ISSN: 0798-1015 DOI: 10.48082/espacios-a20v41n45p17

REVISTA ESPACIOS

Vol. 41 (45) 2020 • Art. 17

Recibido/Received: 10/08/2020 • **Aprobado**/Approved: 19/09/2020 • **Publicado**/Published: 26/11/2020

QModel-XMI: un mecanismo de consulta para modelos XMI

QModel-XMI: a query mechanism for XMI models

MONROY, Martín E.[1]
RODRÍGUEZ, Julio C.[2]
PUELLO, Plinio[3]

Resumen
Existen herramientas que facilitan el análisis de modelos de productos software, pero requieren del conocimiento de expertos. Este trabajo propone un mecanismo de consulta en lenguaje natural, para hacer análisis sobre modelos UML representados en XMI. Los resultados de la evaluación de QModel-XMI revelan su utilidad como herramienta para apoyar el análisis de modelos UML. La herramienta desarrollada constituye una nueva forma de realizar consultas sobre modelos UML representados en XMI, que facilita el proceso de análisis del producto software.
Palabras clave: Q-Model-XMI, análisis de modelos, mecanismo de consulta, ingeniería de software.

Abstract
There are tools that facilitate the analysis of software product models, but they require the knowledge of experts. This work proposes a query mechanism in natural language, to make analysis on UML models represented in XMI. The results of the QModel-XMI evaluation reveal its usefulness as a tool to support the analysis of UML models. The developed tool constitutes a new way of consulting on UML models represented in XMI, which facilitates the analysis process of the software product.
key words: Q-Model-XMI, model analysis, query mechanism, software engineering.

1. Introducción

Un mecanismo de consulta es una estructura que organiza sus partes constitutivas para permitir la formulación de preguntas, con la respectiva representación de las respuestas. En el campo de la ingeniería de software esta estrategia se usa para apoyar procesos de mantenimiento (IEEE, 2016; Canfora, Di Penta, y Cerulo, 2011), facilitando el análisis y la comprensión del producto (Ujhelyi, Bergmann, Hegedüs, Horváth, Izsó, Szatmári, y Varró, 2013; Urma y Mycroft, 2015; Alves, Hage y Rademaker, 2011), así como también la validación y transformación de modelos (Alves et al., 2011; Beyer, Noack y Lewerentz, 2005). Todo mecanismo de consulta requiere de un lenguaje de consulta y una herramienta que interprete las consultas y presente los resultados de su ejecución (Monroy, Arciniegas y Rodríguez, 2017). La revisión de la literatura reveló que existen 23 lenguajes

[1] Profesor de la Universidad de Cartagena. Programa de Ingeniería de Sistemas. Facultad de Ingeniería. Ingeniero de Sistemas, MSc. en Ciencias Computacionales, PhD. en Ingeniería Telemática. mmonroyr@unicartagena.edu.co
[2] Profesor de la Universidad de Cartagena. Programa de Ingeniería de Sistemas. Facultad de Ingeniería. Ingeniero de Sistemas, PhD. en Ingeniería de Sistemas Telemáticos. jrodriguezr@unicartagena.edu.co
[3] Profesor de la Universidad de Cartagena. Programa de Ingeniería de Sistemas. Facultad de Ingeniería. Ingeniero de Sistemas, Dr(c) en Ingeniería. ppuellom@unicartagena.edu.co

especializados y 21 mecanismos de consulta que pueden ser utilizados para analizar productos software (Monroy et al., 2017).

El análisis de un producto software consiste en examinarlo detalladamente, separando o considerando por separado sus partes, para conocer sus características o cualidades, o su estado, y extraer conclusiones. Se puede hacer con base en su código fuente o en los modelos que representan su diseño y arquitectura. La mayoría de los lenguajes especializados identificados permiten hacer consultas sobre el código fuente (Bergmann, Hegedüs, Horváth, Ráth, Ujhelyi y Varró, 2012; Urma y Mycroft, 2015; De Roover, Noguera, Kellens y Jonckers, 2011; Verbaere, Godfrey y Girba, 2008; Störrle, 2013), sólo una parte reducida lo hace sobre los modelos (Liepiņš, 2012; Bruneliere, Cabot, Jouault y Madiot, 2010; Acretoaie y Störrle, 2012). Esta relación se mantiene en las herramientas que implementan mecanismos de consulta, porque la mayoría han sido diseñadas para comprender y analizar código fuente, mientras sólo algunas permiten la interpretación (Acretoaie y Störrle, 2012; Beeri, Eyal, Kamenkovich y Milo, 2008), el análisis (Liepiņš, 2012) y las transformaciones (Bruneliere et al., 2010; Acretoaie y Störrle, 2012; Beeri et al., 2008) de modelos, o la reutilización (Sakr y Awad, 2010) de activos software.

Por ejemplo: MACH integra prototipos para permitir el análisis de sistemas (Störrle, 2013). IQuery permite consultas y transformaciones independientemente del repositorio utilizado para guardar los modelos (Liepiņš, 2012). El Marco de referencia de ingeniería inversa dirigida por modelos MoDisco permite la interpretación de los modelos por medio de consultas (Bruneliere et al., 2010). MQ-2 integra Prolog a la consola de MagicDraw permitiendo hacer consultas sobre el modelo usando notación formal para facilitar su interpretación (Acretoaie y Störrle, 2012). BP-QL Prototype System permite hacer consultas sobre los modelos de proceso de negocio para facilitar su interpretación (Beeri et al., 2008). EMF-IncQuery brinda un framework de alto rendimiento para consultas y transformaciones usando el formalismo declarativo (Bergmann et al., 2012); y A framework for querying graph-based BPM permite hacer consultas sobre los modelos de proceso de negocio para facilitar la reutilización (Sakr y Awad, 2010).

Todas las herramientas identificadas exigen el aprendizaje de lenguajes de consulta especializados (Monroy et al., 2017). Esto limita su capacidad de uso exclusivamente a personas con profundos conocimientos del lenguaje especializado que utiliza el mecanismo de consulta. La única herramienta que almacena los modelos en repositorios XML es MoDisco (Bruneliere et al., 2010), todas las demás requieren de procesos complementarios y el uso de herramientas especializadas de conversión de los modelos UML, representados en XMI a formatos aceptados por los mecanismos en mención, sumando complejidad al proceso de análisis de los modelos. También se resalta que MACH, IQuery, MQ-2 y EMF-IncQuery procesan modelos software representados en UML, MoDisco lo hace sobre Modelos KDM, mientras que *A framework for querying graph-based BPM* y *BP-QL Prototype System* lo hace sobre modelos de procesos de negocio. Esto permite inferir que el análisis de modelos UML usando mecanismos de consulta es un enfoque relativamente nuevo, con un amplio camino por explorar, en el que, por ahora EMF-IncQuery muestra un proceso de evolución que lo ubica como referente (Ujhelyi et al., 2013; Bergmann et al., 2012).

Ante esta situación surge la necesidad de un mecanismo de consulta que permita hacer análisis sobre los modelos de productos software representados en UML. En consecuencia, la principal contribución de esta investigación es el diseño e implementación de un mecanismo de consulta basado en lenguaje natural, para que las personas involucradas en un proceso de ingeniería de software puedan hacer análisis sobre modelos UML representados en XMI, sin tener que aprender complejos lenguajes de consulta especializados. La implementación del mecanismo de consulta está representada por el prototipo QModel-XMI (Query Model in XMI), que sirvió como instrumento para evaluar la validez del diseño propuesto. La utilidad de esta propuesta se evidencia sobre todo en contextos como la educación, porque permite centrar la atención del educando en el

aprendizaje del tema específico de ingeniería de software, haciendo análisis sobre modelos UML sin necesidad de conocer lenguajes de consulta especializados.

A continuación se presentan los métodos y materiales utilizados para el desarrollo de la investigación. Luego se explica el diseño y la implementación del mecanismo de consulta. Posteriormente se exponen los resultados del estudio de caso que se aplicó para evaluar el prototipo QModel-XMI. Más adelante se analizan los resultados y se proponen trabajos futuros. Al final se presentan las conclusiones.

2. Metodología

La investigación se realizó en dos fases. En la primera se hizo una revisión de la literatura aplicando el enfoque metodológico establecido por Kitchenham et al. (2011). Los resultados de esta fase fueron publicados en Monroy et al. (2017). En la segunda fase se hizo el diseño y la implementación del mecanismo de consulta, utilizando el proceso unificado de desarrollo de software (Jacobson, Booch y Rumbaugh, 1999). Como resultado de esta fase se definió la arquitectura del mecanismo de consulta, se desarrolló el prototipo QModel-XMI y se probó su utilidad aplicando un estudio de caso único con dos unidades de análisis. El diseño del estudio de caso se hizo en función de sus cinco componentes: la pregunta de investigación, las proposiciones, las unidades de análisis, la lógica para enlazar los datos con las proposiciones y los criterios para interpretar los hallazgos (Ying, 2017).

La pregunta surge a partir de la intención del estudio de caso: evaluar la utilidad del mecanismo de consulta, por eso se plantea así: ¿Cómo contribuye el mecanismo de consulta al análisis de modelos UML representados en XMI?. La principal proposición afirma que el mecanismo de consulta QModel-XMI facilita el análisis de modelos UML representados en XMI. Como primera unidad de análisis se utilizó un contexto de evaluación del diseño de un producto software, usado para el control de la producción y el área logística en una empresa del sector industrial de la ciudad de Cartagena. La segunda unidad de análisis corresponde a un contexto de aprendizaje del curso de Programación Orientada a Objetos que ofrece el programa de Ingeniería de Sistemas de la Universidad de Cartagena.

El principal enlace lógico entre los datos y las proposiciones en este estudio de caso, se realizó aplicando la técnica de análisis denominada Modelo Lógico, que consiste en observar la coincidencia de los eventos empíricos observados y los eventos teóricos (Ying, 2017). Se seleccionó éste modelo de análisis, porque el estudio de caso se realiza para evaluar el mecanismo de consulta, demostrando que los eventos empíricos observados al utilizarlo coinciden con el propósito por el cual fue creado. Para argumentar la interpretación de los hallazgos se utilizó como principal estrategia, el análisis basado en proposiciones teóricas para hacer inferencias lógicas a partir de las propuestas conceptuales identificadas en la revisión de la literatura.

3. Resultados

En esta sección se presenta el diseño del mecanismo de consulta y su implementación representada en el prototipo QModel-XMI. Se explican las vistas arquitectónicas que lo describen. Además, se presentan los resultados del estudio de caso que se aplicó para evaluar la utilidad del mecanismo de consulta. Por último, se hace el análisis de los resultados obtenidos y se proponen algunos trabajos futuros.

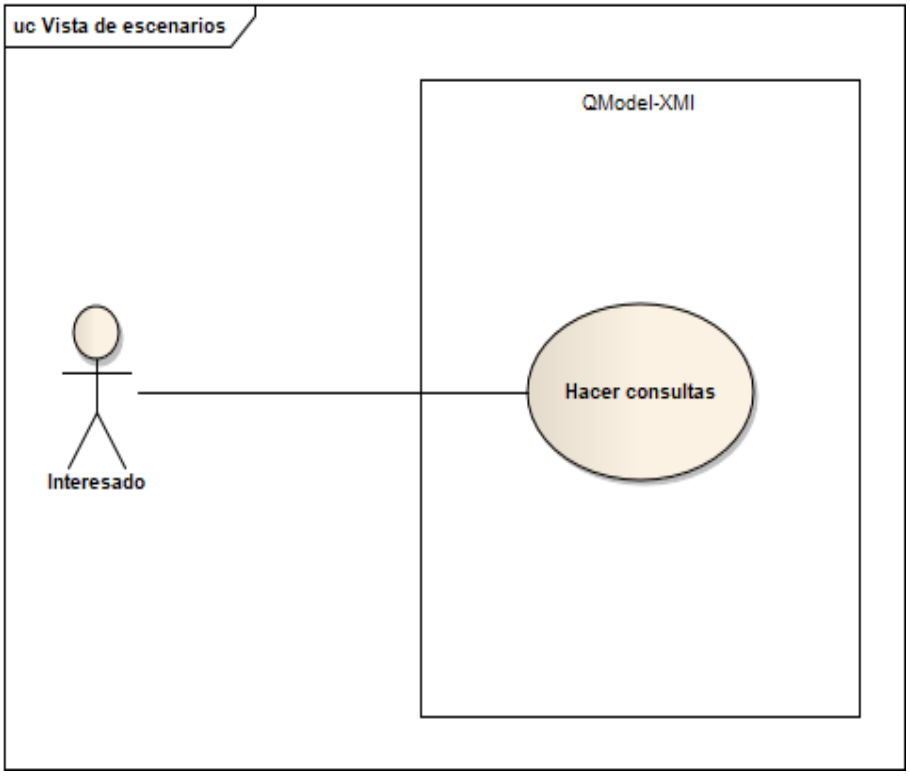
3.1. Diseño del mecanismo de consulta

El diseño del mecanismo de consulta se definió con base en el modelo conceptual y el modelo de caracterización establecido por Monroy et al. (2017), según el cual las consultas se clasifican en: 1) consulta para el cálculo de métricas, cuyo resultado se puede cuantificar y corresponde a una métrica de diseño. 2) Consulta simple: es aquella que puede ser representada por sentencias escritas en el lenguaje XQuery. Permiten identificar los

elementos que conforman el modelo, las relaciones que existen entre ellos y las propiedades que lo describen. 3) Consulta compuesta: involucra propiedades cualitativas individuales y colectivas del sistema y se pueden responder a partir de su semántica, pero no consiguen ser resueltas solamente con consultas XQuery, sino que además requieren de la ejecución de elaborados algoritmos. Dentro de este tipo de consultas están los siguientes grupos: a) Orientadas a establecer aspectos estructurales, b) orientadas a establecer aspectos de comportamiento, c) orientadas a establecer el uso de patrones, d) orientadas a identificar errores y e) orientadas a identificar la conformidad entre la arquitectura conceptual y la arquitectura concreta.

Para representar el diseño del mecanismo se utilizan la vista de escenarios, la vista lógica y la vista de procesos (Kruchten, 1995). El mecanismo se diseñó para ser usado por personas que tienen conocimientos básicos sobre ingeniería de software y UML. Cumple con el único propósito de responder preguntas escritas en lenguaje natural, que se hacen a modelos UML de productos software representados en XMI, como se observa en la vista de escenarios (ver figura 1), donde se aprecia al actor: Interesado, quien representa a la persona que desea hacer la respectiva consulta al mecanismo. Para cumplir con este propósito el mecanismo está estructurado en seis componentes (ver figura 2) y responde a la lógica descrita en la vista de procesos (ver figura 3).

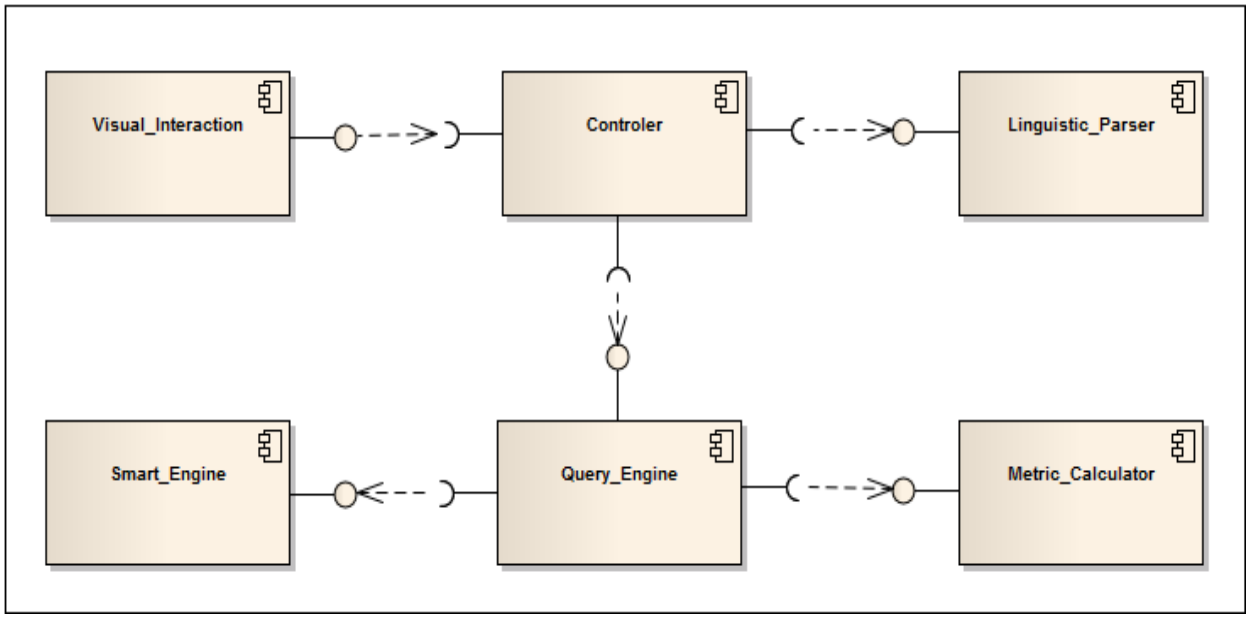
Figura 1
Vista de escenarios



Fuente: Los autores

Cada componente del mecanismo de consulta cumple con una responsabilidad, como se describe a continuación (ver figura 2). El componente "*Visual_Interaction*" permite la interacción entre el actor y el mecanismo de consulta. Brinda la interfaz gráfica para que el actor pueda seleccionar el archivo que contiene el modelo UML representado en XMI, ingresar la consulta en lenguaje natural y visualizar el resultado. El componente "*Controler*" orquesta la ejecución de la consulta. El componente "*Linguistic_Parser*" verificar la validez de la sintaxis y la gramática de la consulta ingresada por el actor. El componente "*Query_Engine*" analiza el tipo de consulta y ejecuta aquellas que son de tipo simple. El componente "*Metric_Calculator*" ejecuta las consultas de tipo métrica. El componente "*Smart_Engine*" ejecuta las consultas compuestas.

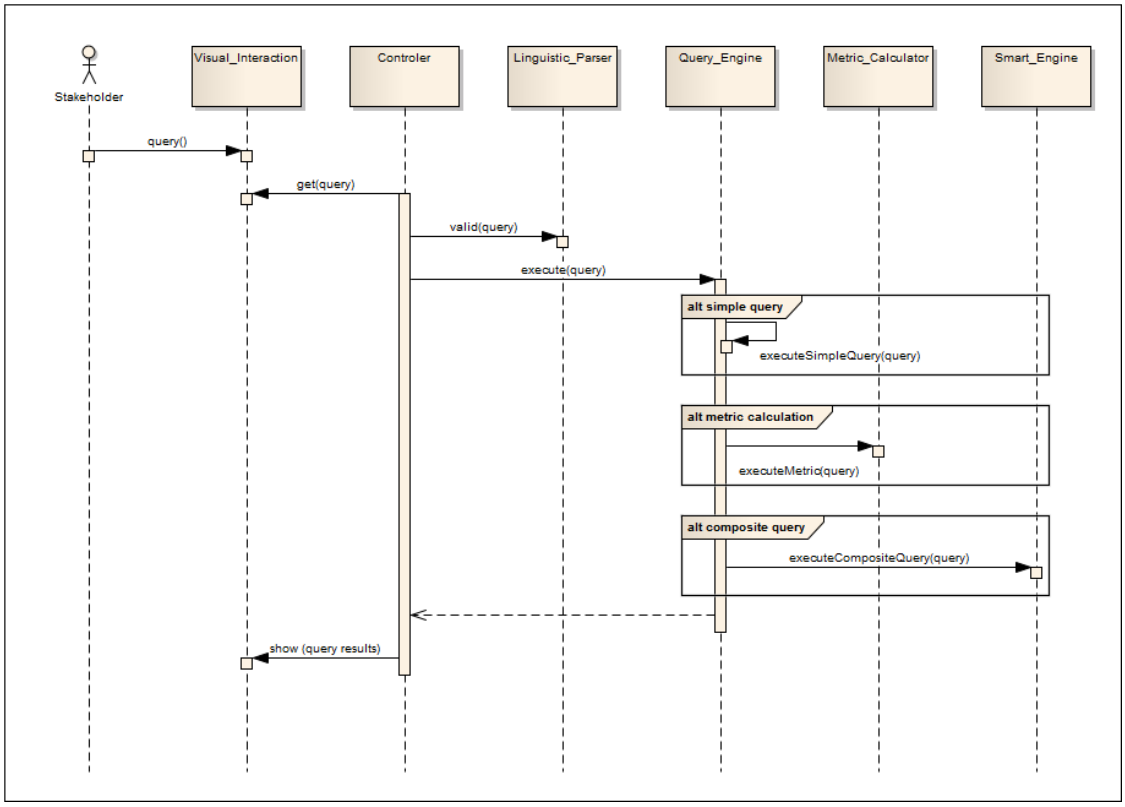
Figura 2
Vista lógica



Fuente: Los autores

El comportamiento del mecanismo de consulta obedece a la lógica representada en la figura 3. Inicialmente el actor selecciona el archivo XMI que contiene el modelo UML y registra la consulta. Luego el componente "*Visual_Interaction*" captura la consulta y la dirige al componente "*Controler*", quien solicita al componente "*Linguistic_Parser*" verificar la validez de la sintaxis y la gramática de la consulta ingresada por el actor. Si la consulta es válida el *Controler* invoca el servicio procesar consulta que ofrece el componente "*Query_Engine*", de lo contrario retorna un mensaje indicando que la consulta no es válida y el control de ejecución queda en el componente "*Visual_Interaction*". El componente "*Query_Engine*" determina el tipo de consulta. Si la consulta es simple, el mismo se encarga de ejecutarla. Si la consulta corresponde a un cálculo de métrica, invoca el servicio del componente "*Metric_Calculator*". Si la consulta es compuesta invoca el servicio del componente del "*Smart_Engine*". Finalmente el resultado de la consulta es presentado en la interfaz de usuario por el componente "*Visual_Interaction*".

Figura 3
Vista de procesos

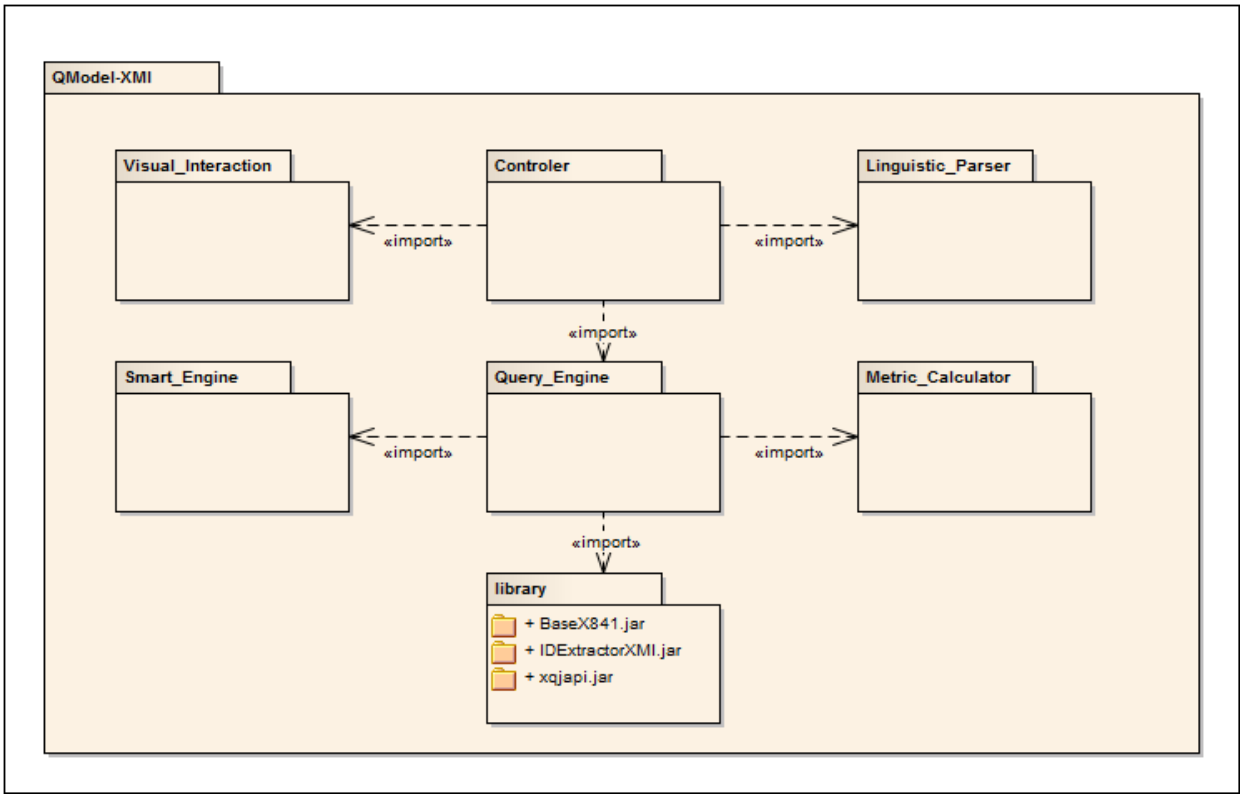


Fuente: Los autores

3.2. Implementación del mecanismo de consulta

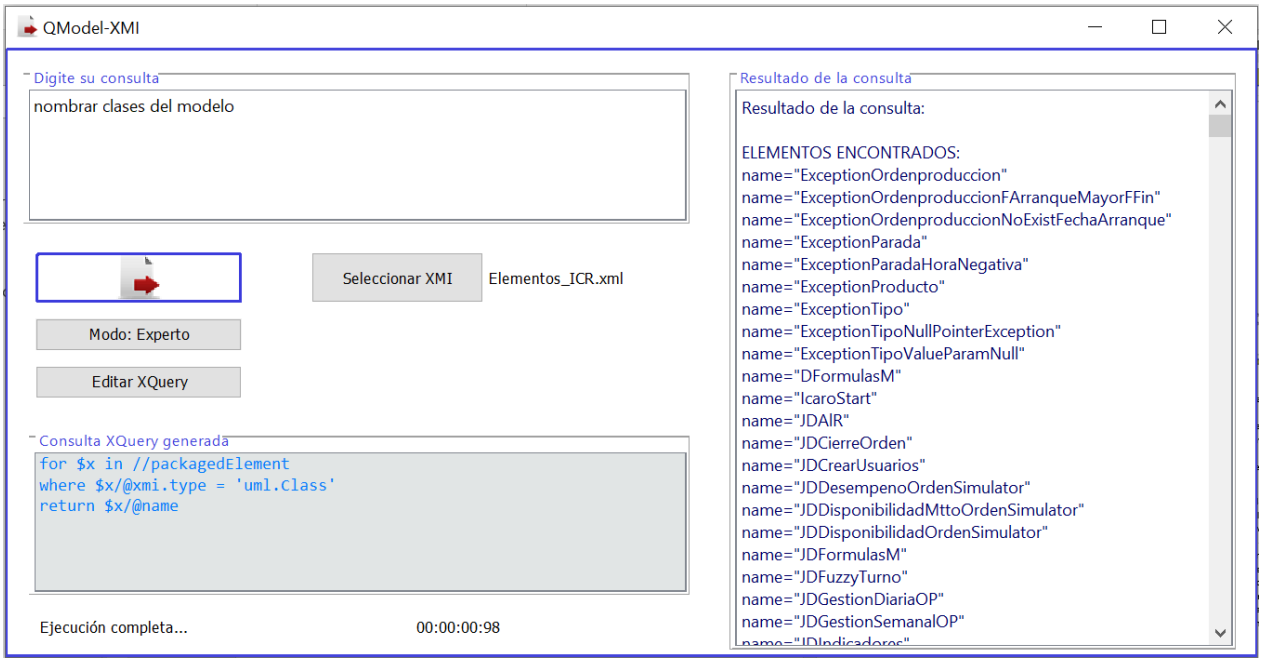
Se construyó el prototipo QModel-XMI a partir del diseño propuesto para el mecanismo de consulta. Se utilizó el lenguaje de programación Java para la implementación del prototipo. Para la realización de las consultas se usó el lenguaje *XQuey* y el motor *BaseX*. Para la primera versión del prototipo sólo se implementaron los componentes *Visual_Interaction*, *Controler*, *Linguistic_Parser*, *Query_Engine* y *Metric_ Calculator*. Esta versión del prototipo sólo acepta consultas en español definidas en Monroy et al. (2017) para modelos UML representados en diagramas de clase, diagramas de paquetes, diagramas de estados, diagramas de actividades y diagramas de casos de uso. La vista de desarrollo de QModel-XMI se representa en la figura 4. En la figura 5 se presenta la interfaz de usuario con resultados de una de las consultas hechas durante el estudio de caso.

Figura 4
Vista de desarrollo



Fuente: Los autores

Figura 5
Interfaz Gráfica de QModel-XMI



Fuente: Los autores

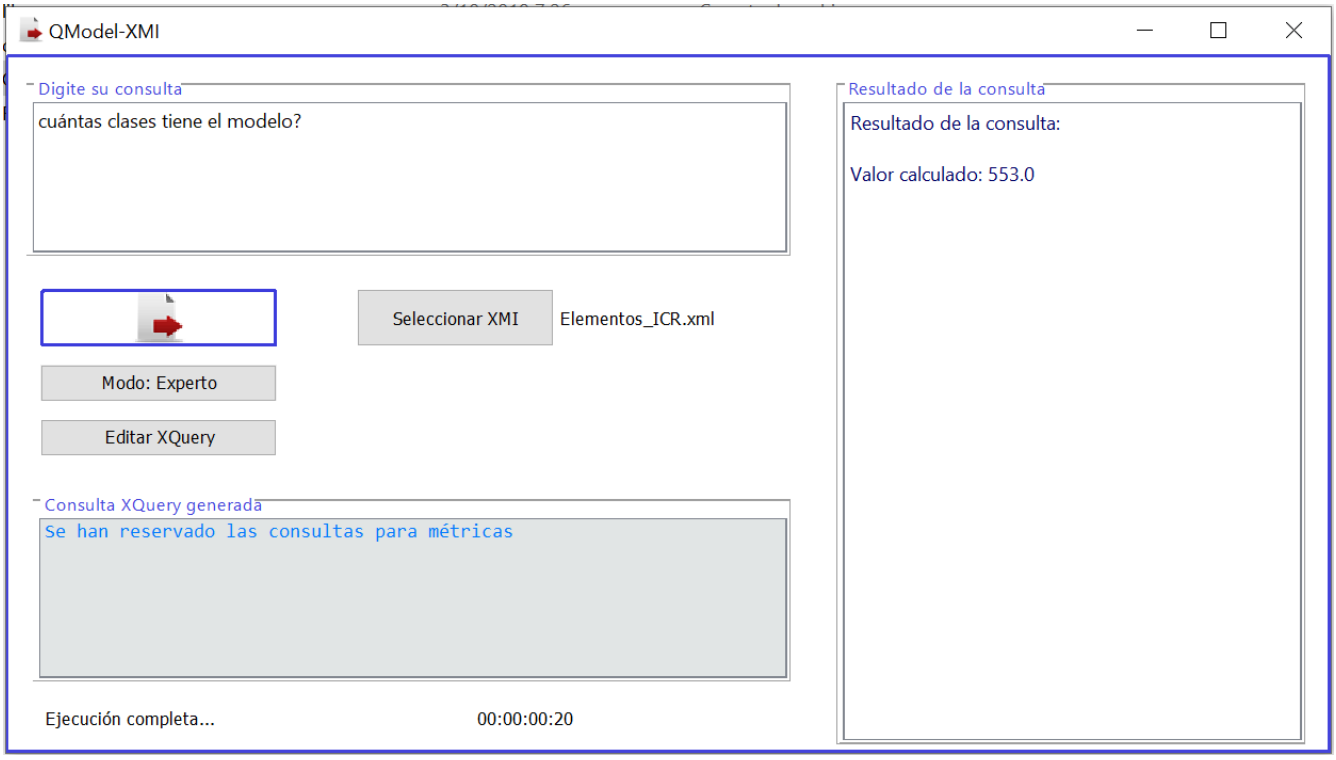
3.3. Estudio de caso

Los resultados del estudio de caso se presentan para cada unidad de análisis. En primera instancia la unidad de análisis correspondiente al contexto de evaluación del diseño de un producto software, usado para el control de la producción y el área logística en una empresa del sector industrial de la ciudad de Cartagena. Debido a que sólo se disponía del código fuente del sistema, fue necesario hacer un proceso de ingeniería inversa aplicando la metodología SAReM (Monroy, Ribon y Puello, 2018), para recuperar los modelos de diseño representados en UML. Para esta unidad de análisis se planteó como objetivo identificar posibles aspectos a mejorar en el diseño del producto software para garantizar su evolución.

Al realizar consultas con QModel-XMI se determinó que esta aplicación está conformada por 553 clases (ver figura 6) distribuidas en 62 paquetes (Ver figura 7), sólo cuenta con 3 interfaces, no tiene ninguna clase abstracta, y el uso de jerarquías de clases es reducido, lo que dificulta la expansión de funcionalidades, la reutilización de activos y la capacidad de análisis y mantenimiento del producto. También se identificó que existen muchas clases con nombres casi idénticos y al observar su código fuente se evidenció que sus responsabilidades eran muy parecidas. La interpretación y análisis de los resultados generó dos recomendaciones para la Empresa del sector industrial de la ciudad de Cartagena: 1) Hacer un análisis de clonación de código para depurar el sistema e identificar posibles activos reutilizables; y 2) Hacer un proceso de refactorización, asignado nombres más descriptivos a algunos módulos y clases, definiendo interfaces para los módulos responsables de la lógica de la aplicación, para garantizar el encapsulamiento, la reutilización y facilitar la extensibilidad y la capacidad de mantenimiento del sistema.

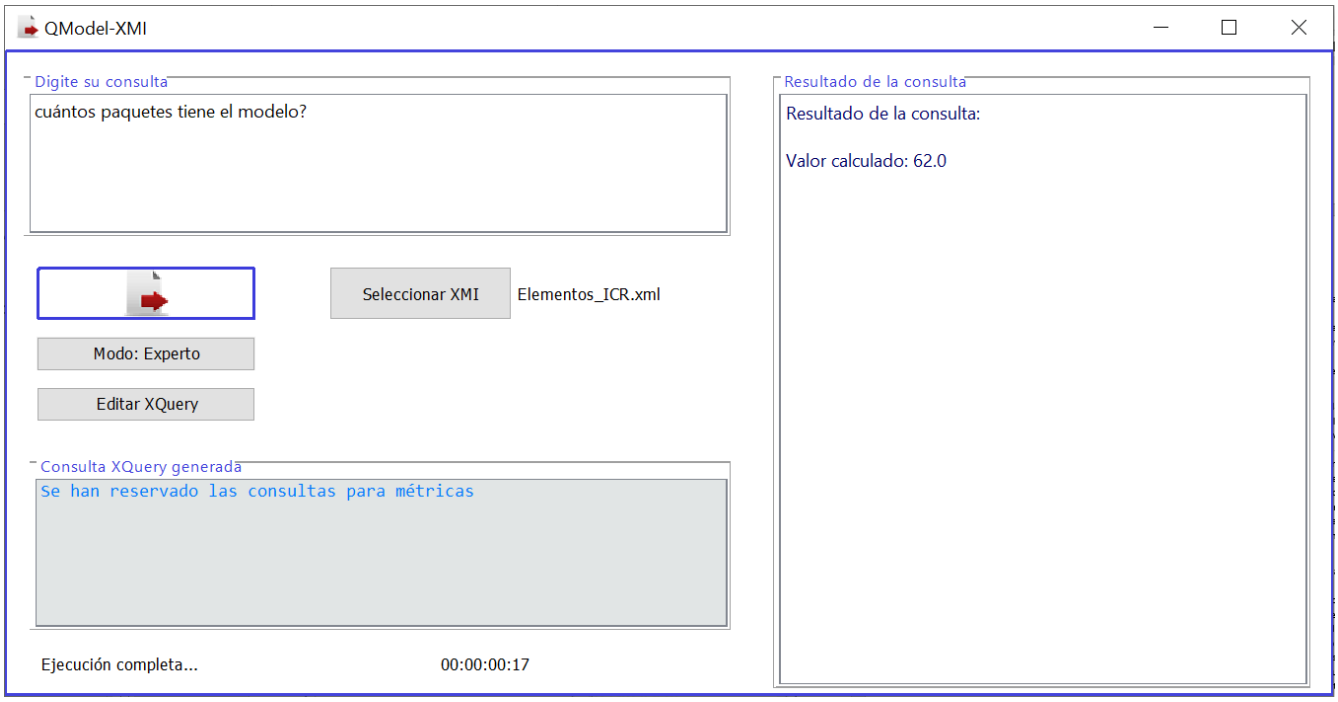
Figura 6

QModel-XMI: Cantidad de clases



Fuente: Los autores

Figura 7
QModel-XMI: Cantidad de paquetes



Fuente: Los autores

La segunda unidad de análisis correspondió al contexto de aprendizaje del curso de Programación Orientada a Objetos, que ofrece el programa de Ingeniería de Sistemas de la Universidad de Cartagena. Se desarrolló una actividad académica con el objetivo de que los estudiantes comprendan y apliquen el concepto de polimorfismo en soluciones prácticas. Se utilizaron los modelos recuperados a partir del código fuente de la aplicación JHotDraw (2007), obtenidos al aplicar la metodología SAReM (Monroy et al., 2018). Después de haber explicado la parte teórica, a los estudiantes se les entregó los modelos recuperados. Haciendo uso del prototipo QModel-XMI los estudiantes identificaron que este sistema está conformado por 1.217 clases distribuidas en 122 paquetes, en los que se definen 37 interfaces. También identificaron la jerarquía de clases que permite definir nuevos tipos de figuras y los métodos que deben implementarse para que la aplicación las integre aplicando el concepto de polimorfismo.

3.3. Análisis de los resultados

Los resultados del estudio de caso permiten afirmar que el mecanismo de consulta diseñado e implementado contribuye al análisis de modelos UML, porque sirvió para identificar aspectos a mejorar en el diseño de un producto software y para apoyar un proceso de aprendizaje significativo. El mecanismo diseñado es una propuesta conceptual que se limita a modelos representados en archivos XMI que correspondan a diagramas UML, por lo tanto, su capacidad está determinada por las posibilidades de consulta que brinda el lenguaje XQuery. El mecanismo por sí mismo no realiza ningún tipo de análisis, sólo sirve como instrumento de ayuda, que facilita el análisis que hace el experto en ingeniería de software, brindando la posibilidad de plantear preguntas en lenguaje natural a modelos UML.

En consecuencia, QModel-XMI constituye una nueva forma de realizar consultas sobre modelos UML representados en XMI, que facilita el proceso de análisis del producto software. Por otra parte, las propiedades semánticas del repositorio XMI dependen del modelo que representan y no se pueden apreciar directamente a partir de su estructura, porque requieren de análisis complementarios para lograr la interpretación, por lo tanto,

se plantea como trabajo futuro la implementación del componente *Smart_Engine* aplicando técnicas de inteligencia artificial. De igual forma, se propone mejorar la capacidad del componente *Linguistic_Parser* para que acepte otros lenguajes y para hacer más flexible la forma como se formulan las preguntas.

4. Conclusiones

Se diseñó un producto software a partir del modelo conceptual establecido en Monroy et al. (2017) y se implementó el prototipo funcional QModel-XMI. Este prototipo sirvió para hacer análisis de un producto software usado por una empresa del sector industrial de la ciudad de Cartagena. También sirvió para que un grupo de estudiantes analizaran el producto JHotDraw (2007), con el propósito de extender sus funcionalidades aplicando el concepto de polimorfismo. Se utilizó el mecanismo lingüístico aprovechando las capacidades descriptivas disponibles en UML, que registran la estructura, el comportamiento y demás propiedades del sistema. Las preguntas se formulan en lenguaje natural para facilitar el trabajo de los diferentes actores que requieran hacer análisis sobre modelos UML. El mecanismo de consulta implementado sólo interpreta preguntas formuladas en español, sobre modelos UML representados en diagramas de clase, diagramas de paquetes, diagramas de estados, diagramas de actividades y diagramas de casos de uso. La capacidad de QModel-XMI está determinada por las posibilidades de consulta que brinda el lenguaje XQuery.

Referencias bibliográficas

Acretoaie, V., & Störrle, H. (2012). MQ-2: a tool for prolog-based model querying. In Proc. co-located Events 8th Eur. Conf. on Modelling Foundations and Applications (ECMFA'12) (pp. 328-331).

Alves, T. L., Hage, J., & Rademaker, P. (2011, September). A comparative study of code query technologies. In 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation (pp. 145-154). IEEE.

Beeri, C., Eyal, A., Kamenkovich, S., & Milo, T. (2008). Querying business processes with BP-QL. Information Systems, 33(6), 477-507.

Bergmann, G., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., & Varró, D. (2012, May). Integrating efficient model queries in state-of-the-art EMF tools. In International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (pp. 1-8). Springer, Berlin, Heidelberg.

Beyer, D., Noack, A., & Lewerentz, C. (2005). Efficient relational calculation for software analysis. IEEE Transactions on Software Engineering, 31(2), 137-149.

Bruneliere, H., Cabot, J., Jouault, F., & Madiot, F. (2010, September). MoDisco: a generic and extensible framework for model driven reverse engineering. In Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 173-174).

Canfora, G., Di Penta, M., & Cerulo, L. (2011). Achievements and challenges in software reverse engineering. Communications of the ACM, 54(4), 142-151.

De Roover, C., Noguera, C., Kellens, A., & Jonckers, V. (2011, August). The SOUL tool suite for querying programs in symbiosis with Eclipse. In Proceedings of the 9th International Conference on Principles and Practice of Programming in Java (pp. 71-80).

IEEE (2016). SWEBOK Guide to the Software Engine Body of Knowledge. Version 3.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The unified software development process. Addison-Wesley Longman Publishing Co., Inc.

JHotDraw Org (2007), JHotDraw as Open-Source Project. Recuperado de: http://www.jhotdraw.org/

Kitchenham, B. A., Budgen, D., & Brereton, O. P. (2011). Using mapping studies as the basis for further research–a participant-observer case study. Information and Software Technology, 53(6), 638-651.

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. IEEE software, 12(6), 42-50.

Liepiņš, R. (2012, September). Library for model querying: IQuery. In Proceedings of the 12th Workshop on OCL and Textual Modelling (pp. 31-36).

Monroy, M. E., Arciniegas, J. L., & Rodríguez, J. C. (2017). Mecanismo de Consulta para el Análisis de Arquitecturas Recuperadas. Información tecnológica, 28(5), 87-100.

Monroy, R. M., Ribon, R., & Puello, P. (2018). A Methodological Approach for Software Architecture Recovery. Indian Journal of Science and Technology, 11(21), 1-8.

Sakr, S., & Awad, A. (2010, April). A framework for querying graph-based business process models. In Proceedings of the 19th international conference on World wide web (pp. 1297-1300).

Störrle, H. (2013, July). MOCQL: a declarative language for ad-hoc model querying. In European Conference on Modelling Foundations and Applications (pp. 3-19). Springer, Berlin, Heidelberg.

Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Szatmári, Z., & Varró, D. (2013). An Integrated Development Environment for Live Model Queries. Science of Computer Programming.

Urma, R. G., & Mycroft, A. (2015). Source-code queries with graph databases—with application to programming language usage and evolution. Science of Computer Programming, 97, 127-134.

Verbaere, M., Godfrey, M. W., & Girba, T. (2008, June). Query Technologies and Applications for Program Comprehension (QTAPC 2008). In 2008 16th IEEE International Conference on Program Comprehension (pp. 285-288). IEEE.

Wu, J., Holt, R. C., & Winter, A. (2002). Towards a common query language for reverse engineering. Fachberichte Informatik, 8.

Yin, R. K. (2017). Case study research and applications: Design and methods. Sage publications.