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 DETECTAR ERRORES EN LA ESCRITURA DE LOS CASOS DE USO ESCRITOS EN UN LENGUAJE DE SÍMBOLOS"

Autor:

Dúval Ricardo Carvajal Suárez

Director del Proyecto de Investigación:

Ing. Gleiston Ciceron Guerrero Ulloa Msc.

QUEVEDO - LOS RIOS - 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

Los sistemas informáticos se están volviendo cada vez más indispensables en todos los aspectos de la vida cotidiana. Considerando que la tecnología cambia aceleradamente, y que los usuarios requieren soluciones rápidas, son las principales razones por las que la Ingeniería de Software busca disminuir el tiempo en implementar los sistemas requeridos [1]. Para la construcción de una aplicación informática que cumpla con los requisitos del cliente y que sea eficiente se necesitan aplicar varios aspectos técnicos [2], que la Ingeniería de Software detalla. Por ejemplo: estilo arquitectural, tecnologías a utilizar, infraestructura, escalabilidad, entre otros [3].

El campo de la Ingeniería de Software es análogo al campo de las construcción de obras civiles. Los documentos de diseño de un Software guardan semejanza con los documentos de diseño de un edificio (por ejemlo). Para la construcción de un edificio se deben analizar las principales características físicas del terreno sobre el cual se va a construir, mientras que, para la construcción del software se debe analizar la/s plataformas sobre las cuales se ejecutará el software. En el caso de un edificio, se deben detallar aspectos de la construcción ambientes y sus dimensiones precisas, columnas, puertas, ventanas, etc., así mismo se necesita documentar con precisión los módulos y las funcionalidades y alcance de cada uno de ellos, los elementos que conformarán el producto de software a desarrollar, además del comportamiento que deben tener todos los artefactos de un software [4]. Lo que se puede mencionar como diferencias que estos elementos de software pueden ser construidos para ser reutilizados en otros sistemas [5].

*Unified Modeling Languag*e (UML) con su traducción al español Lenguaje Unificado de Modelado, Uno de los lenguajes más conocidos por los desarrolladores de software para la representación de los requisitos o necesidades del cliente/usuario final (en adelante solamente usuario) son los diagramas UML [6]. Sin embargo, estos diagramas no son fáciles de comprender para todos los usuarios, ya que, los elementos que intervienen y que darán solución al problema planteado son representados mediante símbolos y algo de texto. UML se utiliza específicamente en la industria del software para especificar, visualizar, construir y documentar los artefactos de un sistema de software [7]. UML se encuentra definido oficialmente por el *Object Management Group* (OMG) con su traducción al español Grupo de Administración de Objetos [8]. Algunos investigadores afirman que, a partir de modelos

dinámicos (diagramas de casos de uso) y estáticos (diagramas de clases) se pueden generar otros modelos UML de manera más eficiente [9], y así obtener software de calidad.

Los estándares ISO/IEC 9126-1, especifican la calidad del software tanto interna, externa y en uso del producto. De ahí la importancia de considerar seguir estándares que permitan asegurar un lenguaje común para el entendimiento de todos los involucrados en el proyecto de software [10].

Los diagramas de casos de uso son una representación del sistema mediante los usuarios (actores) y sus requisitos (casos de uso). Son una buena herramienta para representar los requisitos iniciales del sistema, y pueden ser fácilmente entendidos por todos los involucrados [11]. Sin embargo en un diagrama de caso de uso no se especifican los detalles sino que son los requisitos a *grano grueso* del sistema. La información de cada caso de uso se detalla mediante condiciones previas (precondiciones), poscondiciones y secuencia de eventos normales. Además, incluye la secuencia alternativa de eventos en caso de excepciones o condiciones específicas y las postcondiciones que se deben tomar en cuenta al momento de estar utilizando el software [12]. Estas especificaciones de los casos de uso se denominan los requisitos a *grano fino*, y se plasman en documentos denominados *casos de uso extendidos*. Estos requisitos de grano fino pueden ser comprimidos en los diagramas de clases [13].

Los diagramas de clases en UML representan la estructura estática de los objetos y sus posibles conexiones dentro del software. Se lo utiliza para ilustrar el punto de vista estático, exponiendo un conjunto de clases, interfaces y relaciones [14]. Se lo desarrolla durante la fase de elaboración y se perfecciona posteriormente en la fase de construcción [8], representando un modelo del dominio del sistema. Además, es uno de los diagramas más útiles en UML, ya que trazan claramente la estructura de un sistema concreto al modelar sus clases con sus atributos y operaciones, y las relaciones entre clases [14].

Los diagramas de clases son utilizados por muchas herramientas (Rational Rose[1], MagicDraw[2], ArgoUML[3], por mencionar algunas) para la generación del código de software a partir de los diagramas de clases, sin embargo ninguna considera la documentación de los casos de uso extendidos, por lo tanto, tampoco utilizan las especificaciones a grano fino para obtener diagramas UML que ayudan a generar el código como lo es el diagrama de clases.

[1]https://www.ibm.com/docs/es/rsas/7.5.0?topic=migration-rational-rose-model
[2]https://www.magicdraw.com/
[3]https://argouml-tigris-org.github.io/

Existe una herramienta TDDT4IoTS [15] que está en fase de prueba. En esta herramienta es posible que el analista escriba los requisitos detallados de los casos de uso, y, mediante el uso de un lenguaje de símbolos (SymLen) marcar las palabras que representan nombres de clases, atributos, métodos, interfaces y otros elementos de los diagramas de clases. A su vez, presenta los casos de uso extendidos en lenguaje natural exactamente como el usuario expresó los requisitos detallados que el sistema debe cumplir. De estas especificaciones de grano fino es posible generar los diagramas de clases, y el código de software correspondiente. Al ser una herramienta en su fase inicial, el interprete implementado en TDDT4IoTS no presenta una retroalimentación eficiente, simplemente presenta los artefactos resultado, sin especificar los posibles errores en el uso del lenguaje. Además el actual intérprete no se puede utilizar fácilmente en otra herramienta.

En el presente trabajo se propone una librería escrita en JavaScript denominada *Armadillo* (Armadillo.js). Armadillo permite interpretar las descripciones de los casos de uso extendidos escritas en SymLen. Armadillo le permite al analista conocer los posibles errores en el uso de SymLen, con el objetivo de obtener de forma automática un diagrama de clases pertinente a la información proporcionada por el usuario. Esto permitirá potenciar el uso de la herramienta TDDT4IoTS. Además, Armadillo le permitirá eliminar todos los inconvenientes del intérprete que actualmente utiliza TDDT4IoTS que han sido detectados.

El resto de este documento está organizado por capítulos. En el capítulo I se contextualiza la investigación, donde se especifica la problemática a resolver, los objetivos que se lograron y la hipótesis que se demostró, además de una argumentación que justifica que se haya realizado esta investigación. En el capítulo II se presenta la fundamentación teórica, en la que se presentan los principales trabajos relacionados con el trabajo propuesto en este documento, conceptos y definiciones importantes, y las limitaciones, herramientas y tecnologías se expresan en el marco contextual de la investigación.

CAPÍTULO I

CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

1.1. Problema de investigación.

1.1.1. Planteamiento del problema.

Con el aumento en la complejidad de los productos de software los desarrolladores de sistemas informáticos han encontrado la manera de mejorar el desarrollo de software, mediante el uso del modelado UML [9]. El modelado de software permite a los desarrolladores comprender todo el diseño de software, obteniendo como resultado una visión general del sistema y una herramienta de comunicación con otros desarrolladores [16].

Para empezar con el modelado de un software se necesitan los requisitos planteados por el cliente. Un estudio revelo que el 95% de los documentos de requisitos de un sistema estaban redactados en algún tipo de lenguaje natural [9]. Todos los requisitos planteados son plasmados en casos de uso, siendo la herramienta de modelado más habitualmente utilizada para representar las especificaciones del software [17]. Estos casos de uso permiten analizar más a fondo de forma privada las necesidades del cliente. Sin embargo, al realizar esto suelen surgir confusiones entre el cliente y el desarrollador, debido a que los casos de uso podrían ser modificados, detallando características técnicas del software que el cliente no logrará comprender.

El desacuerdo que existe entre el cliente y el desarrollador podría llevar a resultados poco favorables para ambos. Las descripciones de los casos de uso modificados por el desarrollador podrán ser útiles para construir uno de los diagramas más populares del modelado UML como lo es el diagrama de clases [13]. Pero, para el cliente los casos de uso no tendrán sentido lógico respecto a las condiciones previas dictadas por él.

TDDT4IoTS es una herramienta que usa un lenguaje de símbolos para escribir las descripciones de los casos de uso, permitiendo detallar datos técnicos sobre las clases, interfaces, métodos, etc. que formaran parte del sistema informático a desarrollar. Esta herramienta no cuenta con algún mecanismo que permita notificar si se está utilizando de manera correcta los símbolos respectivos. Además, solo permite generar una estructura JSON del diagrama de clases personalizada, sin tomar en cuenta la posibilidad de generar una estructura que pueda ser comprendida por otras herramientas que realizan el modelado con UML.

Sería importante proveer a los desarrolladores de software o usuarios que utilicen la herramienta TDDT4IoTS, una tecnología que permita detectar los errores cometidos al mo-

mento de usar el lenguaje de símbolos para crear los casos de uso extendidos. Además, seria interesante generar una estructura del diagrama de clases pertinente a los casos de uso y pueda ser utilizado por otras herramientas de modelado UML.

Diagnóstico.

...

Pronóstico.

...

1.1.2. Formulación del problema.

¿Es posible detectar los errores de escritura en los casos de uso extendidos, escritos en el lenguaje de símbolos utilizado en la herramienta TDDT4IoTS para generar un diagrama de clases?

1.1.3. Sistematización del problema.

1. ¿Qué formatos de archivos permitirá a otras herramientas de modelado modificar los diagramas de clases generados por la librería propuesta?
2. ¿Se puede notificar al desarrollador sobre la incorrecta escritura de los casos de uso redactados con el lenguaje de símbolos?
3. ¿Cómo determinar el nivel de efectividad del producto de este proyecto de investigación respecto a la generación del diagrama de clases?

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 los casos de uso escritos con el lenguaje de símbolos usado en la herramienta TDDT4IoTS, detectando los errores de escritura parar generar un diagrama de clases.

1.2.2. Objetivos Específicos.

1. Generar un formato de archivo JSON y XML para que otras herramientas de modelado puedan modificar los diagramas de clases generados por la librería propuesta.

2. Diseñar e implementar una manera de retroalimentar a los usuarios de TDDT4IoTS sobre la incorrecta escritura de los casos de uso redactados con el lenguaje de símbolos.

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

1.3. Justificación.

Los requisitos planteados por el cliente no siempre estarán claros desde el principio [12]. Al momento de comenzar la fase de desarrollo de un sistema informático, pueden surgir problemas que deberán ser resueltos de diferentes formas a las que fueron planteadas al inicio. Existen herramientas que permiten crear los diagramas de casos de uso transmitiendo de forma gráfica los requisitos que se deben ejecutar, pero no detallan toda la información requerida por un solo caso de uso [13].

Toda la información de un caso de uso detalla las condiciones previas, precondiciones y secuencia de eventos que deberá seguir un sistema informático [12]. Es decir, toda esa información permitirá generar otros tipos de diagramas usando el modelado UML. Sin embargo, las herramientas existentes hasta el momento permiten generar varios tipos de diagramas UML, pero es necesario para poder crearlos o modificarlos, construirlos por separado y de manera manual.

Una solución para mejorar el rendimiento de los desarrolladores es utilizar un lenguaje para la escritura de los casos de uso que permita escribir toda la información necesaria de un caso de uso y al mismo tiempo permita detallar información técnica del sistema informático. Se podrá obtener todo lo necesario para crear un diagrama de casos de uso y generar de forma automática la estructura del diagrama de clases pertinente a los datos técnicos detallados con la ayuda del lenguaje de símbolos usando por la herramienta TDDT4IoTS.

Para los usuarios de la herramienta TDDT4IoTS será muy beneficioso contar con una librería que le permita interpretar para detectar si la escritura de la información es correcta. Además,

el producto final de este trabajo permitirá modificar los casos de uso y al mismo tiempo poder hacerlo con los datos técnicos referentes al diagrama de clases, tratando de reducir el tiempo en el refinamiento de modelos del software. Finalmente se puede mencionar que no se ha encontrado aplicaciones o algún tipo de software que permita escribir los casos de uso, haciendo posible la generación de otros tipos de diagramas.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

FUNDAMENTACIÓN TEÓRICA

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 referencial

El diagrama de clases sin duda es el artefacto más importante para modelar un sistema y el punto de partida para otros diagramas [18]. La problemática de la obtención de diagramas de clases a partir de los casos de uso detallados no ha sido tratada a profundidad. Por lo tanto los investigadores se encuentran en un campo fértil de investigación.

Las soluciones que se han propuesto y que se han recuperado en este trabajo, utilizan el procesamiento del lenguaje natural (NLP por sus siglas en inglés *Natural Language Process*). Entre las soluciones encontradas podemos mencionar el trabajo de Chen y Zeng [19], en el cual presentan una aproximación sobre la obtención del diagrama de casos de uso y el diagrama de clases UML a partir de los requisitos del producto expresados en lenguaje natural. Sin embargo, en este trabajo intentan analizar el texto para determinar el conjunto de palabras significativas que pueden representar los elementos de los diagramas, por ejemplo, para una clase un sustantivo, para un método un verbo nominal para los métodos, la conexión entre dos objetos (clases) expresarían una relación. El trabajo de Chen & Zeng [19] tiene buenos resultados con pocos y bien establecidos requisitos de software, que pueden animar a los investigadores a seguir mejorando esa herramienta, como por ejemplo, identificar los diferentes tipos de relaciones entre clases, y su representación en el diagrama UML.

El trabajo presentado por Dawood Omer & Eltyeb [20] es otra de las soluciones propuestas. En [20] proponen un modelo de razonamiento que basado en casos puede facilitar el proceso de generación de diagramas de clases UML a partir de requisitos textuales. Para ello utilizan técnicas de minería de texto. Por lo tanto el trabajo es bastante abrumador: primero se deben tener una base de casos para entrenar el modelo, luego se debe entrenar el modelo. Además, aunque las diferencias sean muy pequeñas, los resultados (diagrama de clases UML) puede variar.

En la misma línea de la aplicación de procesamiento del lenguaje natural para la obtención del diagrama de clases a partir de la descripción textual de los requisitos, podemos citar el trabajo de Abdelkareem M. Alashqar [21], en el que, para lograr este objetivo proponen un algoritmo y una herramienta. El algoritmo consiste básicamente en la separación de la oración en cada palabra que la conforman para luego aplicar un análisis morfológico a cada palabra, según el tipo de palabra (sustantivo, verbos en diferentes voces o tiempos, etc.) determinar los diferentes elementos de los diagramas. En este caso, la herramienta que implementan para aplicar el algoritmo muestra las oraciones que han sido mal escritas. Alshgar en su trabajo [21] especifica que, para el algoritmo funcione correctamente, los requisitos textuales deben estar escritos con ciertas restricciones, como por ejemplo: cada oración debe estar escrita en una línea (separadas con un nueva línea), y que cada oración represente una acción a realizar con el software, entre otras restricciones.

De los trabajos que se revisado hay que recalcar que [21] y [22] se preocupan por el análisis de oraciones (acciones) pasivas negativas. Sin embargo todos los trabajos que utilizan NLP están limitados al idioma en el que están escritos los requisitos, y al uso correcto de la gramática y todos consideran restricciones en la escritura. Por lo que se presenta a Armadillo como una librería que ayuda que SymLen sea un lenguaje a utilizar para la escritura de los casos de uso detallados sin importar el idioma, y sin ninguna restricción. Con Armadillo la eficiencia de los diagramas de clases y la generación de código de software que satisface los requisitos del usuario, depende estrictamente del uso correcto de SymLen, y/o de la corrección de los posibles errores de su uso que Armadillo muestre al analista.

2.1.1. 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 [23]. 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) [24].

2.1.2. 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 [25], 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) [26].

2.1.3. 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 [27].

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. [27].

2.2. 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.

2.2.1. 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 [28].

2.2.2. 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 [29].

2.2.3. 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 [30].

2.2.4. 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 [31].

También existen lenguajes de modelado personalizados para distintas áreas, como por ejemplo en [32] 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.

2.3. Marco referencial.

CAPÍTULO III

BIBLIOGRAFÍA

[1] V. Panthi, A. Tripathi, and D. P. Mohapatra, "Software validation based on prioritization using concurrent activity diagram," *International Journal of Systems Assurance Engineering and Management*, Aug. 2022, ISSN: 09764348. DOI: 10.1007/S13198-021-01551-8.

[2] F. Chen, L. Zhang, X. Lian, and N. Niu, "Automatically recognizing the semantic elements from uml class diagram images," *Journal of Systems and Software*, vol. 193, p. 111 431, Nov. 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111431. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121222001340.

[3] R. Kulesza, M. F. D. Sousa, M. L. M. D. Araújo, C. P. D. Araújo, and A. M. Filho, "Evolution of web systems architectures: A roadmap," *Special Topics in Multimedia, IoT and Web Technologies*, pp. 3–21, Jan. 2020. DOI: 10.1007/978-3-030-35102-1_1/FIGURES/6. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-35102-1_1.

[4] M. S. Hamdi, A. Ghannem, and M. Kessentini, "Requirements traceability recovery for the purpose of software reuse: An interactive genetic algorithm approach," *Innovations in Systems and Software Engineering*, vol. 18, pp. 193–213, 1 Mar. 2022, ISSN: 16145054. DOI: 10.1007/S11334-021-00418-2.

[5] E. Guerra, A. D. O. Dias, L. G. D. Veras, A. Aguiar, J. Choma, and T. S. D. Silva, "A model to enable the reuse of metadata-based frameworks in adaptive object model architectures," *IEEE Access*, vol. 9, pp. 85 124–85 143, 2021, ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3087795.

[6] OMG, *About the unified modeling language specification version 2.4*. [Online]. Available: https://www.omg.org/spec/UML/2.4/.

[7] G. Bergström, F. Hujainah, T. Ho-Quang, *et al.*, "Evaluating the layout quality of uml class diagrams using machine learning," *Journal of Systems and Software*, p. 111 413, Oct. 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111413.

[8] Omg, "An omg ® unified modeling language ® publication omg ® unified modeling language ® (omg uml ®) omg document number: Date," 2009. [Online]. Available: https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi.

[9] M. Jahan, Z. S. H. Abad, and B. Far, "Generating sequence diagram from natural language requirements," *Proceedings of the IEEE International Conference on Requirements Engineering*, vol. 2021-September, pp. 39–48, Sep. 2021, ISSN: 23326441. DOI: 10.1109/REW53955.2021.00012.

[10] F. Losavio, A. Matteo, and I. Pacilli, "Goal-oriented process for domain analysis using quality standards," *Enlace*, vol. 6, pp. 11–28, 3 2009, ISSN: 1690-7515. [Online]. Available: http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=eshttp://ve.scielo.org/scielo.php?script=sci_abstract&pid=S1690-75152009000300002&lng=es&nrm=iso&tlng=es.

[11] C. M. Zapata and G. González, "Formal ocl specification of consistency rules between the uml class and the use case models and the interfaces model," *Revista Ingenierías Universidad de Medellín*, vol. 7, 12 Jun. 2008, ISSN: 2248-4094. [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-33242008000100010.

[12] S. Iqbal, I. Al-Azzoni, G. Allen, and H. U. Khan, "Extending uml use case diagrams to represent non-interactive functional requirements," *E-Informatica Software Engineering Journal*, vol. 14, pp. 97–115, 1 2020, ISSN: 20844840. DOI: 10.37190/E-INF200104.

[13] E. A. Abdelnabi, A. M. Maatuk, and M. Hagal, "Generating uml class diagram from natural language requirements: A survey of approaches and techniques," *2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings*, pp. 288–293, May 2021. DOI: 10.1109/MI-STA52233.2021.9464433.

[14] H. M. Abu-Dalbouh and S. A. Alateyah, "An extension to uml for the modeling of web based bus reservation system," *Journal of Computer Science*, vol. 16, pp. 825–837, 7 2020, ISSN: 15526607. DOI: 10.3844/JCSSP.2020.825.837.

[15] G. Guerrero-Ulloa, D. Carvajal-Suárez, G. Brito Casanova, A. Pachay Espinoza, M. J. Hornos, and C. Rodríguez-Domínguez, *Test-driven development tool for iot-based system*. [Online]. Available: https://aplicaciones.uteq.edu.ec/tddt4iots/.

[16] C. G. Moyano, L. Pufahl, I. Weber, and J. Mendling, "Uses of business process modeling in agile software development projects," *Information and Software Technology*, vol. 152, p. 107 028, Dec. 2022, ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2022.107028. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584922001483.

[17] Z. A. Hamza and M. Hammad, "Analyzing uml use cases to generate test sequences," *International Journal of Computing and Digital Systems*, vol. 10, pp. 125–134, 1 2021, ISSN: 2210142X. DOI: 10.12785/IJCDS/100112.

[18] L. Tan, Z. Yang, and J. Xie, "Ocl constraints automatic generation for uml class diagram," *Proceedings 2010 IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010*, pp. 392–395, 2010. DOI: 10.1109/ICSESS.2010.5552361.

[19] L. Chen and Y. Zeng, "Automatic generation of uml diagrams from product requirements described by natural language," *Proceedings of the ASME Design Engineering Technical Conference*, vol. 2, pp. 779–786, PARTS A AND B Jul. 2010. DOI: 10.1115/DETC2009-86514.

[20] O. S. D. Omer and S. Eltyeb, "Towards an automatic generation of uml class diagrams from textual requirements using case-based reasoning approach," pp. 1–5, Jun. 2022. DOI: 10.1109/ICAAID51067.2022.9799502.

[21] A. M. Alashqar, "Automatic generation of uml diagrams from scenario-based user requirements," *Jordanian Journal of Computers and Information Technology (JJCIT)*, vol. 07, 02 Jun. 2021.

[22] Shweta and R. Sanyal, "Impact of passive and negative sentences in automatic generation of static uml diagram using nlp," *Journal of Intelligent Fuzzy Systems*, vol. 39, pp. 2047–2059, 2 Jan. 2020, ISSN: 1064-1246. DOI: 10.3233/JIFS-179871.

[23] J.-W. Maessen and X. Shen, “Improving the java memory model using crf,” 2000.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.