UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

FACULTAD CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA EN SISTEMAS

Proyecto de Investigación previo a
la obtención del título de Ingeniero
en Sistemas.

Título del Proyecto de Investigación:

"LIBRERÍA JAVASCRIPT PARA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS USANDO UN LENGUAJE DE SÍMBOLOS"

Autor:

Dúval Ricardo Carvajal Suárez

Director del Proyecto de Investigación:

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

MACHALA - EL ORO - ECUADOR

2022

DECLARACIÓN DE AUDITORIA Y CESIÓN DE DERECHOS

Yo, **Dúval Ricardo Carvajal Suárez** declaró que la investigación aquí descrita es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

La Universidad Técnica Estatal de Quevedo, puede hacer uso de los derechos correspondientes a este documento, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento por la normativa institucional vigente.

Dúval Ricardo Carvajal Suárez

070638894-9

CERTIFICACIÓN DE CULMINACIÓN DEL PROYECTO DE INVESTIGACIÓN

El suscrito, Ing. Gleiston Ciceron Guerrero Ulloa Msc. docente de la Universidad Técnica Estatal de Quevedo, certifica que el estudiante Dúval Ricardo Carvajal Suárez, realizó el Proyecto de Investigación de grado titulado "LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS", previo a la obtención del título de Ingeniero en Sistemas, bajo mi dirección, habiendo cumplido con las disposiciones reglamentarias establecidas para el efecto.

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

DIRECTOR DEL PROYECTO DE INVESTIGACIÓN

CERTIFICADO DEL REPORTE DE LA HERRAMIENTA DE PREVENCIÓN DE COINCIDENCIA Y/O PLAGIO ACADÉMICO

El suscrito Ing. Gleiston Ciceron Guerrero Ulloa Msc. certifica que:

El proyecto de investigación titulado "LIBRERÍA JAVASCRIPT QUE PERMITA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS", ha sido analizado mediante la herramienta de URKUND y presentó resultados satisfactorios.

(AQUÍ VA LA CAPTURA DEL % URKUND)

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

DIRECTOR DEL PROYECTO DE INVESTIGACIÓN

CERTIFICACIÓN DE APROBACIÓN DEL PROYECTO DE INVESTIGACIÓN

Titulo:

"LIBRERÍA JAVASCRIPT PARA INTERPRETAR LA ESCRITURA DE LOS CASOS DE USO EXTENDIDOS USANDO UN LENGUAJE DE SÍMBOLOS"

Presentado al Consejo Directivo como requisito previo a la obtención del título de Ingeniero en Sistemas.

Aprobado por:

AGRADECIMIENTO

Debe ser redactada con frases cortas. Hasta máximo una página y centrado el texto

Duval Ricardo Carvajal Suárez

DEDICATORIA

Debe ser redactada con frases cortas. Hasta máximo una página y centrado el texto

Duval Ricardo Carvajal Suárez

RESUMEN Y PALABRAS CLAVES

Debe contener un resumen en español, que incluye una descripción del objeto de investigación en forma concisa y una breve descripción de los métodos o procedimientos utilizados, indicando las conclusiones obtenidas. El resumen no debe constar de más de 300 palabras redactadas en un solo párrafo.

Palabras claves: 1, 2, 3

ABSTRACT AND KEYWORDS

It must contain a summary in English, which includes a concise description of the object of the The abstract must include a concise description of the object of research and a brief description of the methods or procedures used, indicating the conclusions obtained. The abstract should not exceed 300 words in a single paragraph.

Keywords: 1, 2, 3

Tabla de contenido

Introducción

El desarrollo de software se vuelve cada vez más indispensable en todos los aspectos de la vida cotidiana, por lo cual se necesitan herramientas que ayuden a agilizar la construcción de diferentes tipos de sistemas de información basados en computadoras (SI) [1]. Utilizando el Lenguaje unificado de modelado (Unified Modeling Language: UML), se pretenden utilizar el diagrama de clases como elemento principal para identificar de manera sencilla como interactuarán todos los objetos del sistema [2]. El uso de diagrama de clases constituye una buena solución del modelado de SI, en especial, aquellos proyectos en dónde los requisitos están cambiando constantemente [3], por lo que necesario una herramienta que permita la actualización de sus modelos con el menor de los esfuerzos.

CAPÍTULO I
CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

1.1. Problema de investigación.

1.1.1. Planteamiento del problema.

El mayor enigma en el desarrollo de un software surge al diseñar su estructura principal por no conocer totalmente la lógica de negocio que se va a automatizar [4]. Para disminuir el impacto de esta problemática se sugiere el uso de diagramas de clases, es decir que a través de las entidades, sus atributos y métodos facilita las tareas de los diseñadores informáticos y también reduce las tareas de diseño del software [5].

Para construir un diagrama de clases se deben manipular herramientas que permitan crear cada componente, con pocos conocimientos sobre cada objeto que interviene en la construcción del mismo [5]. Además, al momento de continuar con el desarrollo del sistema surgirán modificaciones en requisitos que ya fueron planteados al inicio del proyecto, lo cual provoca malestar en los desarrolladores, ya que; se debe editar los objetos del diagrama sin tener algún tipo de información que permita recordar lo que ya estaba desarrollado [6].

Para obtener información que ya fue analizada sobre le diagrama de clases que se tenga creado, se pretende utilizar un lenguaje simbólico que se encuentra en la herramienta TDDTIoTS (https://aplicaciones.uteq.edu.ec/tddt4iots), este lenguaje no notifica mensajes de advertencia sobre el estado de los textos que describen los casos de uso, además se necesita una documentación más detallada sobre cómo utilizar cada símbolo al momento de describir las acciones de cada caso de uso y no tener inconvenientes al tratar de interpretar cada párrafo. En definitiva, existen varios inconvenientes para crear un diagrama de clases que permita identificar de forma sencilla todo el proceso que se quiere realizar con e software, provocando malos entendidos al momento de empezar a desarrollar el código del sistema.

Diagnóstico.

...

Pronóstico.

...

1.1.2. Formulación del problema.

¿Como interpretar las descripciones de los casos de uso extendidos escritos en el lenguaje de símbolos usado en la herramienta TDDT4IoTS para generar la estructura de un diagrama de clases?

1.1.3. Sistematización del problema.

1. ¿Como se obtendrá el diagrama de clases generado por las descripciones de los casos de uso extendidos?
2. ¿Como ayudar a los usuarios a mejorar la escritura de las descripciones de los casos de uso extendidos escritos en un lenguaje de símbolos usado por la herramienta TDDT4IoTS?
3. ¿De que forma se evaluara el correcto funcionamiento del producto final de este proyecto de investigación?

1.2. Objetivos.

La problematización de este proyecto ha llevado a plantearse los siguiente objetivos.

1.2.1. Objetivo General.

Desarrollar una librería JavaScript que interprete las descripciones de los casos de uso usado la herramienta TDDT4IoTS para visualizar la estructura del diagrama de clases pertinente.

1.2.2. Objetivos Específicos.

1. Generar una estructura JSON y XML con la información obtenida por las descripciones de los casos de uso extendidos para crear un diagrama de clases mediante el uso librerías javascript para crear diagramas.
2. Retroalimentar a los usuarios sobre la mala escritura de las descripciones de los casos de uso extendidos escritos en un lenguaje de símbolos usado por la herramienta TDDT4IoTSs.

3. Evaluar la librería JavaScript con la elaboración de diagramas de clases a partir de casos de uso extendidos correspondientes a sistemas de información reales.

1.3. Justificación.

La construcción de los diagramas de clases tienden a contener información redundante o aveces inconsistente para especificar de forma técnica los requisitos funcionales y no funcionales de un sistema informático. Debido a esto se propone el desarrollo de una librería JavaScript, que mediante el lenguaje de símbolos usado por la herramienta TDDT4IoTs para interpretar las descripciones de los casos de uso extendidos, permitan generar de forma automática el diagrama de clases pertinente a los casos de uso planteados.

Ademas esta librería propone generar 2 tipos de estructuras que son JSON y XML que son usadas a nivel global por varias tecnologías de desarrollo, permitiendo combinar el uso de la estructura generada por esta herramienta con las librerías externas dedicadas a la creación de diferentes tipos de diagramas, obteniendo como resultado una grafico visual en relación a la información devuelta por el producto final de este proyecto.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

En este capítulo, se expondrán los trabajos relacionados con el presentado en este documento que se han identificado en la literatura, y que demuestran que el trabajo propuesto en este documento, aporta algo. Además se contextualiza el trabajo y define los términos novedosos y de poco dominio para los investigadores y para la comunidad.

2.1. Marco conceptual.

En esta sección, se detalla los modelos teóricos, conceptos, argumentos o definiciones que se han desarrollado o investigado en relación con el tema en particular.

2.1.1. JSON.

Javascript Object Notation (JSON) es un formato ligero de intercambio de datos. Consisten en asociación de nombres y valores. A pesar de ser independiente del lenguaje de programación, es admitido en una gran cantidad de lenguajes de programación. Se basa en un subconjunto del Estándar de lenguaje de programación JavaScript [7].

2.1.2. XML.

Es un lenguaje de marcado similar a HTML. Significa Extensible Markup Language y pertenece a la especificación W3C como lenguaje de marcado de propósito general. Esto significa que, a diferencia de otros lenguajes de marcado, XML no está predefinido, por lo que debe definir su propio marcado. El objetivo principal del lenguaje es compartir datos entre diferentes sistemas, como Internet [8].

2.1.3. Compiladores.

Están diseñados para traducir un fragmento de código escrito en un de lenguaje de programación a lenguaje de máquina que es, el que puede entender la computadora. El compilador analiza el código fuente en busca de errores antes de la traducción. Si se detecta un error, el compilador notificar al autor del código para que pueda arreglarlo.. Además, los compiladores modernos o entornos de desarrollo (IDE), puede sugerir soluciones para algunos tipos de errores usando métodos de corrección de errores [9].

2.1.4. Lenguajes de modelado.

Representan una serie de requisitos basados en la construcción de elementos visuales para definir estructuras y comportamientos que tendrán los sistemas computarizados. UML (Lenguaje Unificado de Modelado) a través del mecanismo de perfilado, se han basado históricamente en notaciones gráficas. UML mediante el mecanismo de perfiles, maximiza la comprensión humana y facilita la comunicación entre las partes interesadas como son el cliente y desarrollador [10].

También existen lenguajes de modelado personalizados para distintas áreas, como por ejemplo en [11] proponen un lenguaje de modelado conceptual multinivel al denominan ML2 (Lenguaje de Modelado Multinivel). El lenguaje está orientado al modelado conceptual multinivel (de dominio) y pretende cubrir un amplio conjunto de dominios multiniveles. En el diseño de ML2 sigue un enfoque basado en principios, definiendo su sintaxis abstracta para reflejar una teoría formal para el modelado multinivel que se fue desarrollado previamente.

2.2. Marco referencial.

2.2.1. Lenguaje de restricciones de objetos para la generación de código a partir de modelos de actividad

UML(Unified Modeling Language) es un lenguaje que utiliza el diagrama de actividad para modelar el flujo de trabajo y el flujo de objetos en un sistema [12]. UML no es un lenguaje totalmente formal, su semántica no está totalmente formalizada ocasionando un escenario donde la presentación precisa del modelo es difícil. Por lo tanto, siempre que se utilicen diagramas de actividades, o cualquier diagrama UML, para la generación de código, se recomienda complementarlo con lenguajes de especificación como el lenguaje de restricciones de objetos OCL (Object Constraint Language) [13].

2.2.2. Desarrollo de módulos de software generativo para el diseño orientado al dominio con un lenguaje específico de dominio basado en anotaciones

En la terminología del diseño orientado a objetos [14], un módulo de dominio es un paquete. Los actuales marcos de software DDD (Object-oriented domain-driven design) han utilizando una forma simple de DSL(Domain-specific language) interno para construir el modelo de dominio y utilizar este modelo como entrada para generar un prototipo de software. El DSL interno que utilizan se conoce más formalmente como lenguaje específico del dominio basado en anotaciones (aDSL) [15].

2.2.3. Modelado para la localización de características en modelos de software: tanto la generación de código y los modelos interpretados.

Evaluando LDA (Latent Dirichlet Allocation), cada caso de estudio utiliza un tipo diferente de modelos de software: modelos de software para la generación de código, y modelos de software para interpretación. El primer caso de estudio pertenece a un líder mundial en fabricación de trenes, construcciones y Auxiliar de Ferrocarriles. Una empresa que formaliza los productos fabricados en modelos de software utilizando un lenguaje específico de dominio DSL. Los modelos de software se utilizan para generar el firmware que controla sus trenes [16].

El segundo estudio de caso pertenece a un videojuego comercial, Kromaia, que utiliza modelos de software para razonar sobre el sistema, realizar validaciones y definir el contenido del juego, como los jefes, los mundos y los objetivos. En Kromaia los modelos de software se utilizan para la interpretación. Así, el contenido definido en los modelos se lee e interpreta cuando se lanza el juego (sin alterar el código fuente del videojuego). El videojuego se ha lanzado en todo el mundo en dos plataformas diferentes (PlayStation 4 y STEAM) y en 8 idiomas diferentes. [16].

CAPÍTULO III
BIBLIOGRAFÍA

[1] A. Jeyaraj, "Delone & mclean models of information system success: Critical meta-review and research directions," *International Journal of Information Management*, vol. 54, Oct. 2020, ISSN: 02684012. DOI: 10.1016/J.IJINFOMGT.2020.102139.

[2] R. Karampure, C. Y. Wang, and Y. Vashi, "Uml sequence diagram to axiomatic design matrix conversion: A method for concept improvement for software in integrated systems," *Procedia CIRP*, vol. 100, pp. 457–462, 2021, ISSN: 22128271. DOI: 10.1016/J.PROCIR.2021.05.104.

[3] A. Abdalazeim and F. Meziane, "A review of the generation of requirements specification in natural language using objects uml models and domain ontology," *Procedia CIRP*, vol. 189, pp. 328–334, 2021, ISSN: 22128271. DOI: 10.1016/J.PROCS.2021.05.102.

[4] R. Mörbitz and H. Vogler, "Weighted parsing for grammar-based language models over multioperator monoids," *Information and Computation*, vol. 281, Dec. 2021, ISSN: 10902651. DOI: 10.1016/J.IC.2021.104774.

[5] W. Behutiye, P. Karhapää, L. López, *et al.*, "Management of quality requirements in agile and rapid software development: A systematic mapping study," *Information and Software Technology*, vol. 123, Jul. 2020, ISSN: 09505849. DOI: 10.1016/J.INFSOF.2019.106225.

[6] G. Islam and T. Storer, "A case study of agile software development for safety-critical systems projects," *Reliability Engineering and System Safety*, vol. 200, Aug. 2020, ISSN: 09518320. DOI: 10.1016/J.RESS.2020.106954.

[7] P. Bourhis, J. L. Reutter, and D. Vrgoč, "Json: Data model and query languages," *Information Systems*, vol. 89, Mar. 2020, ISSN: 03064379. DOI: 10.1016/J.IS.2019.101478.

[8] A. Khalili, "An xml-based approach for geo-semantic data exchange from bim to vr applications," *Automation in Construction*, vol. 121, Jan. 2021, ISSN: 09265805. DOI: 10.1016/J.AUTCON.2020.103425.

[9] A. Alwabel, "Coedit: A novel error correction mechanism in compilers using spelling correction algorithms," *Journal of King Saud University - Computer and Information Sciences*, 2021, ISSN: 22131248. DOI: 10.1016/J.JKSUCI.2021.02.010.

[10] L. Addazi and F. Ciccozzi, "Blended graphical and textual modelling for uml profiles: A proof-of-concept implementation and experiment," *Journal of Systems and Software*, vol. 175, May 2021, ISSN: 01641212. DOI: 10.1016/J.JSS.2021.110912.

[11] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi, and V. A. Carvalho, "Multi-level conceptual modeling: Theory, language and application," *Data and Knowledge Engineering*, vol. 134, Jul. 2021, ISSN: 0169023X. DOI: 10.1016/J.DATAK.2021.101894.

[12] J.-W. Maessen and X. Shen, "Improving the java memory model using crf," 2000.

[13] E. V. Sunitha and P. Samuel, "Object constraint language for code generation from activity models," *Information and Software Technology*, vol. 103, pp. 92–111, Nov. 2018, ISSN: 09505849. DOI: 10.1016/J.INFSOF.2018.06.010.

[14] R. Eshuis, "Feature-oriented engineering of declarative artifact-centric process models," *Information Systems*, vol. 96, Feb. 2021, ISSN: 03064379. DOI: 10.1016/J.IS.2020.101644.

[15] D. M. Le, D. H. Dang, and V. H. Nguyen, "Generative software module development for domain-driven design with annotation-based domain specific language," *Information and Software Technology*, vol. 120, Apr. 2020, ISSN: 09505849. DOI: 10.1016/J.INFSOF.2019.106239.

[16] F. Pérez, R. Lapeña, A. C. Marcén, and C. Cetina, "Topic modeling for feature location in software models: Studying both code generation and interpreted models," *Information and Software Technology*, vol. 140, Dec. 2021, ISSN: 09505849. DOI: 10.1016/J.INFSOF.2021.106676.

CAPÍTULO IV
ANEXOS