



**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** declaro 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**

**Título:**

**"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

I	CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN	3
1.1	Problema de investigación. . . . .	4
1.1.1	Planteamiento del problema. . . . .	4
1.1.2	Formulación del problema. . . . .	5
1.1.3	Sistematización del problema. . . . .	5
1.2	Objetivos. . . . .	5
1.2.1	Objetivo General. . . . .	5
1.2.2	Objetivos Específicos. . . . .	6
1.3	Justificación. . . . .	6
II	FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN	7
2.1	Marco conceptual. . . . .	8
2.1.1	JSON. . . . .	8
2.1.2	XML. . . . .	8
2.1.3	Compiladores. . . . .	8
2.1.4	Lenguajes de modelado. . . . .	9
2.2	Marco referencial. . . . .	9
2.2.1	Lenguaje de restricciones de objetos para la generación de código a partir de modelos de actividad . . . . .	9
2.2.2	Desarrollo de módulos de software generativo para el diseño orientado al do- minio con un lenguaje específico de dominio basado en anotaciones . . . . .	10
2.2.3	Modelado para la localización de características en modelos de software: tanto la generación de código y los modelos interpretados. . . . .	10
2.3	Marco referencial. . . . .	10
III	BIBLIOGRAFÍA	11

# Introducción

Los sistemas informáticos se están volviendo cada vez más indispensable en todos los aspectos de la vida cotidiana. Por ende, los desarrolladores de software buscan disminuir el tiempo que toma desarrollar diferentes tipos de aplicaciones [1]. Para la construcción de una aplicación informática se necesitan aplicar varios conceptos técnicos para su correcto funcionamiento [2]. Al igual que el plano de un edificio se detallando las principales características físicas del terreno, columnas, puertas, ventanas, etc. Se necesita documentar con precisión el comportamiento que deben tener todos los artefactos de un software y de ser necesario que sean reutilizables [3].

*Unified Modeling Language* (UML) con su traducción al español Lenguaje Unificado de Modelado, se utiliza específicamente en la industria del software para especificar, visualizar, construir y documentar los artefactos de un sistema de software [4]. UML se encuentra definido oficialmente por el *Object Management Group* (OMG) con su traducción al español Grupo de Administración de Objetos [5]. Algunos investigadores han propuesto que la mejor forma de generar modelos UML es a partir de modelos estáticos como los diagramas de casos de uso y diagramas de clases [6].

Los diagramas de casos de uso representan al sistema mediante los usuarios (actores) y sus requisitos (casos de uso). La información de cada caso de uso detalla las condiciones previas, precondiciones y secuencia de eventos. 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 [7]. Existe la forma de comprimir globalmente los requisitos de un sistema, una de las herramientas más populares para poder realizarlo son los diagramas de clases [8].

En UML los diagramas de clases 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 [9]. Se lo desarrolla durante la fase de elaboración y se perfecciona posteriormente en la fase de construcción [5], representando el 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, atributos, operaciones y relaciones entre clases [9].

En la herramienta TDDT4IoTS [10] es posible escribir las descripciones de los casos de uso

usando un lenguaje de símbolos, pero es necesario verificar si los casos de uso están escritos de manera correcta. Este trabajo tiene como idea, desarrollar una librería escrita en el lenguaje de programación JavaScript denominada **Armadillo.js**. Armadillo permitirá interpretar las descripciones de los casos de uso extendidos, escritos en un lenguaje de símbolos usado por la herramienta TDDT4IoTS. Finalmente se obtiene como resultado la estructura de un diagrama de clases pertinente a la información obtenida por los casos de uso de forma automática.

**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 [6]. 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 [11].

Para empezar con el modelado de un software se necesitan los requisitos planteados por el cliente. Un estudio reveló que el 95% de los documentos de requisitos de un sistema estaban redactados en algún tipo de lenguaje natural [6]. Todos los requisitos planteados son plasmados en casos de uso, siendo la herramienta de modelado más habitualmente utilizada para representar las especificaciones del software [12]. 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 [8]. 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, sería 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 para 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 [7]. 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 [8].

Toda la información de un caso de uso detalla las condiciones previas, precondiciones y secuencia de eventos que deberá seguir un sistema informático [7]. 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**

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 [13].

### **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 [14].

### **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 [15].

#### **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 [16].

También existen lenguajes de modelado personalizados para distintas áreas, como por ejemplo en [17] 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 [18]. 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) [19].

### **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 [20], 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) [21].

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

Evalutando 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 [22].

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. [22].

## **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. 111431, Nov. 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111431. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0164121222001340>.
- [3] 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.
- [4] 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. 111413, Oct. 2022, ISSN: 01641212. DOI: 10.1016/J.JSS.2022.111413.
- [5] 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>.
- [6] 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.



- [7] 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.
- [8] 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.
- [9] 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.
- [10] G. G. Ulloa, D. C. Suárez, G. B. Casanova, P. E. A., J. Hornos, and R. D. C, *Test-driven development tool for iot-based system*. [Online]. Available: <https://aplicaciones.uteq.edu.ec/tddt4iots/>.
- [11] 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. 107028, Dec. 2022, ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2022.107028. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0950584922001483>.
- [12] 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.
- [13] 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.
- [14] 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.

- [15] 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.
- [16] 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.
- [17] 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.
- [18] J.-W. Maessen and X. Shen, “Improving the java memory model using crf,” 2000.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] 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.