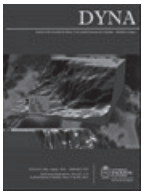
UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS



DYNA
http://dyna.medellin.unal.edu.co/

A framework for consistences in association relations between classes in UML

Javier Dario Fernández-Ledesma [a]

[a] *Facultad de Ingenierías, Universidad Cooperativa de Colombia. javier.fernandez@ucc.edu.co*

Received: September 2th, 2013. Received in revised form: May 9th, 2014. Accepted: May 28th, 2014.

Abstract

The following article shows the process of building and validating a framework for the management of consistencies in class diagrams in UML, operating specifically on class diagrams, through the application of transformation rules, using both graph grammar and OCL (Object Constraint Language). The proposed framework, after examining the techniques of consistency management, operates on a class diagram, since this constitutes a structural diagram. This is even more important when modeling, and it facilitates the application of rules based on the treated techniques, thus contributing to provide the community of analysts and modelers with a support tool for the refinement and quality improvement of the diagrams. It also operates on a typical case of application to show the tool's advantages, thus making it easier to comprehend and understand.

Keywords: Consistence, Framework, UML.

Un framework para consistencias en relaciones de asociación entre clases en UML

Resumen

El siguiente articulo muestra el proceso de construcción y validación de un framework para el manejo de consistencias en diagramas de clases de UML, específicamente opera sobre los diagramas de clase, mediante la aplicación de reglas de transformación, usando tanto la gramática de grafos como el OCL (Object Constraint Language). El framework propuesto, luego de un recorrido sobre las técnicas de manejo de consistencias, opera sobre el diagrama de clase toda vez que este constituye el diagrama estructural, si se quiere, más importante, a la hora de modelar, y facilita la aplicación de reglas desde las técnicas tratadas, contribuyendo con ello a dotar a la comunidad de analistas y modeladores de una herramienta soporte para el refinamiento y mejoramiento de la calidad de los diagramas, opera así mismo, sobre un caso típico de aplicación para mostrar las bondades de la herramienta, lo cual facilita su comprensión y entendimiento.

Palabras clave: Consistencia, Framework, UML.

1. Introducción

Este articulo constituye un aporte a las metodologías de desarrollo de software orientado a modelos (MDA) por tanto se centra en el manejo de los modelos independiente de las plataformas de modelado y permite ahondar en las problemáticas aún abiertas que existen para el desarrollo de software desde las etapas tempranas con técnicas, métodos y metodologías que provean a la industria de software de productos de alta calidad.

Por otro lado, es el resultado del proceso de construcción de un framework base para la implementación de un método usado en la traducción y la validación de diagramas de UML, en específico, el diagrama de clases.

En la primera parte se define el marco conceptual y el estado del arte sobre los componentes de la propuesta, seguidamente se muestra la metodología general desarrollada en la construcción del framework y se termina con el análisis, diseño, programación y resultados de implementación del framework propuesto en un caso de estudio.

2. Antecedentes

2.1. Antecedentes conceptuales

El UML (Lenguaje Unificado de Modelado) se ha definido como un lenguaje gráfico para visualizar, especificar, construir y documentar los elementos de un

sistema con gran cantidad de software [1], la evolución natural del UML ha llevado al desarrollo de nuevas metodologías como es el caso de MDA (Arquitectura Dirigida por Modelos), la cual propone basar el desarrollo de software en modelos especificados utilizando Metalenguajes como UML, a partir de los cuales se realicen trasformaciones que generen código u otros modelos, con características de una tecnología particular (o con menor nivel de abstracción). Por lo tanto MDA define un entorno de trabajo para procesar y relacionar modelos.

Enmarcado entonces en este esquema de trabajo y con UML como metalenguaje para el metamodelado, se encuentran los diagramas de clase como un tipo particular de diagramas donde se muestra la estructura del sistema en términos de clases, interfaces, colaboraciones y relaciones [1], el diagrama de clases es un grafo, donde los nodos son elementos del tipo clasificador y los arcos son las distintas relaciones estáticas que existen entre elementos. En este diagrama se especifican los elementos del sistema, que serán referenciados por el resto de los diagramas, lo cual lo convierte en el diagrama central de un modelo.

Las clases son representadas en la forma de una caja rectangular, como se muestra en la siguiente Fig.1.

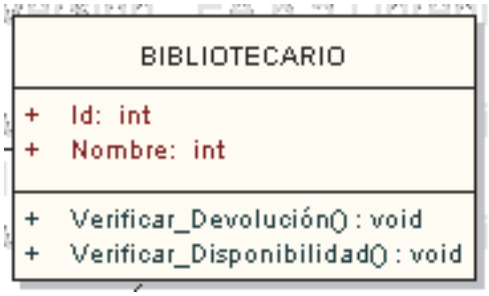


Figura 1. Representación de una Clase

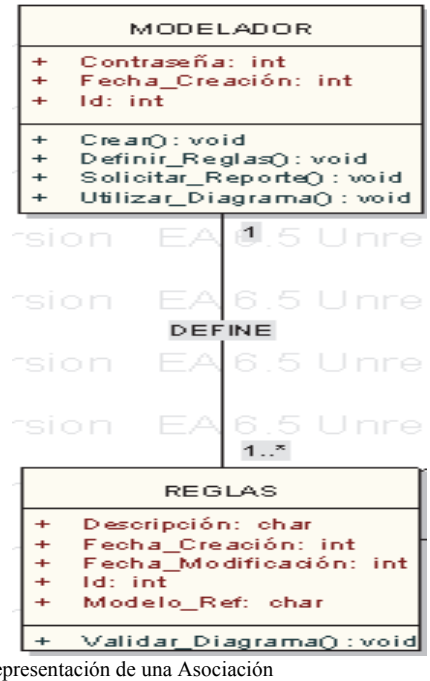


Figura 2. Representación de una Asociación
Fuente: El autor

Una clase representa un concepto del sistema que está siendo modelado y define la estructura (atributos) y comportamiento (métodos) de los objetos que la instancian.

Las relaciones que existen entre clases son: asociación, agregación, composición y generalización. La relación más básica entre clases es asociación, representada por una línea.

El caso más común es una asociación binaria, pero se permite la modulación de relaciones entre *N* clases. Cada extremo de una asociación debería llevar un nombre y una multiplicidad.

Este tipo de relaciones entre clases permite visualizar el manejo de reglas de consistencia a partir de técnicas como: OCL (Lenguaje de Especificación de Objetos), el cual es un lenguaje formal para expresar restricciones libres de efectos colaterales. El OCL permite especificar restricciones y otras expresiones incluidas en sus modelos. El componente central construido por este lenguaje es la expresión, que se manipula a través de operaciones las cuales garantizan la transparencia referencial. Así, toda expresión válida para el lenguaje OCL debe tener asociado un tipo de datos. Las expresiones se pueden mapear fácilmente al concepto de objetos, ya que cada tipo básico no predefinido se puede vincular a una clase del diagrama, y algunas de las operaciones de un tipo básico no predefinido se modelan como atributos y métodos/responsabilidades de la clase original. En cada definición de OCL se describe cada restricción semántica del diagrama de clases, junto a otras propiedades también descritas con expresiones. Cada definición en OCL se aplica a lo que se llama un contexto, que representa el destinatario del diagrama de clases sobre quien se aplica esa definición.

Otro enfoque sobre la transformación para la consistencia de modelos lo provee la gramática de grafos, la cual se compone de una serie de reglas, cada regla define su parte izquierda un sub-grafo patrón y en su parte derecha un sub-grafo sustitución, donde la parte izquierda y derecha (pre- y post- condiciones) se encuentran en los grafos que definen el modelo. Así, cuando se aplica una gramática a un grafo, las ocurrencias de los patrones de las reglas de la parte izquierda son sustituidas por las reglas de la parte derecha, de este modo el grafo se va transformando. Las reglas pueden contener también una condición, que debe ser satisfecha para que se pueda aplicar, así como acciones que se ejecutan si la regla es aplicada. Algunos enfoques ofrecen también especificaciones para el control de flujo en la ejecución de las reglas. Por otra parte, el uso de un modelo (en forma de gramáticas de grafo) para representar manipulaciones de modelos tiene ciertas ventajas sobre una representación implícita (es decir, representando la computación que realiza la transformación en un programa textual). Por ejemplo, las gramáticas de grafos son una representación gráfica, abstracta, formal, declarativa y de alto nivel de las computaciones. Además, los fundamentos teóricos de los sistemas de reescritura de grafos pueden ayudar a probar propiedades de la transformación, como la corrección y la convergencia (terminación).

2.2. Antecedentes investigativos

Algunos de los problemas que se suelen presentar en el manejo de la consistencia en diagramas de clase de UML

están referenciados en [2], como problemas inherentes a la: ambigüedad, inconsistencia, suficiencia y dirección cognoscitiva errada. Siendo los trabajos más representativos sobre estos problemas los desarrollados por [3] y [4].

[4] Específica en su trabajo, los aspectos de la asociación en UML que deben ser clarificados, en términos de la concepción estática de la asociación frente a la concepción dinámica, es decir, si la asociación expresa el estado de la información del sistema (estructura de datos), si expresa el dinamismo de las interacciones dentro del sistema (intercambio de mensajes), o bien si, tal vez, representa ambos aspectos a la vez. La "cosificación" de la asociación, que permite que cada instancia de una asociación (denominada "enlace") tenga identidad, propiedades y comportamiento propio. La multiplicidad de la asociación, especialmente en el caso de asociaciones ternarias y de grado superior. La navegabilidad de la asociación, sus implicaciones sobre la comunicación mediante mensajes. La visibilidad de las asociaciones en relación con el concepto de "interfaz" y finalmente, la implementación de las asociaciones en lenguajes de programación orientados a objetos.

[4] y [3] Específicamente abordan los problemas de la asociación desde el punto de vista de las propiedades: multiplicidad, navegabilidad y visibilidad.

Y es en este contexto donde aparecen los trabajos más sobresalientes hasta el momento relacionados con la implementación de asociaciones en diagramas de clases de UML y que proponen técnicas, métodos y metodologías para la solución de problemas, estos trabajos son:

En 2000, en la Universidad de Nantes en Francia [5], se pretende introducir semántica formal para UML basada en tipos de datos abstractos, utilizando una herramienta llamada probador *Larch*, que es un asistente de prueba, el cual permite definir especificaciones algebraicas y es capaz de soportar pruebas sobre diagramas UML. Este trabajo tiene en cuenta las características de multiplicidad y navegabilidad de las asociaciones y aunque los autores reconocen que está aún lejos de una estrategia para chequear inconsistencias, ya que los resultados dependen de la forma de los axiomas y las propiedades conocidas por el sistema, que se consideran aún en un estado muy incipiente, su gran fortaleza reside en la utilización en primera instancia de un lenguaje de especificación formal.

Luego, en 2003 en la Universidad Carlos III de Madrid [4], se desarrolla una tesis doctoral que establece una serie de principios de diseño y plantillas de código para guiar a los desarrolladores en la implementación de las asociaciones en un lenguaje de programación orientado a objetos, a partir de un análisis de los aspectos: multiplicidad, navegabilidad y visibilidad de las asociaciones en UML. Estos principios son aplicados en una herramienta llamada JUMLA (*Java code generator for UML Associations*), que genera automáticamente código para implementar las asociaciones del UML. La herramienta lee un modelo almacenado en formato XMI y crea archivos de salida java para las clases implicadas, insertando en ellos el código para las asociaciones. El aporte más importante de este trabajo es que analiza deficiencias semánticas relacionadas con las asociaciones, las cuales en su mayoría son tratadas en [3] y propone las soluciones respectivas; además produce la transformación directamente del PIM (modelo independiente de la plataforma) a código fuente, sin pasar por el PSM (modelo específico de la plataforma).

Por su parte, [6] plantea el uso del DL (*Description Logic*) como lenguaje formal para codificar los diagramas de clase, estas deben ser representadas por conceptos y los atributos y roles de asociación deben ser representados por elementos descriptivos del lenguaje. Así, cada parte de los diagramas es codificado por la adecuada inclusión de aserciones.

Luego, [7] establecen que las dificultades más importantes que aparecen en la implementación de las relaciones de asociación, son: Una relación de asociación tiene semántica propia, adicional a la semántica de las clases participantes en la relación y los objetos de las clases participantes tienen un comportamiento y una estructura adicional a la especificada por su clase de dominio (por participar en una relación de asociación). Para lo cual propone un *framework* basado en patrones de diseño para la implementación de relaciones de asociación, agregación y composición. Para ello se presenta una interpretación semántica de estos conceptos que permite eliminar ambigüedades introducidas por UML y que se logra a través de un conjunto de propiedades que permiten caracterizar estas relaciones. Los patrones de diseño utilizados en el *framework* son: *Decorator*, para representar el comportamiento y propiedades de los objetos de las clases participantes en las relaciones, adicionales a las propias, inherentes de la clase, *Mediator* para representar las propiedades especificadas sobre la relación, y *Template Method* para definir la estrategia común de la creación de objetos. Con la utilización de estos patrones de diseño la implementación de cada uno de los elementos de una relación de asociación se centraliza, lo que facilita el mantenimiento del código y beneficia la reusabilidad de los distintos elementos participantes en la relación, así el *framework* propuesto proporciona una implementación natural de las propiedades del marco conceptual definido, cumpliendo los requisitos intrínsecos de las relaciones y generando una solución software sencilla, con elementos reusables y de fácil mantenimiento.

Adicional a esto, [8] presentan un enfoque metodológico para la definición de transformaciones entre modelos. Este enfoque propone el uso de gramáticas de grafos para realizar una descripción precisa y operativa de las transformaciones entre modelos. En [8] proponen la utilización de gramática de grafos concretamente para la conversión de relaciones de asociación expresadas en modelos de análisis (PIM) hacia un modelo de diseño elaborado en términos de un lenguaje orientado a objetos (PSM). Para especificar sin ambigüedad los elementos que forman parte de estos modelos se utilizan metamodelos. La gramática propuesta está formada por 11 reglas que reflejan la estrategia de implementación, y utiliza los patrones de diseño y los fundamentos del framework propuesto en [7]. Esta técnica tiene varias ventajas sobre otras, como que cuenta con una sólida base matemática, permite una representación gráfica, facilitando la comprensión de las transformaciones, y ya existen algoritmos y herramientas

para su aplicación. Los problemas encontrados en estas transformaciones consisten entre otros en que no soportan herencia en la definición de los metamodelos y en que en la aplicación de la gramática de grafos, el modelo debe validar siempre uno de los metamodelos propuestos.

Hacia el año 2005, se presentan una serie de fundamentos para la transformación de herramientas basadas en UML como es el caso de los trabajos de [9], quien utiliza el lenguaje algebraico *NEREUS* para traducir el metamodelo UML y las restricciones expresadas en OCL. Así, las expresiones *NEREUS* a su vez se traducen a *EIFFEL*, un lenguaje orientado a objetos. Estas transformaciones son soportadas por una librería de componentes reusables y por un sistema de reglas de transformación, que posibilitan las dos traducciones. *NEREUS* consiste de varias construcciones para expresar clases, asociaciones y paquetes. El texto de la especificación *NEREUS* se completa gradualmente; primero se obtienen los axiomas y luego se transforman las asociaciones, instanciando componentes reusables que existen en este lenguaje. Finalmente, las especificaciones OCL son transformadas usando un conjunto de reglas de transformación.

Por su parte los trabajos de [10], tratan el problema de consistencia entre diagramas definidos en modelos UML mediante el uso de un formalismo para representar los diagramas y modelos que permita la investigación y manipulación de la semántica de los modelos elaborados por el usuario. [10] propone los sistemas de representación del conocimiento (SRC) los cuales permiten a través de definiciones de alto nivel, la representación sistematizada de dominios, mientras que situaciones específicas se traducen como los individuos que representan a los conceptos definidos. De esta forma, las consecuencias implícitas asociadas al conocimiento explícitamente representado pueden ser descubiertas a través del uso de bases de conocimiento. El resultado de este trabajo es una aplicación denominada *Model Consistency Checker* (MCC) implementada en *Java* y el formalismo que utiliza es *Description Logics* (DL), dado que experimentos anteriores probaron que este formalismo podría servir de base para la detección y resolución de un conjunto amplio de problemas de consistencia. Finalmente, propone un entorno llamado *Racer*, el cual es un motor de inferencia para DL que es completo y eficiente y brinda los servicios de razonamientos usados por MCC. La interfaz gráfica del sistema está integrada como *plug-in* a *Poseidón*, una herramienta CASE para UML.

Posteriormente, se desarrolló en la Universidad Vrije Universiteit Brusel [11] una tesis doctoral, cuyo resultado es un prototipo en *Java* que permite al desarrollador de software hacer la transición del nivel de diseño al nivel de implementación con seguridad respecto a las restricciones de las clases especificadas en el modelo. En él se hace una clasificación de las restricciones en seis categorías, siendo una de ellas las correspondientes a asociación entre dos clases. El prototipo desarrollado está basado en la herramienta *CASE Argo/UML*. La lectura del modelo se hace desde archivos XMI, generados a partir de los diagramas UML. Las restricciones están expresadas en OCL y son traducidas dentro de declaraciones ejecutables *Java* a través de un módulo normalizador y cada restricción es representada por una clase. El analizador de puntos de inserción, evalúa expresiones OCL y sugiere puntos en el código de aplicación donde las restricciones podrían ser violadas, permitiéndose la interacción del usuario para borrar y agregar puntos. Esta propuesta no analiza las deficiencias semánticas del estándar UML.

Finalmente, los trabajos de [12] y [13], proponen un modelo para las asociaciones, basado en un marco conceptual que identifica las propiedades más relevantes de la asociación como: Dinamicidad, Mutabilidad, Multiplicidad, Propagación de Borrado, Navegabilidad, Proyección de Identificación, Reflexividad, Simetría y Transitividad.

Bajo el desarrollo y conocimiento de estos trabajos previos se propuso la construcción de un *Framework* que permitiera el manejo de la consistencia en diagramas de clase de UML como un aporte a la solución de los problemas encontrados y enunciados en los trabajos previos y como un aporte a la industria del software con herramientas de apoyo para la construcción de modelos que cumplan con altos estándares de calidad para la correcta y eficiente construcción de sistemas de software.

3. Metodología

En la construcción del framework para el manejo de consistencias en diagramas de UML a partir de reglas de transformación se ha construido un modelo general haciendo uso del UML como lenguaje de metamodelado y a continuación se muestran tanto el diagrama de Casos de Uso en la Fig. 3 como el diagrama de clases, en la Fig. 4, del framework.

El framework permite validar reglas de consistencias definidas por el usuario en el módulo de definición de reglas, de donde el analizador de consistencias que constituye otro módulo del aplicativo toma información para validar si los diagramas creados o importados a la interfaz del aplicativo por medio del módulo de diagramas cumple o no con la reglas definidas y generar entonces en el módulo de reportes un informe de inconsistencias presentes en el modelo.

El framework consiste de tres módulos funcionales fundamentalmente: el módulo de diagramas, allí se crean, modifican e importan los diagramas que desean ser validados y se guardan como documentos XML; el módulo de reglas, allí se crean, modifican y se almacenan las reglas de definición de consistencia en los modelos y el módulo de validación, en el cual se cruza la información contenida en los diagramas a analizar y se confronta con la definición de las reglas que se utilizarán en la validación, las cuales han sido especificadas y formadas con OCL y con DL, generando un informe sobre inconsistencias detectadas en el diagrama.

La interfaz del framework se ha construido en el lenguaje Java, para soportar la importación de modelos que requieren el uso de Api's específicas, como se puede apreciar en la Fig. 5, la cual muestra el módulo de diagramas para la creación de relaciones de asociación entre clases.

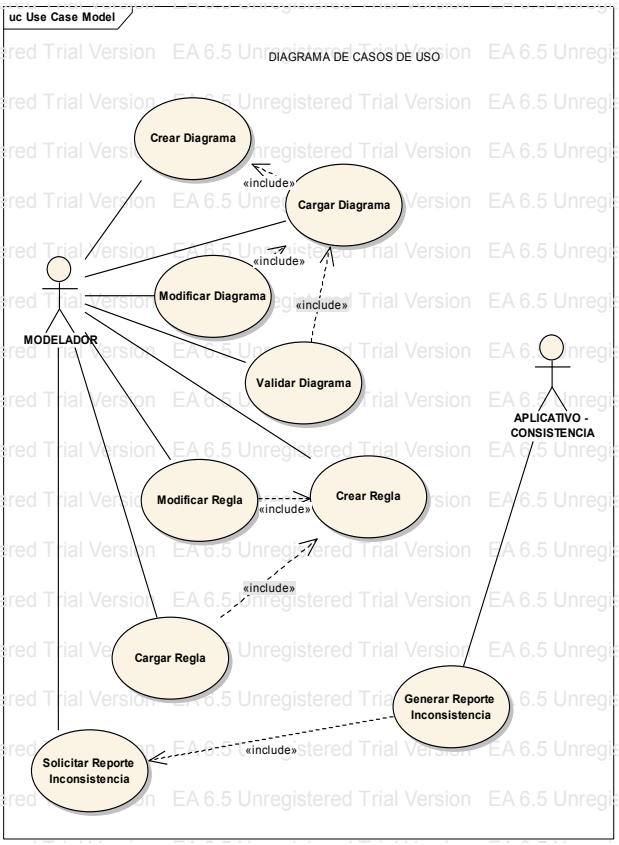


Figura 3. Casos de Uso del Framework
Fuente: El autor

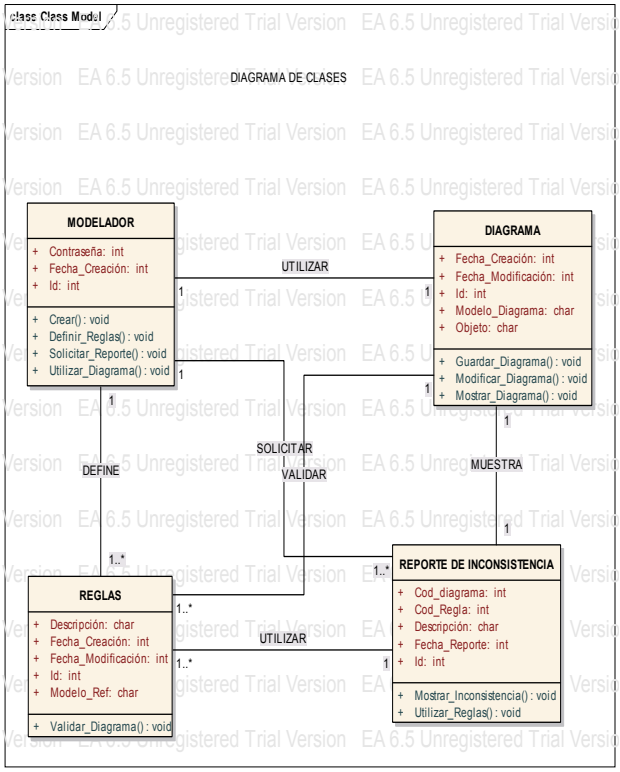


Figura 4. Diagrama de Clases del Framework
Fuente: El autor

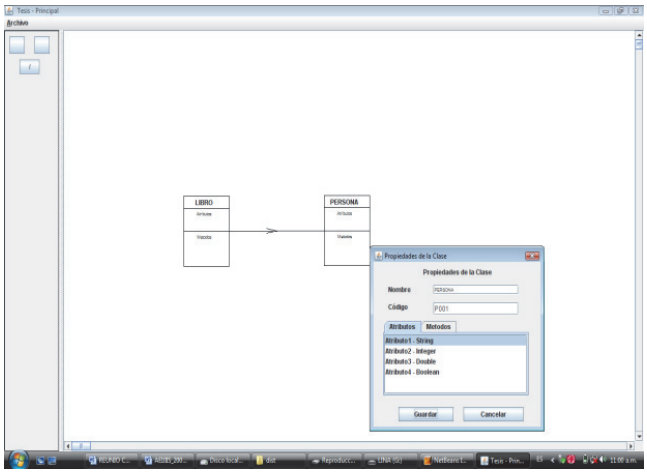


Figura 5. Módulo de Diagramas del Framework
Fuente: El autor

4. Resultados

Para la validación del framework se ha definido un caso típico consistente en el proceso de préstamo bibliotecario, en el cual el usuario para pedir el préstamo de un libro debe de identificarse ante el bibliotecario y solicitar el libro deseado, y poder efectuar el préstamo, el libro debe de estar disponible y el usuario debe de estar a paz y salvo con la biblioteca, en caso de tener alguna deuda pendiente el usuario debe de cancelarla para poder retirar el libro. Después de haber cumplido estas condiciones el bibliotecario asigna una fecha de devolución para el libro y lo entrega al usuario.

Así mismo en el momento de la devolución el usuario debe saldar la deuda para poder proceder a la devolución, en el momento en el cual el bibliotecario recibe el libro, éste verifica que la devolución se esté efectuando dentro del

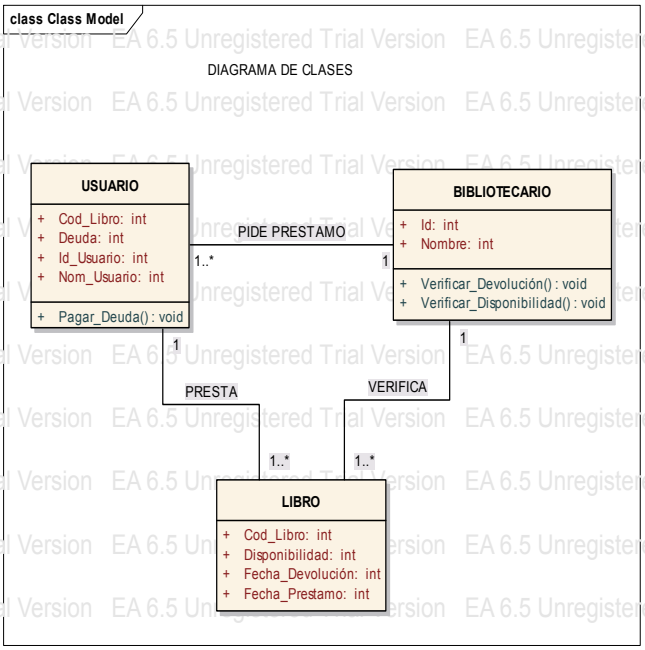


Figura 6. Diagrama de Clases Caso: Préstamo Bibliotecario
Fuente: El autor

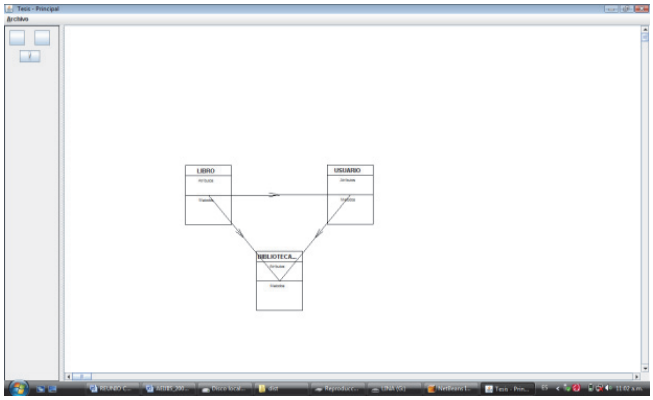


Figura 7. Implementación de caso de estudio en el Framework propuesto
Fuente: El autor

periodo de tiempo determinado, en ese tiempo el libro se acepta y el usuario se retira. Si la fecha de devolución no es la asignada en el momento de realizar el préstamo, se calcula un pago en función de los días de retraso y se le acredita como una deuda del usuario. La deuda debe de cancelarse en su totalidad; de lo contrario, la no cancelación de esta imposibilita al usuario el préstamo y devolución de libros. El siguiente es el diagrama de clases que representa el caso de estudio utilizando una herramienta Case comercial de UML:

Luego, se procedió a construir el Diagrama de Clases del caso de estudio como se muestra en la Fig. 7, en el framework propuesto, donde se ha validado automáticamente con las reglas de consistencia propuestas en el mismo.

Este diagrama se ha chequeado automáticamente con el fin de permitir su presentación final y almacenamiento en el framework.

5. Conclusiones

El framework para el Manejo de Consistencias en Diagramas de Clase de UML, permite, entre otros aspectos:

- Coadyuvar en los procesos de diseño de software mediante la verificación de reglas de consistencia sobre diagramas de clase de UML.
- Permitir el uso de un framework para la verificación de modelos existentes y modelos creados en la herramienta con altos criterios de calidad.
- Dotar de un sistema integral, flexible y seguro para la gestión de la consistencia en modelos de UML para el diseño de software.
- Aumento de la productividad en los índices de gestión y control de los procesos de desarrollo de software mediante la reducción de ocurrencia de errores en las fases de análisis y diseño de software.
- Disminuir los tiempos de modelado de software.

Por lo pronto, el framework ha permitido el mejoramiento de los siguientes indicadores en los contextos de construcción de software donde se ha implementado:

- Mejoramiento en la calidad y la productividad en los procesos de desarrollo de software.
- Disminución hasta de un 20% en la ocurrencia de errores en las fases de análisis y diseño de software.
- Aumento en los niveles de calidad de los modelos de software.
- Integridad en el manejo de los modelos de software.

Así mismo, tanto las herramientas de diseño utilizadas como las de programación han resultado adecuadas a los propósitos planteados en la construcción de este framework, a un bajo costo, flexible y adaptable.

Referencias

[1] Booch, G., The Unified modelling language, User guide. 1a ed., México, Addison-Wesley, 1999.

[2] Simons, C., CMP: A UML context modeling profile for mobile distributed systems, en 40th Annual Hawaii International Conference in System Sciences, Hawaii, EEUU, pp. 289b-289b, 2007.

[3] Stevens, P., Small-scale XMI programming: a revolution in UML tool use?. Automated Software Engineering, 10(1), pp 7-21, 2003.

[4] Génova, G., Entrelazamiento de los aspectos estático y dinámico en las asociaciones UML, Tesis de Doctorado en ingeniería informática, Universidad Carlos III de Madrid, España, 2003, 234 P.

[5] André, P., Romanczuk, A., Royer, J. C. and Vasconcelos, A., Checking the consistency of UML class diagrams using larch prover, en: Rigorous Object-Oriented methods, Nantes, Francia, 2000, pp. 1-16.

[6] Calvanese, D. and De Giacomo, G., Description logics for conceptual data modeling in UML, en 15th European summer school in logic language and information, Vienna, Austria, 2003, pp. 18-29.

[7] Ruiz, M., Manoli, A., Torres, V. and Pelechano, V., Un Framework para la implementación de relaciones de asociación, agregación y composición en UML, en Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software, España, 2004, pp. 245-256.

[8] Muñoz, J., Manoli, A., Ruiz, M. and Pelechano, V., Transformación de relaciones de asociación mediante gramáticas de grafos aplicando MDA, en 1er Taller sobre Desarrollo de Software Dirigido por Modelos, Málaga, España, 2004, pp. 43 - 56.

[9] Favre L., Foundations for MDA-based forward engineering. Journal of Object Technology, 4 (1), pp. 129-153.

[10] Wagemann, J. P. S., Un framework para el manejo de consistencia en diseños UML 2.0, Tesis de Doctorado en Ciencias de la Computación, Universidad de Chile, Chile, 2005, 70 P.

[11] Vanderperren, W., Suvée, D., Verheecke, B., Cibrán, M. A. and Jonckers, V., Adaptive programming, en 4th International conference on Aspect-oriented software development, Chicago, EEUU, 2005, pp. 75-86.

[12] Albiol, M. A., Tratamiento de relaciones de asociación en entornos de producción automática de código, Tesis de Doctorado en Informática, Universidad Politécnica de Valencia, España, 2006, 309 P.

[13] Arango F., Gómez M. and Zapata C., Transformación del modelo de clases UML a Oracle9i® bajo la directiva MDA: Un caso de estudio, DYNA, 73 (149), pp. 165-179, 2006.

J. D. Fernández-Ledesma, recibió su grado de Ing. Industrial en el 2000 en la Universidad de Antioquia, su grado de Esp. en Ingeniería de Software en 2002 en la Universidad Nacional de Colombia Medellín, su grado de MSc. en Ingeniería en 2007 en la Universidad de Antioquia Medellín, Colombia. Desde el año 2005 trabaja como docente investigador de la Universidad Cooperativa de Colombia, Medellín. Actualmente trabaja como profesor asociado en la Facultad de Ingeniería Industrial de la Universidad Pontificia Bolivariana y como docente e investigador de la Facultad de Ingenierías de la Universidad Cooperativa de Colombia. Sus áreas de interés son: simulación discreta, optimización e ingeniería y desarrollo de software.