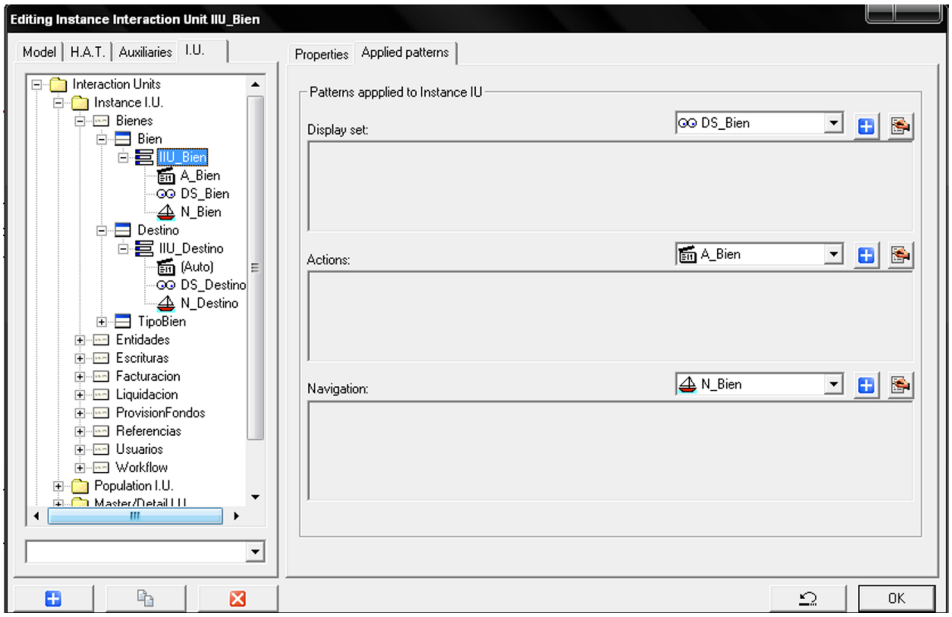
En la figura 15 se presenta la especificación del patrón auxiliar de selección definida *estado de escritura,* y en la figura 16 se muestran los componentes de una unidad de interacción de instancia para un *bien*.

Figura 15. Especificación del elemento auxiliar selección definida implementado en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*

Figura 16. Especificación de una unidad de interacción de instancia usando patrones auxiliares en ONME



Nota: La figura se construyó usando una versión de OLIVA NOVA MODEL EXCECUTION de prueba cedida por la empresa *Computer Aided Requierement*

Una breve descripción de los elementos (patrones) más relevantes que provee OO-Method para la construcción de la interfaz gráfica hombre-máquina se presenta a continuación:

Una *unidad de interacción de servicio* es un elemento conceptual que permite crear una ventana reducida para capturar los parámetros de un servicio que afecta a un único atributo.

Una *unidad de interacción de instancia* es un elemento conceptual que posibilita crear una ventana para capturar los atributos de un objeto determinado.

Una *unidad de interacción de población* permite, además de capturar los atributos de un objeto, mostrar en una matriz de resumen los datos de los objetos del mismo tipo y actualiza inmediatamente las adiciones, eliminaciones y modificaciones realizadas sobre los objetos mostrados en la matriz.

Una ***unidad de interacción maestro - detalle*** facilita la construcción de un objeto gráfico complejo para capturar estructuras agregadas, como una factura con sus respectivas líneas: presenta un área inicial para capturar los datos del objeto principal o maestro (factura) y una matriz en la que cada fila representa un detalle (línea). También es posible definir estructuras maestro con varios detalles o varios niveles de maestro y detalle.

Una ***selección definida*** es una estructura que contiene los posibles valores que puede adoptar un determinado atributo, como el estado civil (casado, soltero, viudo, separado, unión libre). La estructura se asocia a un atributo y permite la selección del valor deseado en tiempo de ejecución.

Un ***filtro*** es una estructura conceptual que permite, mediante una o más condiciones, seleccionar los objetos de una población de objetos que se desean presentar en algún elemento de interfaz gráfica.

Los ***permisos de navegación*** son especificaciones que otorgan permisos al usuario para navegar entre unidades de interfaz a partir de la interfaz actual. Por ejemplo, desde una factura hacia el cliente titular de la factura.

Los ***criterios de ordenamiento*** permiten definir si la información de los objetos se presenta en forma ascendente o descendente e incluso si crea jerarquías anidadas.

Los elementos del modelo de presentación pertenecen a una categoría tan detallada que es casi improbable capturarlos a partir del LEM.

5.1.5 Elementos del modelo conceptual deducibles del LOM

En este apartado se proponen las características y elementos de modelado de las vistas que conforman a OO-Method, con el propósito de determinar cuáles de estos elementos, ya sean elementales o complejos, se pueden obtener mediante el LOM. Los elementos simples son aquellos que se refieren a una sola entidad y los complejos son los que pueden incluir varios nexos o relaciones entre dos o más entidades, siendo una entidad cualquier elemento de modelado.

Bajo estas condiciones, los elementos que se pueden obtener del LOM, para este trabajo, son los siguientes.

Para el *diagrama de clases* se pueden obtener los elementos simples: clase, actor, atributo y servicio; y se pueden obtener los elementos complejos: precondición, agregación, herencia, nexo agente-servicio y nexo atributo-clase.

Para el *diagrama de transición de estados* se puede obtener el elemento simple: estado de un objeto; y los elementos complejos: transición entre estados y servicio que activa una transición.

Ahora bien, como se mostró en capítulo 4, los modelos que se pueden obtener de un método como el LOM son modelos generales que tienen que ser complementados acudiendo a la experticia del ingeniero de software. Esto se debe a que, en esta fase, no es posible determinar todas las características que poseen los elementos del modelo conceptual y a que el nivel de abstracción en el LOM (CIM) es mayor que el del PIM. En esta fase además pueden ocurrir situaciones como, por ejemplo, obtener atributos de una clase, pero que sea difícil establecer su tipo; obtener servicios, pero no el tipo, y que sea difícil establecer a qué le corresponden.

5.2 Oliva Nova Model Execution ONME

OLIVANOVA Model Execution ONME es una implementación de OO-Method. Está constituido por un modelador (OLIVANOVA *Modeler*), varios motores de transformación (OLIVANOVA *Transformation Engines*) (Object Management Group - OMG., 2003) y algunas herramientas complementarias para soportar el ciclo de desarrollo de software.

ONME cubre la mayor parte del ciclo de vida del desarrollo de software. Se le puede considerar como una herramienta MDSD que procura seguir los estándares de MDA. En este sentido, ONME ofrece facilidades para desarrollar software partiendo de un modelo conceptual o CIM, para luego transformar este modelo en especificaciones propias de un entorno específico, valiéndose de la máquina de transformación que equivale a un PSM y a sus transformaciones CIM-PIM y PSM-Código. Adicionalmente, cubre las actividades de mantenimiento y actualización de versiones mediante facilidades que se integran en la herramienta; sin embargo, como la mayoría de las herramientas MDA/MDSD, prácticamente ignoran al CIM.

ONME, cuyo proceso se muestra en la figura 5, está constituido por dos modelos generales: el modelo conceptual y el modelo de ejecución. El modelo conceptual, a su vez, está conformado por el modelo de objetos, el modelo dinámico, el modelo funcional y el modelo de presentación, cada uno de los cuales representa una perspectiva diferente del sistema. El modelo de ejecución define las reglas para: uno,

traducir el modelo conceptual en una especificación formal y dos, para transformar tal especificación en código fuente en diversos entornos de programación.

OLIVANOVA Modeler es definido por sus constructores como una herramienta de edición y validación de modelos conceptuales de OO_METHOD. Las principales funciones de esta herramienta son las siguientes (Molina y Pastor, 2004): soporta las cuatro vistas del modelo conceptual (ver la sección 4.1); soporta el modelado de vistas legadas; asiste la escritura de fórmulas; valida automáticamente cada uno de los modelos; crea modelos a partir de la importación de modelos existentes creados por OLIVANOVA Modeler, modelos creados por herramientas de terceros (en formato XMI (Osis y otros, 2007)) y estructuras de bases de datos; soporta el modelado cooperativo; genera documentación automática de cada modelo; y genera automáticamente métricas para medir el tamaño funcional asociado a un modelo.

Un *OLIVANOVA Transformation Engine* es la implementación de un proceso de compilación de un modelo conceptual de OO-Method. Pueden existir diversas máquinas (engine), una para cada plataforma de implementación, y se caracteriza por lo siguiente (Molina y Pastor, 2004):

- Cada *Transformation Engine* implementa un repositorio del modelo conceptual, un repositorio del modelo de aplicación, un conjunto de correspondencias entre los elementos del modelo conceptual y los elementos del modelo de aplicación, y un conjunto de transformaciones asociadas a cada uno de los elementos del repositorio del modelo de aplicación.

- El repositorio del modelo conceptual es común a todos los *Transformation Engines* y puede cargar un modelo conceptual representado en un archivo XML de intercambio producido por OLIVANOVA *Modeler*.

- Las correspondencias acceden al repositorio del modelo conceptual y se encargan de crear progresivamente elementos del modelo de aplicación en el repositorio correspondiente.

- El repositorio del modelo de aplicación contiene elementos para cada elemento del modelo conceptual: los elementos necesarios para implementar la estrategia de ejecución de aplicaciones en la plataforma software específica del *Transformation Engine* y los elementos requeridos por la plataforma software específica.

- Las transformaciones acceden al repositorio del modelo de aplicación y generan la porción de código de la aplicación correspondiente a cada elemento del repositorio.

En conclusión, ONME posee características necesarias y suficientes para ser considerada una herramienta MDSD, que además es compatible con las directrices propuestas por MDA y que coincide con las tendencias de desarrollo conducidas por modelos.

6. Patrones de Modelado

6.1 ¿Qué es un patrón de modelado?

En esta propuesta se considera patrón de modelado a una estructura gramatical y un conjunto de heurísticas que conducen a identificar elementos de modelado conceptual (PIM) de OO_Method a partir de especificaciones de términos LOM. Un término LOM se especifica mediante un conjunto de unidades gramaticales (ítems), cada una de las cuales puede regirse por la estructura de un patrón de modelado. Por ejemplo, cada nombre de término o alias se especifica mediante un sustantivo o una frase nominal, mientras que un ítem se especifica mediante una frase verbal que describe una característica o relación y, por lo general, obedece a una estructura predefinida.

Para identificar elementos de modelado, cada unidad gramatical se compara con una o más estructuras de patrón; si la unidad gramatical obedece a una estructura de patrón, se examinan las heurísticas del patrón seleccionado y, si aplican, diremos que se ha encontrado un elemento de modelado conceptual que tiene el mismo nombre de su patrón.

Con base en los términos LOM es posible identificar elementos básicos simples y complejos, y elementos complejos adicionales, como se mostró en la tabla 2. Adicionalmente, los patrones simples abstraen características de elementos básicos simples de modelado: clases de objetos, agentes, servicios, actores, estados y atributos. Los patrones complejos abstraen características de elementos de modelado compuestos por varios elementos básicos, tales como un nexo entre clases o una relación de agente-servicio.

Un patrón se especifica mediante su nombre, una estructura gramatical y un conjunto de heurísticas. El nombre indica el tipo de patrón referido; la estructura gramatical consiste en uno o más componentes también gramaticales; y las heurísticas son criterios aplicables para decidir si un término dado se tipifica como un elemento de modelado o no. En las dos siguientes secciones se describen los patrones básicos y complejos catalogados a partir del trabajo de investigación apalancado por Chaparro (2009).

6.2 Patrones inferidos de términos LOM

6.2.1 Patrón clase de objetos

Describe la estructura y las condiciones bajo las cuales un término LOM tipo objeto se puede transformar en una clase de objetos del modelo conceptual.

Estructura: Nombre Común (sustantivo)

Las heurísticas requeridas para decidir si un término LOM tipo objeto se puede transformar o no en una clase de objetos son las siguientes:

- Gramaticalmente el término candidato es representado por un nombre (sustantivo) común de una o más palabras.

- Implica que posee una o más características o atributos que comparte con otros objetos del mismo tipo.

- Al menos uno de sus atributos se puede utilizar como identificador visible de cada uno de los objetos del mismo tipo.

- Desde el punto de vista del ingeniero de software, pudieran asociársele algunas acciones.

- Pudiera poseer nexos con otros términos de tipo objeto.

Se deben cumplir obligatoriamente las tres primeras heurísticas.

Después de aplicar las condiciones antes enunciadas a los términos LOM tipo objeto de la tabla 1, se obtienen, entre otras, las siguientes clases: escritura, fase de escrituración, bien, notario, oficina de registro, gasto e impuesto.

6.2.2 Patrón atributo

Describe la estructura y las condiciones bajo las que un término LOM tipo atributo se puede transformar en atributo de una clase de objetos.

Estructura: Nombre Común (sustantivo)

Las heurísticas requeridas para decidir si un término LOM se puede transformar o no en un atributo son las siguientes:

- Gramaticalmente el término candidato se describe como un nombre (sustantivo) común de una o más palabras.

- Define una característica de un objeto o una clase de objetos.

- No ha sido definido como una clase de objetos (*ciudad de nacimiento* puede ser un atributo de empleado, pero *ciudad sola* puede ser una clase de objetos).

- Es posible determinar un dominio del que puede tomar sus valores. Por ejemplo, para un estudiante universitario, su edad en años puede oscilar en el rango edad>13 y edad <80.

Después de aplicar las anteriores condiciones a los términos LOM tipo atributo de la tabla 1, se obtienen los siguientes atributos: tarifa, valor impuesto, importe, fecha de petición y número de escritura.

6.2.3 *Patrón agente*

Describe la estructura y las condiciones requeridas para que un término LOM tipo sujeto se pueda transformar en un agente.

Estructura: Nombre Común (sustantivo)

Las heurísticas requeridas para decidir si un término LOM tipo sujeto se puede transformar o no en agente del sistema son las siguientes:

- El término representa a un usuario u otro sistema que interactuará con el sistema que se está construyendo (en este caso, por ejemplo, el administrativo podría acceder al sistema de gestión de escrituras para iniciar el trámite de una escritura).

- Aparece en al menos un ítem de impacto, señalando que él inicia o efectúa una acción declarada o no como servicio.

- No ha sido asignado como elemento tipo clase de objetos; si ha sido asignado, se establece claramente que desempeña dos roles diferentes en el sistema (como clase y como agente).

Bajo las anteriores condiciones se transforman en actores del sistema los siguientes términos LOM tipo sujeto de la tabla 1: usuario, funcionario y administrador.

6.2.4 Patrón estado

Describe la estructura y las condiciones requeridas para que un término LOM tipo estado se pueda transformar en un estado de un objeto.

Estructura: Nombre Común + Calificativo

Donde ***Nombre Común*** hace referencia a un objeto previamente asignado a una clase o no, y ***Calificativo*** indica la situación de tal objeto. En ocasiones, el calificativo se presenta como una forma verbal en pasado como, por ejemplo, ***firmada*** del estado ***escritura firmada*** o ***registrada*** de ***escritura registrada***.

Las ***heurísticas*** requeridas para decidir si un término LOM tipo estado puede ser transformado en un estado de un objeto son las siguientes:

- El término representa una situación específica de un objeto o un sujeto (que puede o no haber sido seleccionado previamente como clase o como actor respectivamente).

- Hace referencia a un término de tipo objeto o de tipo sujeto al que califica (Escritura Firmada, cliente deshabilitado) o describe la situación de actividad (Escritura en edición)

Bajo las anteriores condiciones, todos los términos LOM tipo estado de la tabla 1 se transforman en estados de objetos de la clase escritura. Por lo tanto, son estados: escritura firmada, escritura registrada, escritura ratificada, escritura en notaría, escritura en registro, escritura finalizada.

Por otro lado, un elemento tipo estado está asociado a un objeto de una clase y a un conjunto de servicios que entran o salen de él.

6.2.5 Patrón servicio

Describe la estructura y las condiciones requeridas para que un término LOM tipo acción se pueda transformar en un servicio.

Estructura: Verbo infinitivo + Nombre Común

Donde ***Verbo Infinitivo*** indica la acción, y ***Nombre Común*** es el objeto sobre el que recae la acción.

Las ***heurísticas*** requeridas para decidir si un término LOM tipo acción se puede transformar o no en un servicio son las siguientes:

- El término representa una acción concreta.

- Sería posible determinar el sujeto (o agente) que es responsable de tal acción.

- Podría determinarse a qué objeto o clase de objetos le corresponde tal acción.

Bajo las anteriores condiciones, los términos LOM tipo acción que se transforman en servicios son los siguientes: liquidar gasto, registrar escritura, liquidar impuesto, firmar escritura, pagar impuesto y solicitar trámite (de escritura).

6.3 Patrones inferidos de ítems del LOM

La mayor parte de ítems de términos LOM se rigen por patrones complejos. La tabla 2 muestra los tipos de ítems usados para especificar los diferentes tipos de términos LOM. Cada tipo de ítem, excepto el de tipo definición, se rige por un patrón que permite identificar un elemento de modelado complejo, ya sea básico o adicional, como se muestra a continuación.

6.3.1 Patrón nexo entre clases

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo entre entidades se pueda transformar en nexo entre clases (asociación, agregación, herencia).

Estructura: Término-O + frase verbal + Término-O

Donde ***Término-O*** se refiere a un término tipo objeto, y ***frase verbal*** es una frase que indica el nexo entre los términos tipo objeto (o las clases, si ya han sido seleccionados como tal).

Las heurísticas requeridas para decidir si un ítem de nexo entre entidades se puede o no transformar en nexo entre clases son las siguientes:

- El ítem incluye al menos dos términos de tipo objeto o dos términos de tipo sujeto, enlazados por una frase verbal.

- La probabilidad es mayor si los términos ya han sido asignados como clases de objetos o actores.

- La frase verbal representa una relación estática, nunca una acción, aunque esté escrita indicando una supuesta acción.

- Si la frase verbal empieza con la palabra ***incluye*** o las palabras ***es parte de*** o las palabras ***está conformado/a por***, el nexo determina una relación de agregación.

- Si la frase verbal contiene las palabras ***es un*** o ***es una***, el nexo determina una relación de herencia.

- En los demás casos diferentes a los casos anteriores, la frase verbal determina una relación de asociación.

Si alguno de los términos enlazados por la frase verbal mencionada antes está en plural, indica que la cardinalidad es múltiple en el lado del objeto o clase correspondiente con el término en plural. Por ejemplo, el ítem tipo nexo del término LOM Escritura: una escritura (Ob.1) es elaborada por un notario (Ob.14), indica que existe una asociación entre las clases escritura y notario, cuya cardinalidad en el lado de notario es uno y la otra cardinalidad se debe inferir a partir del dominio del negocio. De otro lado, en este mismo término, el ítem: una escritura (Ob.1) incluye uno o más negocio/s (Ob.4), señala que existe una agregación entre las clases escritura y negocio, es decir que una escritura es una agregación de negocios.

6.3.2 *Patrón relación agente-servicio*

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo sujeto-acción se pueda transformar en una relación agente-servicio.

Estructura: Término-Su + frase verbal + Término-A / Término-A + frase verbal + Término-Su

Donde ***Término-Su*** indica un término tipo sujeto, ***Término-A*** se refiere a un término tipo acción y la ***frase verbal*** establece cuál sujeto ejecuta la acción. Los elementos correspondientes a este tipo de patrón por lo general están presentes en ítems de términos LOM de tipo sujeto y en algunos casos en ítems de tipo acción.

Las heurísticas requeridas para decidir si un ítem de tipo nexo sujeto-acción se puede o no transformar en una relación agente-servicio son las siguientes:

- El ítem incluye un término de tipo acción y un término de tipo sujeto enlazados por una frase verbal.

- La probabilidad de que el ítem sea una relación agente-servicio es mayor si el término tipo sujeto ya ha sido asignado como agente del sistema o si el término tipo acción ya ha sido asignado como servicio de una clase.

- La frase verbal especifica el nexo indicando qué o quién efectúa la acción.

- La frase verbal, aunque parezca lo contrario, nunca configurará una relación estructural (nexo) entre clases.

- Y finalmente, solo se consideran acciones que inicia el sujeto en el sistema automatizado, nunca acciones que realiza el presunto sujeto fuera de tal sistema.

Los elementos que obedecen a este patrón por lo general están presentes en los ítems de impacto de términos tipo sujeto y en ítems de noción de términos tipo acción. Por ejemplo, el ítem del término tipo acción Registrar_Escritura: la acción de registrar escritura (A2) es efectuada por un administrativo (Su1), conduce a inferir la relación agente-servicio entre el actor ***administrativo*** y el servicio ***registrar escritura***, que previamente debieron haber sido definidos como tal.

6.3.3 Patrón transición entre estados

Describe la estructura y las condiciones requeridas para que un ítem de tipo transición entre estados se pueda transformar en una transición entre estados de un objeto de una clase.

Estructura: Término-A + frase verbal + Término-O + "de" + Término- E+"a"+Término-E

Donde ***Término-A*** indica un término tipo acción que causa la transición, ***Término-O*** se refiere a un término tipo objeto, ***Término-E*** indica un término tipo estado y la ***frase verbal*** señala que hay un cambio de estado. Elementos correspondientes a este tipo de patrón están presentes en ítems de términos LOM de tipo acción.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo transición entre estados se puede o no transformar en una transición entre estados de objetos de una clase son las siguientes:

- El ítem incluye un término de tipo acción que causa la transición, un término de tipo objeto, que es el que indica el objeto que cambia de estado, y dos términos de tipo estado que representan los estados origen y destino.

- La probabilidad de que el ítem sea un/a (elemento de) transición entre estados es mayor si los términos mencionados en el numeral anterior ya han sido asignados a elementos de modelado del tipo respectivo. Es decir, servicio, clase y estados.

- La frase verbal especifica un cambio de estado.

Los elementos que obedecen este patrón están presentes en los ítems de impacto de términos tipo acción. Por ejemplo, el ítem en el término tipo acción Registrar_Escritura: registrar escritura (A2) cambia el estado de la escritura (Ob1) de escritura firmada (E1) a escritura registrada (E2), indica la transición, los estados y la acción que genera la transición.

6.3.4 Patrón precondición

Describe la estructura y las condiciones requeridas para que un ítem de tipo restricción de acción se pueda transformar en precondición de un servicio.

Estructura: "Para" + Término-A + frase verbal + Condición

Donde ***Término-A*** indica un término tipo acción sobre el que se aplica la condición, la ***frase verbal*** indica el nexo con la condición y la ***condición*** puede incluir términos tipo objeto o términos tipo atributo o términos tipo estado o términos tipo acción.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo restricción de servicio se puede o no transformar en una precondición sobre un servicio son las siguientes:

- El ítem incluye un término de tipo acción que es sobre el que se aplica la condición.

- La condición indica que esta se debe cumplir con carácter obligatorio.

- La probabilidad es mayor si el Término-A ya ha sido asignado como servicio.

- La frase verbal puede incluir palabras como —es obligatorio, —es imprescindible o —es necesario.

Los elementos que obedecen este patrón están presentes en los ítems de impacto de términos tipo acción. Por ejemplo, el ítem, en el término tipo acción Registrar Escritura: Para registrar escritura (A2) es obligatorio que todos los impuestos (Ob17) estén pagados, indica la condición que se debe cumplir para que se pueda ejecutar la acción de registrar escritura.

6.3.5 *Patrón nexo servicio-clase*

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo acción-objeto se pueda transformar en un nexo servicio-clase (servicio de una clase/ acción).

Estructura: El/La + Término-O + frase verbal + (Término-A)

Donde ***Término-O*** indica un término tipo objeto sobre el que recae la acción, ***Término-A*** representa la acción tal y como está escrita en la lista de términos LOM, y la frase verbal describe la acción escrita en la forma normal del lenguaje español.

Las heurísticas requeridas para decidir si un ítem de tipo nexo acción-objeto se puede o no transformar en un nexo servicio-clase son las siguientes:

- El ítem incluye un término de tipo objeto que es sobre el que recae la acción.

- La acción se describe en la frase verbal y se escribe entre paréntesis como término tipo acción, de igual forma que como está escrito en la lista de LOM.

- La probabilidad es mayor si el Término-O y el Término-A ya han sido asignados como clase y servicio respectivamente.

- En algunos casos puede incluir un término tipo sujeto o un término tipo objeto, que no son de interés para este patrón.

Los elementos que obedecen este patrón están presentes en los ítems de impacto de términos tipo objeto. Por ejemplo, el ítem en el término tipo objeto Escritura: la escritura (Ob.1) se registra (registrar escritura (A.2)), indica que la acción registrar recae sobre el objeto escritura.

6.3.6 *Patrón nexo atributo-clase*

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo atributo-objeto se pueda transformar en un nexo atributo-clase (atributo de clase). Su estructura es:

Estructura: "El/La/Un/Una + Término-O + frase verbal + Término-At".

Donde ***Término-At*** indica un término tipo atributo, ***Término-O*** es un término tipo objeto y la ***frase verbal*** especifica que el objeto tiene un atributo.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo nexo atributo-objeto se puede o no transformar en un nexo atributo-clase son las siguientes:

- El ítem incluye un término de tipo atributo y un término tipo objeto.

- La frase verbal establece una relación de pertenencia entre el atributo y el objeto. El atributo es una característica del objeto.

- La probabilidad es mayor si el Término-O y el Término-At ya han sido asignados como clase y atributo respectivamente.

Los elementos que obedecen a este patrón están presentes en los ítems de noción de términos tipo atributo. Por ejemplo, el ítem en el término tipo objeto Valor Impuesto: una escritura (Ob.1) tiene un número (At.5), indica que el atributo es de la clase de objetos escritura.

6.3.7 Patrón nexo servicio-atributo

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo acción-atributo se pueda transformar en un nexo servicio-atributo (servicio que afecta el valor del atributo).

Estructura: Término-A + frase verbal + Término-At

Donde ***Término-At*** indica un término tipo atributo, ***Término-A*** es un término tipo acción y la ***frase verbal*** especifica la afectación de la acción sobre el atributo.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo nexo acción-atributo se puede o no transformar en un nexo servicio-atributo son las siguientes:

- El ítem incluye un término de tipo atributo y un término tipo servicio.

- La frase verbal establece cómo el atributo es afectado por la acción.

- La probabilidad es mayor si el Término-A y el Término-At ya han sido asignados como servicio y atributo de una clase respectivamente.

Los elementos que obedecen a este patrón están presentes en los ítems de impacto de términos tipo atributo. Por ejemplo, el ítem en el término tipo atributo Valor Impuesto: liquidar impuesto (A7) modifica el valor impuesto (At2), indica que el valor del atributo es modificado por la acción.

6.3.8 *Patrón nexo estado-clase*

Describe la estructura y las condiciones requeridas para que un ítem de tipo nexo estado-entidad se pueda transformar en un nexo estado-clase (estado de objetos de una clase).

Estructura: Término-E + frase verbal + Término-O

Donde ***Término-E*** indica un término tipo estado, ***Término-O*** es un término tipo objeto y la ***frase verbal*** especifica la correspondencia entre los dos términos.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo nexo estado-entidad se puede o no transformar en un nexo estado-clase son las siguientes:

- El ítem incluye un término de tipo estado y un término tipo objeto.

- La frase verbal establece a qué objeto le corresponde el estado.

- La probabilidad es mayor si el Término-E y el Término-O ya han sido asignados como estado y clase de objetos respectivamente.

Los elementos que se rigen por este patrón están presentes en los ítems de noción de términos tipo atributo. Por ejemplo, el ítem en el término tipo estado—escritura firmada: Escritura Firmada (E1) es un estado de una escritura(Ob1), indica que Escritura posee un estado denominado ***escritura firmada***.

6.3.9 *Patrón nexo estado-servicio*

Describe la estructura y las condiciones necesarias para que un ítem de tipo nexo estado-acción se pueda transformar en un nexo estado-servicio (servicios posibles en este estado).

Estructura: Término-A + frase verbal + Término-E

Donde ***Término-E*** indica un término tipo estado, ***Término-A*** es un término tipo acción y la ***frase verbal*** especifica la correspondencia entre los dos términos.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo nexo estado-acción se puede o no transformar en un nexo estado-servicio son las siguientes:

- El ítem incluye un término de tipo acción y un término tipo estado.

- La frase verbal establece que en el estado especificado puede ocurrir la acción.

- La probabilidad es mayor si el Término-A y el Término-E ya han sido asignados como acción y estado de una clase de objetos respectivamente.

Los elementos que se rigen por este patrón están presentes en los ítems de impacto de términos tipo estado. Por ejemplo, el ítem en el término tipo estado Escritura Firmada: registrar escritura (A2) ocurre a partir de escritura firmada (E1), indica que es permitido efectuar la acción desde el estado ***escritura firmada***.

6.3.10 Nexo servicio-estado

Describe la estructura y las condiciones necesarias para que un ítem de tipo nexo acción-estado se pueda transformar en un nexo servicio-estado (servicio que conduce a un estado). Su estructura es:

Estructura: Término-A + frase verbal + Término-E

Donde ***Término-E*** indica un término tipo estado, ***Término-A*** es un término tipo acción y la ***frase verbal*** especifica la correspondencia entre los dos términos.

Las ***heurísticas*** requeridas para decidir si un ítem de tipo nexo acción-estado se puede o no transformar en un nexo servicio-estado son las siguientes:

- El ítem incluye un término de tipo acción y un término tipo estado.

- La frase verbal establece que la acción prevista produce que el objeto adopte el estado especificado.

- La probabilidad es mayor si el término-A y el término-E ya han sido asignados como acción y estado de una clase de objetos respectivamente.

Los elementos que se rigen por este patrón están presentes en los ítems de impacto de términos tipo estado. Por ejemplo, el ítem en el término tipo estado Escritura Firmada: firmar escritura (A5) conduce a escritura firmada (E1), indica que la acción conduce a que el objeto escritura asuma el estado *escritura firmada*.

Con respecto a los dos últimos patrones, si un servicio A se puede ejecutar en un determinado estado E, y el mismo servicio A conduce al mismo estado E, entonces el servicio tiene carácter reflexivo, porque gráficamente (DTE) su arco sale de un estado y vuelve a ese mismo estado.

6.4 Lenguaje acotado

Recordemos que el LOM aplica los principios fundamentales de circularidad y de mínimo vocabulario en toda su extensión, inspirado en el LEL de Leite (Kaplan et al., 2000). El principio de circularidad está garantizado cuando en la definición de un término LOM se usan otros términos LOM. El principio de mínimo vocabulario se implementa cuando se define una estructura concreta para especificar cada tipo de ítem de un término. Sin embargo, tales estructuras hacen uso de un conjunto de expresiones externas que es necesario precisarlas para minimizar su impacto en la descripción de cada término.

Cada tipo de término utiliza diversos tipos de ítems que cumplen funciones preestablecidas en su especificación. En esta medida, las expresiones que se señalan en la tabla 3 conforman lo que aquí se denomina lenguaje natural acotado (lenguaje acotado). Sin embargo, los ítems de definición que son comunes a todos los tipos de términos y que aparecen en primer lugar en la noción, por su naturaleza, se tienen que definir con cualquier expresión del lenguaje natural, es decir, no obedecen al uso de lenguaje acotado.

Nótese que algunas expresiones de la tabla 3 no están completamente determinadas, sino que requieren un complemento que se puede ver en la estructura del tipo de ítem correspondiente. Este es el caso de la expresión *–es –verbo pasivo–*, donde *verbo pasivo* se refiere a la acción ejecutada sobre el objeto, indicada por un verbo conjugado en pasivo (ver 4.3 ejemplo 1). Otros casos como este se pueden confrontar con los ejemplos para tener una mejor visión de la aplicación de la expresión.

Tabla 3. Lenguaje acotado y su relación con los elementos de modelado

TÉRMINO	EXPRESIÓN		ELEMENTO DE MODELADO
	NOCIÓN	IMPACTO	
Objeto	Frase verbal / -es de		Agregación referencial
	-incluye una o más, -parte de		Agregación inclusiva
	-es un/a		Generalización (herencia)
	-tiene un/a		Pertenencia atributo
		-es - verbo pasivo - -por un/a	Acción de una clase Relación agente-servicio
Atributo		-reemplaza el, -incrementa el -decrementa el	Afectación funcional acción-atributo
	-no puede superar -no puede ser inferior a – y -varía entre --- y ---		Restricción dominio de valor-atributo
Sujeto	-es un/a		Generalización (herencia)
	-tiene		Pertenencia atributo
		-efectúa la acción de	Agente de servicio
Acción	-la acción de… + -es efectuada por		Agente de servicio
		-cambia el estado de la/el… + de… + a…	Asociación servicio transición
		-es obligatorio que --- -se debe hacer ---	Precondición de servicio
Estado	-es un estado de un/a		Pertenencia estado
		-conduce a	Transición a objeto actual
		-ocurre a partir de	Transición a próximo estado

6.4.1 Expresiones para términos tipo objeto

Los términos de tipo objeto poseen ítems de nexo entre clases y de nexo acción-objeto. Cada tipo de nexo utiliza las siguientes expresiones que entran a conformar parte del lenguaje acotado.

Los ítems de nexo entre objetos utilizan las expresiones –es de / frase verbal–, –incluye una o más– y –hace parte de–, para establecer una relación de agregación. Por ejemplo: una escritura (Ob.1) es solicitada por (es de) un cliente (Ob.20) y una escritura (Ob.1) incluye uno o más negocio/s (Ob.4).

Los ítems de nexo atributo-objeto utilizan la expresión –tiene un/a–, para establecer una relación de pertenencia del atributo. Por ejemplo: una escritura (Ob.1) tiene un número (At.5).

Los ítems de nexo atributo-objeto utilizan las expresiones –es - verbo pasivo más - por un/a–, para establecer una acción que recae sobre un objeto y (opcionalmente) qué sujeto realiza la acción. Por ejemplo: una escritura (Ob.1) es registrada (registrar escritura (A.2)) por un administrativo (Su.2), y escritura (Ob.1) es firmada (firmar escritura (A.5)).

6.4.2 Expresiones para términos tipo atributo

Los términos de tipo atributo poseen ítems de nexo acción-atributo. Cada tipo de nexo utiliza las siguientes expresiones que entran a conformar parte del lenguaje acotado.

Los ítems de nexo acción-atributo utilizan la expresión –reemplaza el, –incrementa el y –decrementa el, más un término tipo –con/en el valor– más el valor. Por ejemplo: liquidar impuesto (A7) reemplaza el valor impuesto (At2) con el valor calculado.

Los ítems de restricción-valor-atributo utilizan la expresión –no puede superar–, –no puede ser inferior a ~ y –varía entre ~ y ~ – y puede incluir otro atributo de referencia. Por ejemplo: el valor impuesto (At2) no puede superar el 10 % de valor negocio (At3).

6.4.3 Expresiones para términos tipo sujeto

Los términos de tipo sujeto poseen ítems de nexo entre entidades, atributo-sujeto, nexo sujeto-acción. Cada tipo de nexo utiliza las siguientes expresiones que entran a conformar parte del lenguaje acotado.

Los **ítems** de nexo entre entidades utilizan la expresión –es un/a– para establecer una relación de herencia (generalización). Por ejemplo: un funcionario (su2) es un usuario (Su1).

Los ítems de nexo sujeto-acción utilizan la expresión –tiene– para establecer una relación de pertenencia. Por ejemplo: un funcionario (su2) tiene un código (At1).

Los ítems de nexo atributo-sujeto utilizan la expresión –efectúa la acción de– para establecer el sujeto que activa una acción. Por ejemplo: un funcionario (Su2) efectúa la acción de registrar escritura (A2).

6.4.4 Expresiones para términos tipo acción

Los términos de tipo acción poseen ítems de nexo sujeto-acción, transición entre estados y precondición. Cada tipo de nexo utiliza las siguientes expresiones que entran a conformar parte del lenguaje acotado.

- Los ítems de nexo sujeto-acción utilizan las expresiones –la acción de...– más –es efectuada– para establecer el sujeto que activa la acción. Por ejemplo: la acción de registrar escritura (A2) es efectuada por un administrativo (Su1).

- Los ítems de nexo estado-estado utilizan las expresiones cambia el estado de la/el... de... a...- para establecer una transición entre estados. Por ejemplo: registrar escritura (A2) cambia el estado de la escritura (Ob1) de escritura firmada (E1) a escritura registrada (E2).

- Los ítems de nexo restricción-acción utilizan las expresiones -es obligatorio que ~, -se debe hacer ~ para establecer que una condición previa (precondición) se debe cumplir para poder ejecutar la acción. Por ejemplo: para registrar escritura (A2) es obligatorio que todos los impuesto/s (Ob17) estén pagados, o registrar escritura (A2) se debe hacer dentro de los 30 días después de haber realizado la acción firmar escritura (A2).

6.4.5 Expresiones para términos tipo estado

Los términos de tipo estado poseen ítems de nexo estado-entidad, estado-acción y acción estado. Cada tipo de nexo utiliza las siguientes expresiones que entran a conformar parte del lenguaje acotado.

- Los ítems de nexo estado-entidad utilizan la expresión es un estado de un/a- para establecer la pertenencia de un estado a un objeto. Por ejemplo: escritura firmada (E1) es un estado de una escritura (Ob1).

- Los ítems de nexo acción-estado utilizan la expresión conduce a para establecer que una acción está ligada a una transición que lleva al objeto al estado actual. Por ejemplo: firmar escritura (A5) conduce a escritura firmada (E1).

- Los ítems de nexo estado-acción utilizan la expresión -ocurre a partir de- para establecer que una acción está ligada a una transición que lleva al objeto al estado actual. Por ejemplo: escritura firmada (E1) ocurre a partir de registrar escritura (A2).

Como se puede ver, las diversas expresiones del lenguaje acotado se elaboran principalmente alrededor de las frases verbales utilizadas para especificar los diversos tipos de ítems.

La estrategia de acotar el lenguaje natural utilizado en la definición de términos es útil en cuanto facilita al ingeniero de requisitos la actividad de especificación (de ítems), facilita la identificación de nexos entre entidades y prepara el camino para descubrir elementos de modelado, especialmente relaciones.

7. Proceso para Generar el Modelo Conceptual

La generación de elementos de modelado a partir de requisitos funcionales y con la asistencia del LOM requiere un proceso disciplinado y sistemático. El modelo de proceso que se muestra en la figura 17 permite producir un conjunto de elementos de modelado que posteriormente serán utilizados para generar el modelo conceptual inicial del sistema (Chaparro, 2009) en términos de OO- METHOD. Las etapas que implica este proceso se describen a continuación.

7.1 Obtención de los Requisitos de Usuario RU

A partir de un diálogo fluido entre el ingeniero de software y el ***stakeholder*** (usuario), se obtienen la descripción actual del sistema (dominio del negocio) y los requisitos funcionales (requisitos de usuario RU) para el nuevo sistema. Estos últimos se refinan de manera que expresen la perspectiva del ingeniero de software, es decir, lo que

Sommerville (2005) denomina Requisitos del Sistema. La descripción del sistema y los requisitos funcionales sirven de entrada a la siguiente etapa.

7.2 Lenguaje común / lista de términos LOM

El lenguaje común o el lenguaje del dominio del problema está conformado por un conjunto de términos que el cliente repite con frecuencia o términos importantes del sistema. A partir de este se identifican los términos relevantes y se separan del vocabulario meramente común. Con los términos relevantes se conforma la primera lista de términos LOM que supone la aplicación de un procedimiento del LEL propuesto por Hadad (Kaplan et al., 2000). Sin embargo, es probable que esta primera lista no esté libre de imprecisiones, pues puede incluir términos desclasificados o que realmente no son términos LOM, o incluso pueden faltar algunos términos.

7.3 Refinamiento de la lista de términos LOM

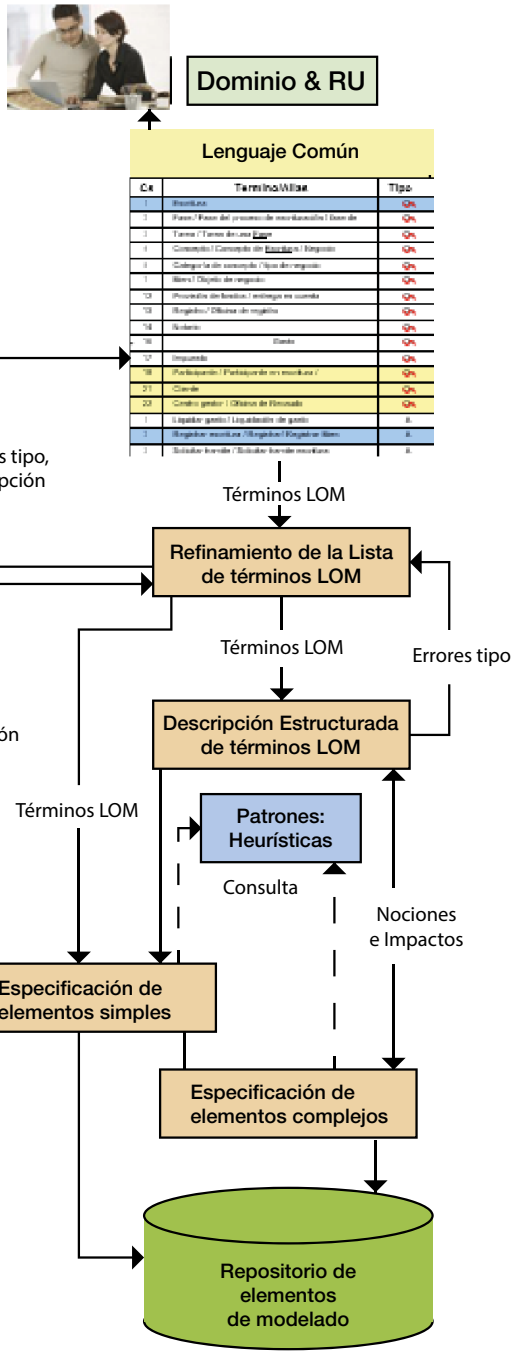
Con base en la primera lista de términos LOM se genera una segunda lista mucho más depurada. Los términos de esta nueva lista[2] deben tener características que indican que existe una importante probabilidad de ser definidos como elementos simples de modelado (teniendo en cuenta lo dispuesto en los capítulos 5 y 6). Para esto se centra la atención en el tipo de cada término, pero se desechan los términos que solo tienen utilidad como componentes del lenguaje común entre el cliente y el ingeniero de software. En esta etapa puede surgir la necesidad de redefinir algunos términos o de cambiar sus tipos.

7.4 Descripción estructurada de términos LOM

A partir de la lista depurada de términos LOM se describe cada término siguiendo las pautas de estructuración previstas para cada tipo de ítem de noción y de impacto (capítulo 6). Un análisis detallado de la descripción del término puede indicar que es necesario cambiar su tipo; por ejemplo, algunos términos de tipo sujeto pueden ser cambiados a términos de tipo objeto.

[2] Aunque esta lista de términos no es relevante para este proceso, sí lo es para el ingeniero, pues le permite tener un mejor conocimiento del lenguaje del dominio del negocio.

Figura 17. Proceso de requisitos para producir elementos de modelado con el apoyo del LOM



Fuente: Los autores

7.5 Asignación y especificación de elementos simples de modelado

Se seleccionan los elementos pertinentes de modelado con base en los términos LOM, previa verificación de las heurísticas previstas en el respectivo patrón. La asignación se hace así: si el término es de tipo objeto, clase de objetos; si es de tipo atributo, atributo de clase; si es de tipo acción, servicio; si es de tipo sujeto, actor; si es de tipo estado, estado; y si es de tipo atributo, atributo. Posteriormente, se especifica (en el sistema) cada elemento de acuerdo con las características de su tipo. Aquí, las heurísticas pudieran indicar que existen términos LEM que no corresponden con elementos de modelado o que sus tipos no son los correctos. En el primer caso se desechan y en el segundo se cambia el tipo del término LOM.

7.6 Asignación y especificación de elementos complejos de modelado

A partir de los ítems de noción e impacto de cada término se verifican la estructura y las heurísticas propias del patrón aplicable, para obtener elementos complejos de modelado. En seguida, se especifica (en el sistema) cada elemento de acuerdo con su tipo; por ejemplo, un elemento agregación requiere que indique las clases relacionadas, el nombre de la asociación y sus cardinalidades. Por otro lado, las heurísticas pudieran indicar que algunos ítems no corresponden con elementos de modelado o que no corresponden con los patrones previstos en su definición, y por tanto deben ser desechados o pueden cambiar de tipo.

7.7 Repositorio de elementos de modelado

Con los elementos simples y complejos de modelado se conforma un repositorio (BD) que se utiliza posteriormente para generar, de manera automática, el modelo conceptual preliminar del sistema.

8. Aproximaciones Similares

Existen diversas aproximaciones gramaticales orientadas a generar modelos del sistema a partir de técnicas gramaticales. Las primeras propuestas, antes de MDA, perseguían producir modelos orientados a objetos de procesamiento de lenguaje natural con bases técnicas de Inteligencia Artificial. Posteriormente, con la aparición de

MDA y MDSD, empezaron a aparecer trabajos cuyo propósito es obtener modelos de requisitos en el marco de MDA, esto es Modelos Independientes de la Computación (CIM), y resultaron algunos trabajos que han propuesto técnicas gramaticales para generar CIM.

8.1 Generación de modelos OO a partir del lenguaje natural

La idea de obtener modelos Orientados a Objetos (OO) de un sistema a partir de la aplicación de técnicas gramaticales ha sido objeto de diversos estudios. A continuación se enuncian algunas propuestas tomadas de Zapata et al. (2006), que poseen alguna similitud con el presente trabajo.

- Barr (*Identificación de patrones de especificación en el bosquejo del sistema de tiempo real Fallstudle*) propone detectar patrones a partir de especificaciones en lenguaje natural no estructurado para transformarlas en especificaciones en lenguaje de especificación formal. El proceso de transformación es facilitado con la utilización de un lenguaje semiformal que sirve como mediador entre las especificaciones natural y formal.

- Juristo et al. (1999) presentan un proceso cuya entrada es una especificación en lenguaje natural y cuya salida es un modelo orientado a objetos del sistema. La especificación inicial se conforma identificando sinónimos y homónimos, y clasificando el discurso en estático y dinámico, para luego transformarla en el modelo orientado a objetos usando como mediador un lenguaje de utilidad.

- Bryant (2000) propone una notación basada en lenguaje natural y en una teoría denominada "gramática de dos niveles". En este método, las especificaciones pueden ser convertidas en diseños orientados a objetos y su proceso puede ser automatizado.

- Fliedl et al. (2000) proponen especificar los requisitos en dos fases: una especificación inicial en lenguaje natural, que se transforma en una especificación intermedia denominada "modelo de prediseño conceptual Klagenfurt", KCPM (por sus siglas en inglés). A partir de este último se puede obtener el diseño orientado a objetos utilizando un conjunto de reglas.

Por otro lado, el Lenguaje de Léxico Extendido - LEL es una estrategia basada en el lenguaje natural para ayudar en el proceso de requisitos, la cual no usa técnicas

de I.A., sino que estructura un conjunto limitado del lenguaje natural. El LEL es una propuesta de Leite, basada en la acotación del lenguaje del dominio mediante la identificación y la especificación de símbolos[3] particulares del sistema (Kaplan et al., 2000).

El LEL limita el lenguaje del dominio del negocio alrededor de un conjunto de términos relevantes del dominio del problema. Es un método de elicitación[4] de requisitos que identifica el lenguaje del dominio que funciona como lenguaje común entre el ingeniero de software y los ***stakeholders***. Concretamente, en la especificación de símbolos se aplican dos principios fundamentales: mínimo vocabulario y circularidad. El principio de mínimo vocabulario dispone que los símbolos del LEL se deban definir utilizando un conjunto restringido y preferentemente predeterminado de términos del lenguaje natural no pertenecientes al LEL. El principio de circularidad establece que un símbolo dado se debe describir usando preferiblemente otros términos LEL y evitando, hasta donde sea posible, términos externos al mismo.

8.2 Modelos independientes de la computación. Construcción y transformaciones

La tendencia de construir Modelos Independientes de la Computación CIM y transformarlos en modelos independientes de la plataforma apenas está comenzando. De hecho, todavía no hay un acuerdo concreto sobre qué constituye un CIM; sin embargo, algunas aproximaciones apuntan a señalar que el modelo del proceso del negocio, el modelo de clases, el modelo de requisitos y el modelo del entorno del negocio son los principales componentes. En lo que todos están de acuerdo es en que el modelo de requisitos es el modelo obligado del CIM, como se puede concluir de los casos que se presentan enseguida.

8.2.1 Modelado independiente de computación con MDA

Osis (2004) presenta la propuesta TFM4MDA, la cual se fundamenta en el análisis formal del sistema de negocios y se centra en lo que el usuario necesita, más que en lo que quiere, lo que permite el chequeo de requisitos textuales y la identificación de requisitos faltantes.

[3] Un símbolo es un concepto relevante o particular del dominio del negocio actual. Es la contrapartida de un término del LOM.

[4] Palabra muy utilizada en ingeniería de requisitos, derivada del inglés Elicit, que aproximadamente significa extraer algo de un proceso elaborado.

Esta aproximación permite construir el modelo de casos de uso, auxiliado en un método basado en metas, y definir el modelo conceptual usando transformaciones graficas de FM; además, propone de una herramienta MDSD para la implementación de TFM4MDA.

Algunos principios interesantes de TFM4MDA son los siguientes:

- Se debe separar el dominio del problema del dominio de la aplicación, obedeciendo al principio de separación de vistas de MDA.

- Un CIM de MDA debería estar conformado por el modelo de requisitos del sistema, el modelo de proceso del negocio, el modelo de objetos y el modelo del entorno en el que operará el sistema.

- Es frecuente y prácticamente normal que las especificaciones informales posean partes ambiguas, razón por la que es difícil detectar errores y corregirlos.

- La precisión de una especificación formal radica en que, aunque la especificación no corresponda con lo que el usuario necesita, es fácil de identificar sus incorrecciones y mejorarlas.

- En cuanto al análisis del problema, se debe analizar separadamente el contexto del negocio y el contexto de la aplicación. La primera premisa implica que el contexto de la aplicación restringe el dominio de la aplicación y no al contrario (lo satisface completamente). La segunda premisa implica que la funcionalidad es la que determina la estructura del sistema planeado.

- Lo anterior implica que, si se tiene conocimiento de un sistema complejo que funciona en el mundo real, se puede conformar un modelo de funcionamiento topológico del sistema.

8.2.2 Mejoramiento de la arquitectura conducida por modelos con modelos de requisitos

Debnath et al. (2008) proponen definir modelos de requisitos orientados al lenguaje natural con el propósito de inferir de ellos un CIM representado mediante un diagrama inicial de clases. Con tal propósito definen modelos de requisitos orientados al lenguaje natural, que, en última instancia, son especificaciones de requisitos utilizando el LEL y los escenarios propuestos por Leite (Kaplan et al., 2000). Los requisitos así especificados sirven de entrada a un proceso de transformación regido por

un conjunto de reglas gramaticales que sirven para generar clases y relaciones en un ambiente UML (el modelo inicial de clases). Cada regla indica cómo un término del LEL corresponde a un elemento del diagrama de clases.

La transformación se puede hacer manual o de forma automática. La transformación manual implica la especificación de requisitos y la aplicación de las reglas de transformación por parte del desarrollador. La transformación automática implica la aplicación de un proceso automático que involucra la representación de las reglas usando XML y XMI, para finalmente generar un modelo de clases UML.

8.2.3 Desarrollo de una herramienta para la generación del diagrama de clases con el modelo two_hemisphere

En este trabajo Nikiforova y Pavlova (2008) proponen una aproximación de transformaciones CIM-PIM con base en un modelo denominado *two-hemisphere 2HMD* (una analogía a los dos hemisferios del cerebro). Aquí el resultado de la transformación es un modelo de clases UML. Al igual que el cerebro humano, el modelo *Two-Hemisphere* consta de dos hemisferios: uno de ellos es responsable de la lógica del modelo de negocio y el otro de los conceptos del negocio.

9. Consideraciones Finales

El presente libro es el resultado de un esfuerzo de investigación en el que se propone el Léxico Orientado al Modelado (LOM), que es un modelo gramatical de términos del sistema. Esta es una propuesta original cuyo propósito es sentar las bases de un Modelo Independiente de la Computación (CIM).

En definitiva, el LOM se puede definir mediante la 4-tupla <T, D, P, M>, donde ***T*** es una colección de términos relevantes del sistema, ***D*** es un conjunto de descripciones de términos (una para cada uno), ***P*** es el conjunto de patrones que permiten decidir si un término se convierte en un elemento de modelado y ***M*** es el conjunto de elementos de modelado conceptual que se quiere generar. Se han desarrollado alrededor de cinco tipos de términos y 10 tipos de ítems de especificación como una estrategia para analizar los requisitos del sistema y proyectar su transición hacia el modelo conceptual (PIM).

El LOM propuesto en esta investigación genera bases fuertes para construir la definición formal de un CIM para OO_Method; además, se mostró que las

correspondencias encontradas permitirán obtener un PIM para OO_Method, eso es, para generar su modelo conceptual (PIM).

El detalle y la sistematicidad con que se ha definido el LOM, entendido con sus términos e ítems, facilitan la identificación de los términos y los nexos entre términos. Esto tiene dos ventajas visibles:

- Apoya el trabajo (manual) del ingeniero de software, principalmente cuando tiene que enfrentarse a dominios de problemas desconocidos o poco conocidos.

- Prepara el terreno para construir herramientas MDSD que capturen el modelo como un CIM, que fácilmente puede traducir las especificaciones obtenidas a un PIM. En este caso, el modelo conceptual de OO_Method que sirvió de base para establecer los patrones de transición.

Uno de los autores ha probado con éxito este LOM en prácticas de clase con casos reales, durante varios cursos de ingeniería de software. Esto no solo ha probado su efectividad, sino que ha servido para refinarlo. Este modelo tiene las ventajas de acercar al desarrollador al lenguaje del usuario cuando se identifican los términos, ayudar a descubrir los requisitos del sistema en detalle cuando se definen los términos mediante los ítems de cada término y orientar la identificación de términos de modelado mediante patrones.

Aunque el LOM se inspira en el LEL, su alcance es mucho más ambicioso. El objetivo principal del LEL es establecer un lenguaje común entre el usuario y el desarrollador, mientras que el propósito del LOM es llegar a generar un CIM que defina las transformaciones y el procedimiento para generar un PIM.

La investigación aquí presentada sienta las bases para realizar otras investigaciones en el futuro. Teniendo en cuenta que los patrones de modelado, junto con las heurísticas identificadas, facilitan la conformación de elementos de modelado que conduzcan a la definición de reglas concretas de transformación, otros trabajos que se alientan son:

- Conformar un modelo CIM formal con base en las especificaciones y los elementos del LOM.

- Construir una herramienta que implemente el CIM y sus transformaciones para obtener el PIM.

- Con base en un CIM formal y la implementación de las transformaciones requeridas, generar un CIM inicial que pueda refinar el ingeniero de software para construir el CIM completo del sistema.

- También se puede avanzar en identificar posibles elementos de modelado faltantes para lograr un LOM mucho más completo.

- Refinar las especificaciones términos y nexos entre términos del LOM.

- Explorar las posibilidades de las técnicas de procesamiento del lenguaje natural para facilitar la identificación y la captura de los términos y nexos entre términos, a partir de textos que describen el sistema y su funcionalidad.

- Estudiar un conjunto de tecnologías actuales que se aplican a la construcción de herramientas MDA, sus modelos y sus transformaciones, tales como MOF, XML y XMI.

Estas ideas de próximas actividades de investigación serían aportes importantes para construir herramientas MDA-MDSD que faciliten el trabajo del ingeniero de software. Estas herramientas liberarían al ingeniero del duro trabajo de escribir código que, en su mayoría, ya existe y está probado, y encaminaría al ingeniero a conocer el dominio del problema, a diseñar y, de inmediato, a realizar una programación de alto nivel (gráfica y textual) que conduzca a la solución final: el código completo del sistema.

Referencias

Brown, A. W., Conallen, J., & Tropeano, D. (2005). Introduction: Models, modeling, and model-driven architecture (MDA). En S. Beydeda, M. Book, & V. Gruhn, *Model-Driven Software Development* (págs. 1-16). Springer.

Bryant, B. R. (2000). Object-oriented natural language requirements specification. *Proceedings 23rd Australasian Computer Science Conference. ACSC 2000 (Cat. No. PR00518)*, 21 - 30.

Caplat, G., & Sourrouille, J. L. (2002). Model mapping in MDA. En *Workshop in Software Model Engineering (WISME2002)* (Vol. 196).

Chaparro, L. O. (2009). *Técnica de obtención de requisitos con base en estructura de lenguajes léxicos y patrones y desarrollo de un prototipo CASE Asociado.* Universidad de Boyacá.

Chaparro, L. O., & Gómez, J. F. (2012). Una visión del desarrollo de software utilizando modelos. *Gerencia Tecnológica Informática, 11*(29), 69-82.

Debnath, N., Leonardi, M. C., Mauco, M. V., Montejano, G., & Riesco, D. (2008). Improving model driven architecture with requirements models. *Fifth International Conference on Information Technology: New Generations (itng 2008)* (pp. 21-26). IEEE.

Favre, J.-M. (2004). Towards a basic theory to model. Model driven engineering. *3rd Workshop in Software Model Engineering, WiSME.*, 262-271.

Fliedl, G., Kop, C., Mayerthaler, W., Mayr, H. C., & Winkler, C. (2000). Linguistic aspects of dynamics in requirements specifications. *Proceedings 11th International Workshop on Database and Expert Systems Applications*, 83 - 90.

Insfran, E., Pelechano, V., Gomez, J., & Pastor, O. (2002). Un estudio comparativo de la expresividad de relaciones entre clases en OO-Method y UML.

Juristo, N., Morant, J. L., & Moreno, A. M. (1999). A formal approach for generating oo specifications from natural language. *Journal of Systems and Software, 48*(2), 139-153.

Kaplan, G. N., Hadad, G. D., Doorn, J. H., & Sampaio do Prado Leite, J. C. (2000). Inspección del léxico extendido del lenguaje. *WER'oo III WORKSHOP DE ENGENHARIA DE REQUISITOS*, (pp. 70 - 91).

Metzger, A. (2005). A systematic look at model transformations. En *Model-driven Software Development* (pp. 19-33). Springer.

Molina, J. C., & Pastor, O. (2004). MDA, OO-Method y la tecnología OlivaNova Model Excecution. *I Taller sobre desarrollos dirigidos por modelos, MDA y aplicaciones*.

Molina, P. J. (2003). *Especificación del interfaz de usuario: de los requisitos a la generación automática.* Thesis, Universidad Politécnica de Valencia.

Nikiforova, O., & Pavlova, N. (2008). Development of the Tool for Generation of UML Class Diagram from Two-hemisphere model. *The Third International Conference on Software Engineering Advances*, 105 - 112.

Object Management Group - OMG. (2003). *MDA Guide Version 1.0.1.* Joaquin Miller and Jishnu Mukerji. https://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf

Object Management Group - OMG. (2014). *Model Driven Architecture - MDA.* https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

Osis, J. (2004). Software Development with Topological Model in the Framework of MDA. *CAiSE Workshops*, 211 - 230.

Osis, J., Asnina, E., & Grave, A. (2007). Computation independent modeling within the MDA. *IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'07)*, (pp. 22-34).

Pons, C., Giandini, R. S., & Pérez, G. (2010). *Desarrollo de software dirigido por modelos. conceptos teóricos y su aplicación.* (E. d. Plata, Ed.). Mc Graw Hill Education.

Seidewitz, E. (2003). What models mean. *IEEE software, 20*(5), 26-32.

Sommerville, I. (2005). *Ingeniería del Software. 7ma ed.* Pearson-Prentice Hall.

Zapata, C. M., Jaramillo, A. F., & Arango, F. (2006). Una propuesta para mejorar la completitud de requisitos utilizando un enfoque lingüístico. *Revista Científica Ingeniería y Desarrollo, 19*(19), 1 - 16.

Este libro se terminó de imprimir
en el mes de Julio de 2023 en
Panamericana Formas e Impresos S.A.