



UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

Facultad de ciencias de la ingeniería



ESTUDIANTES:

Anderson Jaime

John Vera

William Vera

CARRERA:

Ingeniería de software

CURSO:

7to "A"

ASIGNATURA:

Aplicaciones distribuidas

DOCENTE:

Ing. Gleiston Guerrero

TEMA:

Desarrollo de Aplicaciones en Ambientes Distribuidos

GitHub (Practica):

<https://github.com/JohnVeraXD/PRACTICA-PETRI-App-Distribuidas-.git>

Índice

1.	Introducción	9
2.	Modelamiento de Aplicaciones Distribuidas	10
2.1.	Diagramas UML	10
2.1.1.	Diagrama de casos de uso	11
2.1.2.	Componentes de casos de uso	11
2.1.3.	Actores en el caso de uso	12
2.1.4.	Relaciones en casos de uso	12
2.1.5.	Casos de uso	13
2.1.6.	Notaciones	13
2.1.7.	Generalización	14
2.1.8.	Extensión	14
2.1.9.	Inclusión	14
2.2.	Diagrama de secuencia.....	15
2.2.1.	Componentes de los diagramas de secuencia.	15
2.2.2.	Actor en diagramas de secuencia	16
2.2.3.	Clase.....	16
2.2.4.	Mensajes.....	17
2.2.5.	Activación.....	17
2.3.	Diagrama de despliegue	18
2.3.1.	Artefactos en los diagramas de despliegue	18
2.3.2.	Manifestación de un artefacto.....	18
2.3.3.	Nodos.....	19
2.3.4.	Entorno de ejecución.....	19
2.3.5.	Dispositivos.....	19
2.3.6.	Rutas de comunicación.....	20
2.3.7.	Despliegue.....	20

2.3.8.	Especificaciones de despliegue	21
2.3.9.	Beneficios de un diagrama de despliegue.....	21
2.3.10.	Desafíos de un diagrama de despliegue	22
2.4.	Diagrama de componentes	23
2.4.1.	Algunos usos del ama de componentes.....	23
2.4.2.	Componentes en diagramas de componentes	24
2.4.3.	Paquetes	24
2.4.4.	Interfaz.....	25
2.4.5.	Relación de dependencia.....	25
2.5.	Diagrama de clases	26
2.5.1.	Componentes	26
2.5.2.	Clase	26
2.5.3.	Atributos	27
2.5.4.	Identificadores	28
2.5.5.	Métodos.....	28
2.5.6.	Relaciones	29
2.5.7.	Multiplicidad	29
2.5.8.	Interfaces	30
2.5.9.	Asociación	30
2.5.10.	Agregación.....	32
2.5.11.	Composición	33
2.5.12.	Generalización o Herencia.....	33
2.5.13.	Notas	34
2.6.	Diagramas BPMN	35
2.6.1.	Conceptos básicos de BPMN	35
2.6.2.	Usos comunes de BPMN	35
2.6.3.	Modelamiento con BPMN en sistemas distribuidos.....	36

2.6.4.	Elementos de BPMN.....	36
2.6.5.	Notación y símbolos utilizados en BPMN	37
2.6.6.	Eventos.....	38
2.6.7.	Eventos de inicio.....	38
2.6.8.	Eventos intermedios	39
2.6.9.	Eventos finales	39
2.6.10.	Actividades	39
2.6.11.	Puertas de enlace.	40
2.6.12.	Flujo de secuencia	40
2.6.13.	Flujo de mensajes.....	41
2.6.14.	Asociación	41
2.6.15.	Pool	41
2.6.16.	Carril	42
2.6.17.	Objeto de datos	42
2.6.18.	Mensaje	43
2.6.19.	Grupo (un cuadro alrededor de un grupo de objetos dentro la misma categoría).....	43
2.6.20.	Símbolos de eventos de BPMN	44
2.6.21.	Ejemplo del diagrama BPMN	48
2.7.	Redes de Petri	48
2.7.1.	Importancia de la modelación con redes de Petri.....	49
2.7.2.	Conceptos de redes de Petri.....	49
2.7.3.	Elementos de una red de Petri.....	49
2.7.4.	Propiedades y características de las redes de Petri.....	50
2.7.5.	Propiedades.....	51
2.7.6.	Características	51
2.7.7.	Modelamiento de sistemas distribuidos.....	52

2.7.8.	Modelamiento con redes de Petri en sistemas distribuidos ...	52
2.7.9.	Evaluación y validación de sistemas distribuidos representados mediante con redes de Petri.....	53
2.7.10.	Ejemplo de Petri.....	53
3.	Desarrollo de Sistemas basados en IoT	54
3.1.	Metodología de Desarrollo	55
3.1.1.	Metodologías ágiles	55
3.1.2.	Metodologías tradicionales	57
3.1.3.	Metodologías híbridas	58
3.1.4.	Consideraciones	60
3.2.	Alternativas de Solución.....	61
3.2.1.	Plataforma de hardware.....	61
3.2.2.	Servicios en la nube para IoT	63
4.	Desarrollo de la práctica	66
4.1.1.	Problema planteado:.....	66
4.1.2.	Solución.....	66
5.	Conclusiones	69
6.	Bibliografía	70

Ilustración 1 Extraído de: UML Diagrams in Teaching Software Engineering Classes. A Case Study In Computer Science Class [9]	12
Ilustración 2 Ejemplo de relaciones de casos de uso.....	12
Ilustración 3 Representación de casos de uso.....	13
Ilustración 4 Extraído de: UML Templates Distilled	13
Ilustración 5 Extraído de: Improving the Security of UML Sequence Diagram Using Genetic Algorithm [13]	14
Ilustración 6 Representación de clases en diagramas de secuencia. ...	16
Ilustración 7 Tipos de mensajes en los diagramas de secuencia.	17
Ilustración 8 Artefacto con atributos, imagen basada en [21].	18
Ilustración 9 Manifestación de un componente como un artefacto, imagen basada en [21].	18
Ilustración 10 Representación cúbica y ejemplos de representaciones de iconos. [23].	19
Ilustración 11 Rutas de comunicación entre nodos, imagen basada en [22].	20
Ilustración 12 Una representación visual de la ubicación de implementación de los artefactos, imagen basa en [25]	20
Ilustración 13 Una especificación de implementación para un artefacto implementado, Imagen basada en [26].	21
Ilustración 14. Diagrama de componentes.....	23
Ilustración 15. Notación de Componente.....	24
Ilustración 16. Ejemplo de componentes.	24
Ilustración 17. Utilización de interfaz.	25
Ilustración 18 Notación de una relación de dependencia.	25
Ilustración 19. Estructura de la clase.	26
Ilustración 20 Atributos de una clase.	27
Ilustración 21. Forma correcta de los Identificadores.....	28
Ilustración 22. Ejemplo de clase con métodos.	29
Ilustración 23 Cardinalidad	29
Ilustración 24. Implementación de la interfaz.	30
Ilustración 25. Asociación en el diagrama de clases [44].	31
Ilustración 26. Agregación en el diagrama de clases [43].	32
Ilustración 27. Composición en el diagrama de clases.	33

Ilustración 28. Herencia en el diagrama de clases.....	34
Ilustración 29. Eventos [79].	38
Ilustración 30. Evento inicio [81].	38
Ilustración 31. Evento intermedio [82].....	39
Ilustración 32. Evento final [83].....	39
Ilustración 33. Actividades [84].	40
Ilustración 34. Puertas de enlace [85].	40
Ilustración 35. Flujo de secuencia [86].	40
Ilustración 36. Flujo de mensajes [87].....	41
Ilustración 37. Asociación [88].	41
Ilustración 38. Pool [89].	42
Ilustración 39. Carril [90].....	42
Ilustración 40. Objeto de datos [91].	43
Ilustración 41. Mensaje [75].	43
Ilustración 42. Grupo [76].	44
Ilustración 43. Símbolo de mensaje [77].	44
Ilustración 44. Símbolo de temporizador [78].	45
Ilustración 45. Símbolo de escalación [79].	45
Ilustración 46. Símbolo condicional [80].....	45
Ilustración 47. Símbolo de enlace [81].....	46
Ilustración 48. Símbolo de error [82].	46
Ilustración 49. Símbolo de cancelación [83].	46
Ilustración 50. Símbolo de compensación [84].....	47
Ilustración 51. Símbolo de señal [85].....	47
Ilustración 52. Símbolo múltiple [86].	47
Ilustración 53. Símbolo de paralelas múltiples[87].	47
Ilustración 54. Símbolo de finalización[88].	48
Ilustración 55. Ejemplo de diagrama BPNM [90].....	48
Ilustración 56. Ejemplo de Petri [113].	54
Ilustración 57. Metodología Scrum.	55
Ilustración 58. Metodología Kanban.	56
Ilustración 59. Modelo en cascada.	58
Ilustración 60. Metodología DevOps.....	59
Ilustración 61. Arduino UNO R3.....	62

Ilustración 62, Raspberry Pi4 2GB.....	63
Ilustración 63. Plataformas IoT en la nube.....	63
Ilustración 64.Diagrama Petri 1.....	67
Ilustración 65. Diagrama Petri 2.....	67
Ilustración 66. Diagrama Petri 3.....	68

1. Introducción

En el desarrollo de las aplicaciones distribuidas, el diseño y la creación de software que operan en redes interconectadas de computadoras es una tarea cada vez más relevante. Este informe se centra en dos áreas de investigación cruciales en este contexto: el modelamiento de aplicaciones distribuidas y el desarrollo de sistemas basados en IoT (Internet de las Cosas).

Se explica el proceso de modelamiento de aplicaciones distribuidas. Esto implica el uso de diferentes técnicas, como el empleo de diagramas UML y BPMN, así como la aplicación de Redes de Petri para representar sistemas distribuidos. Estas herramientas nos permiten comprender cómo se estructuran y operan estos sistemas de manera efectiva. Incluye el desarrollo de sistemas IoT.

En la sección final, llevaremos a cabo una práctica utilizando un ejemplo con Redes de Petri. Esto nos permitirá aplicar los conceptos aprendidos y profundizar en su aplicación práctica en sistemas distribuidos.

2. Modelamiento de Aplicaciones Distribuidas

El modelamiento de aplicaciones distribuidas es esencial para el desarrollo de sistemas modernos, están compuestas por varios componentes que se ejecutan en diferentes nodos de una red y colaboran entre sí para cumplir un objetivo común. La comunicación entre estos componentes es fundamental y puede lograrse mediante diferentes mecanismos. La consistencia de los datos y la gestión de la concurrencia son desafíos importantes en entornos distribuidos, se utilizan técnicas como la replicación de datos y las transacciones distribuidas para abordar estos problemas, deben ser escalables y tolerantes a fallos, lo que implica diseñar sistemas que puedan manejar un mayor volumen de carga y continuar operando incluso si algunos componentes fallan.

2.1. Diagramas UML

UML, o Unified Modeling Language, es esencial en el proceso de diseño y desarrollo de software, para representar la arquitectura de un sistema distribuido, UML proporciona el diagrama de despliegue, que muestra la distribución física de los componentes del sistema en los nodos de red y las relaciones entre ellos para comprender la distribución de componentes y su comunicación con el entorno. Modela las interacciones entre los componentes distribuidos mediante los diagramas de secuencia y los diagramas de colaboración, mostrando los mensajes intercambiados entre los componentes y el flujo de control durante la ejecución del sistema, lo que permite comprender cómo se realizan las tareas y su coordinación de acciones en el entorno distribuido. Dentro de los servicios web, se utiliza para modelar su interacción en entornos distribuidos, incluye la descripción de los servicios ofrecidos, los protocolos de comunicación utilizados (como SOAP y REST), y operaciones invocadas, facilita la especificación de la arquitectura de servicios distribuidos y su integración en aplicaciones amplias [3].

UML establece una notación y semántica estándar, no prescribe un proceso o método estándar para el desarrollo de sistemas. Existen varias metodologías populares, como Catalysis, Objectory, Shlaer/Mellor, Fusion, OMT y Booch, cada una con sus propias técnicas y enfoques. Además, muchas organizaciones han desarrollado sus propias metodologías internas, combinando diferentes diagramas y técnicas de modelado [4].

Sirve como una base sólida para la implementación del sistema, proporcionan una representación clara y detallada de los requisitos y la arquitectura del software, lo que facilita la traducción de estos modelos en código implementable. Esto mejora la coherencia y la comprensión entre los miembros del equipo durante la implementación del sistema, es una herramienta invaluable en el proceso de desarrollo de software, ya que proporciona una manera estandarizada y visual de comunicar, documentar, analizar, diseñar y finalmente implementar sistemas de software complejos [5].

2.1.1. Diagrama de casos de uso

El modelado de casos de uso es una técnica efectiva y simple para capturar los requisitos del sistema desde la perspectiva del usuario, modela cómo funciona un sistema o negocio, o cómo los usuarios desean que funcione. Aunque no es una aproximación orientada a objetos, es una forma de modelar procesos que sirve como punto de partida para el análisis orientado a objetos. Durante el análisis de negocio del sistema, se pueden desarrollar modelos de casos de uso y organizarlos en paquetes que representen los diferentes dominios del negocio del sistema [6].

El objetivo es construir un diagrama de caso de uso para cada tipo de escenario diferente en el sistema, mostrando una secuencia única de interacciones entre actores y el sistema. Las secuencias alternativas se modelan en casos de uso separados y se relacionan con el caso de uso original mediante la relación "Extiende", para evitar la redundancia, partes del comportamiento pueden ser modeladas en casos de uso separados relacionados mediante la relación "Usa" [7].

2.1.2. Componentes de casos de uso

Un modelo de casos de uso consta de actores y casos de uso. Los actores representan usuarios y otros sistemas que interactúan con el sistema, mientras que los casos de uso representan el comportamiento del sistema en respuesta a un estímulo del actor. Cada caso de uso se documenta con una descripción del escenario y puede tener propiedades como condiciones pre- y post-escenario [8].

2.1.3. Actores en el caso de uso

Son entidades externas que interactúan con el sistema, como usuarios, otros sistemas informáticos o dispositivos físicos. Comunican sus necesidades al sistema mediante mensajes. Se representan en los diagramas de casos de uso con el icono estándar de "stick man" o "monigote" y su nombre. Existen tres tipos principales de actores: principales, de apoyo y pasivos. Los principales tienen objetivos de usuario que guían los casos de uso, los de apoyo brindan servicios al sistema, y los pasivos están interesados en el comportamiento del caso de uso, pero no tienen un rol activo [9].

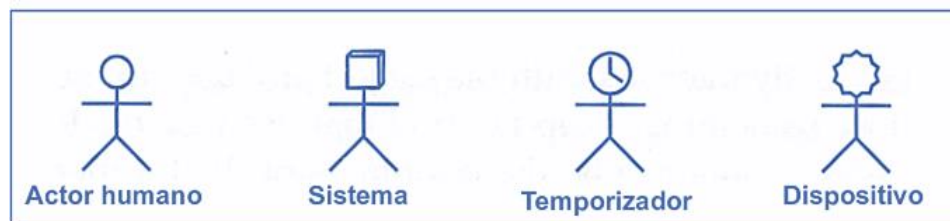


Ilustración 1 Extraído de: UML Diagrams in Teaching Software Engineering Classes. A Case Study In Computer Science Class [9]

2.1.4. Relaciones en casos de uso

Las relaciones sirven para comprender cómo interactúan diferentes entidades en el sistema, son siempre binarias y pueden establecerse asociaciones entre actores, casos de uso, subsistemas, componentes y clases. Una relación común entre actores es la de generalización, donde un actor general describe un comportamiento amplio, actores especializados heredan y extienden ese comportamiento de alguna manera [9].

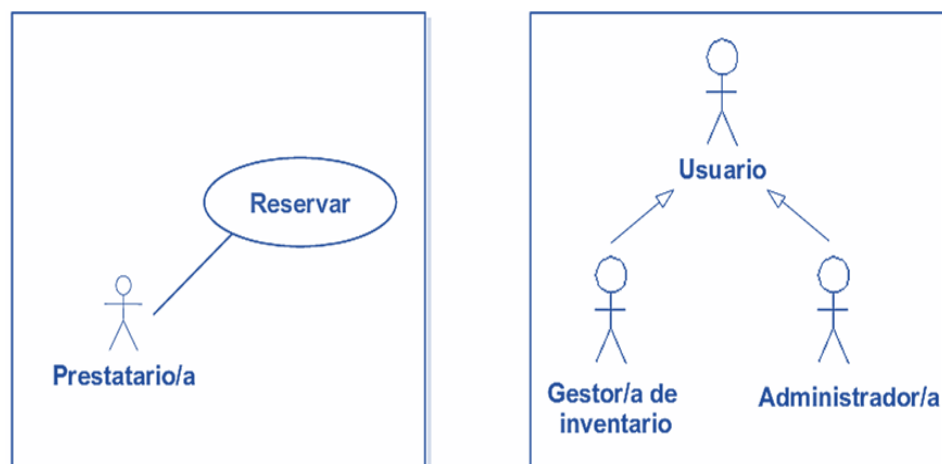


Ilustración 2 Ejemplo de relaciones de casos de uso

2.1.5. Casos de uso

Representan conjuntos de acciones realizadas por el sistema que conducen a un resultado observable. Especifican comportamientos que el sistema puede llevar a cabo en colaboración con actores, sin detallar su estructura interna. Pueden incluir variaciones, manejo de errores y excepciones. Cada instancia de un caso de uso es un escenario específico que muestra un uso particular del sistema. Se inician por un actor y proporcionan valores a los actores involucrados, se representan con una elipse que contiene su nombre, y opcionalmente estereotipos y propiedades [10].



Ilustración 3 Representación de casos de uso

2.1.6. Notaciones

Relación	Descripción	Notación
Asociación	Línea de comunicación entre un actor y un caso de uso en el que participa.	—
Generalización	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades al caso de uso base.	—>
Inclusión	Inserción de comportamiento adicional en caso de uso base que describe explícitamente la inserción.	- - - «include»>
Extensión	Inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él.	- - - «extend»>
Realización	Establece una relación entre el caso de uso y los diagramas que describen la funcionalidad del caso de uso.	- - ->

Ilustración 4 Extraído de: UML Templates Distilled

2.1.7. Generalización

Establece una conexión entre un caso de uso más específico (el "hijo") y uno más general (el "padre"). El caso de uso hijo hereda las relaciones y comportamientos del padre, pudiendo agregar atributos y operaciones propios. Además, puede añadir comportamientos adicionales a la secuencia del padre en puntos arbitrarios. Aunque puede modificar operaciones y secuencias heredadas, debe hacerlo de manera que se mantenga la intención del padre. El caso de uso padre también puede ser abstracto, lo que significa que no se instancia directamente y sirve como un concepto genérico para sus subcasos de uso [11].

2.1.8. Extensión

Permite que un caso base incorpore comportamiento adicional definido en otro caso de uso extensor. Esto enriquece el comportamiento original del caso base en puntos específicos, bajo ciertas condiciones. El extensor puede afectar a múltiples casos base y, a su vez, ser extendido por otros casos de uso. Los puntos de extensión indican dónde se integran los fragmentos de comportamiento adicional y se representan mediante texto dentro del caso de uso [12].

2.1.9. Inclusión

La relación de inclusión en los casos de uso indica que el comportamiento de un caso de uso (incluido) se inserta en el comportamiento de otro caso de uso (base o inclusor) en una ubicación específica. Esta inserción no está condicionada y tiene como objetivo reutilizar partes de comportamiento compartidas por varios casos de uso. Un caso de uso incluido puede ser insertado en varios casos de uso base y puede, a su vez, incluir otros casos de uso. Esta relación se representa mediante un símbolo de dependencia con el estereotipo "include" [13].

2.2. Diagrama de secuencia

Los diagramas de secuencia ilustran la interacción entre objetos mediante la secuencia de mensajes que intercambian. Tienen dos formas de uso: la instancia, que describe un escenario específico, y la genérica, que abarca todas las posibles alternativas en un escenario, permitiendo ramificaciones, condiciones y bucles. Se presentan dentro de un marco con el prefijo "sd" seguido del nombre del diagrama en la esquina superior izquierda. Visualmente, representan una interacción bidimensional, con la dimensión vertical como el eje de tiempo y la horizontal mostrando las líneas de vida de los objetos involucrados. Cada línea de vida indica la participación de un objeto en la interacción y su existencia, representada por un rectángulo con una línea discontinua debajo. Una cruz al final de la línea de vida indica la destrucción del objeto [14].

2.2.1. Componentes de los diagramas de secuencia.

Muestran la interacción entre objetos en un sistema a lo largo del tiempo, centrándose en la secuencia de mensajes que envían y reciben. Los elementos principales de un diagrama de secuencia incluyen actores, que son entidades externas que interactúan con el sistema; clases, que representan los objetos que participan en la interacción; mensajes, que indican la comunicación entre objetos; líneas de vida, que muestran la existencia de los objetos y su participación en la interacción; y la activación, que es representada por barras verticales para mostrar el período durante el cual un objeto está activo y ejecutando una acción, estos elementos se combinan para proporcionar una representación visual y secuencial de cómo interactúan los objetos en un sistema, ya sea describiendo un escenario específico o mostrando todas las posibles alternativas en un escenario [15].

2.2.2. Actor en diagramas de secuencia

Representa una entidad externa que interactúa con el sistema. Los actores pueden ser humanos, otros sistemas informáticos o incluso hardware. En el diagrama de secuencia, los actores se representan como cajas rectangulares con el nombre del actor dentro de ellas. Estas cajas suelen ubicarse en la parte superior del diagrama para indicar su origen externo al sistema modelado. Los actores envían mensajes a las clases u objetos del sistema y reciben respuestas de ellos. Su inclusión en el diagrama de secuencia ayuda a visualizar y comprender las interacciones entre el sistema y su entorno externo, proporcionando así un contexto claro para el comportamiento del sistema desde una perspectiva de usuario o sistema externo [16].

2.2.3. Clase

Objeto o entidad en el sistema que participa en la interacción, se utilizan para modelar objetos que realizan acciones o procesos dentro del sistema, muestra como una caja rectangular con el nombre de la clase en la parte superior de la caja, puede tener atributos y métodos que definen su comportamiento y estado. Las clases pueden enviar y recibir mensajes entre sí y con los actores, lo que refleja la comunicación y la colaboración entre los distintos componentes del sistema, permite visualizar cómo interactúan entre sí para lograr un determinado comportamiento [17].

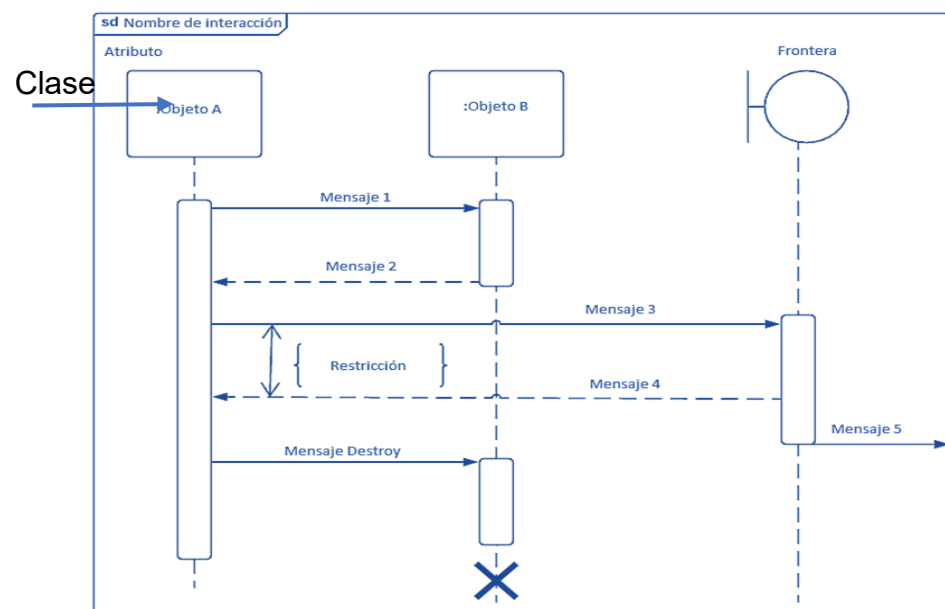


Ilustración 6 Representación de clases en diagramas de secuencia.

2.2.4. Mensajes

Flechas que representan comunicaciones entre objetos. Las medias flechas representan mensajes asincrónicos, los mensajes asincrónicos son enviados desde un objeto que no va a esperar una respuesta del receptor para continuar con sus tareas. representación de una interacción comunicativa entre objetos en el sistema modelado, implica transferencia de información necesaria para ejecutar una acción específica, cuando un objeto recibe un mensaje, se desencadena una actividad o una instancia de ejecución en respuesta a este evento [18].



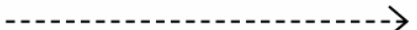


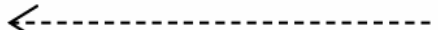
Mensaje síncrono	
Mensaje asíncrono	
Mensaje de creación	
Mensaje encontrado	
Mensaje perdido	
Mensaje de retorno	

Ilustración 7 Tipos de mensajes en los diagramas de secuencia.

2.2.5. Activación

Los elementos de activación en un diagrama de secuencia reflejan la ocurrencia de ejecución de los objetos, mostrando el foco del control en un momento dado. Cuando un objeto está activado, significa que está ejecutando su propio código o esperando el retorno de otro objeto tras haberle enviado un mensaje. Aunque su representación es opcional, se utiliza un rectángulo estrecho sobre la línea de vida del objeto para denotar esta activación. Estos cuadros de activación también indican el tiempo que un objeto necesita para completar una tarea, proporcionando así una perspectiva temporal del proceso en el diagrama de secuencia [19].

2.3. Diagrama de despliegue

Modela el mapeo físico de las piezas de software de un sistema al hardware que va a ejecutarlo, los elementos de software normalmente se manifiestan utilizando artefactos y se asignan al entorno de hardware o software que los alojará, llamados nodos. Debido a que muchos nodos pueden estar asociados con la implementación de un sistema, la comunicación entre nodos se puede modelar utilizando rutas de comunicación, emergen como una herramienta de modelado que permite a los ingenieros de software y los arquitectos de sistemas crear una representación visual de la topología física [20].

2.3.1. Artefactos en los diagramas de despliegue

Los artefactos son elementos físicos que representan información relacionada con el desarrollo de software, como DLLs, manuales de usuario o ejecutables. Aunque comúnmente se usan para representar versiones compiladas de componentes, pueden representar cualquier elemento empaquetable. Se muestran mediante un rectángulo clasificador con un papel doblado en la esquina superior derecha [21].



Ilustración 8 Artefacto con atributos, imagen basada en [21].

2.3.2. Manifestación de un artefacto

Un artefacto es una representación de otro elemento UML. Por ejemplo, logging.jar puede ser una manifestación del componente LoggingSubsystem. La relación de manifestación se representa con una línea discontinua desde el artefacto al elemento que representa, y se etiqueta la línea con la palabra clave «manifest» [22].

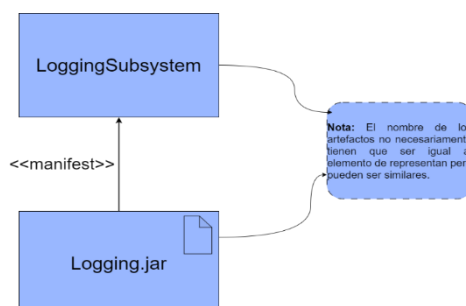


Ilustración 9 Manifestación de un componente como un artefacto, imagen basada en [21].

2.3.3. Nodos

Un nodo es una entidad física capaz de ejecutar artefactos. Los nodos pueden variar en tamaño desde un simple dispositivo integrado hasta una granja de servidores. Son una pieza clave en cualquier diagrama de implementación ya que muestran dónde se ejecuta un determinado fragmento de código y cómo se comunican entre sí las diferentes partes del sistema a nivel de ejecución [22].



Ilustración 10 Representación cúbica y ejemplos de representaciones de iconos. [23].

2.3.4. Entorno de ejecución

Es un nodo especializado que representa una configuración de software para albergar tipos específicos de artefactos, espera un entorno de ejecución proporcione servicios a los artefactos hospedados en interfaces mutuamente acordadas, una aplicación Java 2 Enterprise Edition (J2EE) espera ejecutarse en un entorno de software llamado Servidor de Aplicaciones, enumera varios servicios que deben ser proporcionados por un servidor de aplicaciones, como conectividad a bases de datos, contratos de ciclo de vida y localización de recursos [23].

2.3.5. Dispositivos

Un dispositivo es una especialización de un nodo que representa una máquina física con capacidad de cómputo. No existen restricciones de tamaño implícitas en un dispositivo, este puede representar desde un controlador integrado hasta el hardware donde se instala. Un dispositivo como un nodo se representa con el estereotipo <<device>>

2.3.6. Rutas de comunicación

Se representan como una comunicación genérica entre nodos. Por ejemplo, un enrutador puede pasar solicitudes HTTP a un servidor web que utiliza una conexión con socket patentado a una base de datos. Mostrando una ruta de comunicación en UML como una línea continua dibuja de un nodo a otro, normalmente no se muestra ninguna direccionalidad en la línea por qué se entiende que es bidireccional [1].

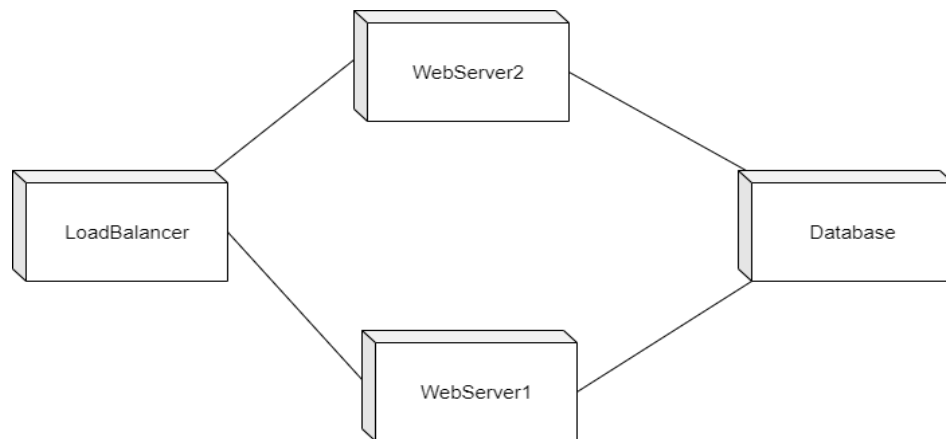


Ilustración 11 Rutas de comunicación entre nodos, imagen basada en [22].

2.3.7. Despliegue

Es una herramienta para definir la arquitectura de ejecución de sistemas y asignar artefactos de software a elementos del sistema. Proporciona un modelo simplificado de despliegue adecuado para la mayoría de las aplicaciones modernas. Cuando se necesitan modelos más detallados, el paquete puede ser ampliado mediante perfiles o metamodelos para representar entornos de hardware y/o softwares específicos [25].

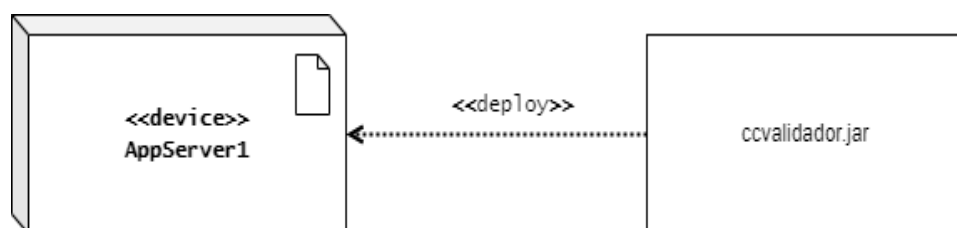


Ilustración 12 Una representación visual de la ubicación de implementación de los artefactos, imagen basa en [25]

2.3.8. Especificaciones de despliegue

Las especificaciones de despliegue detallan cómo se implementará un artefacto en un destino específico, pueden ser representadas como elementos en las relaciones de implementación, consisten en una serie de propiedades que describen los requerimientos particulares del artefacto, como transacciones de base de datos o detalles de conexión con un servidor. Por ejemplo, un artefacto podría necesitar ciertas configuraciones específicas para funcionar correctamente en un entorno determinado, se modelan como artefactos de especificación de implementación para facilitar su gestión y comprensión [26].

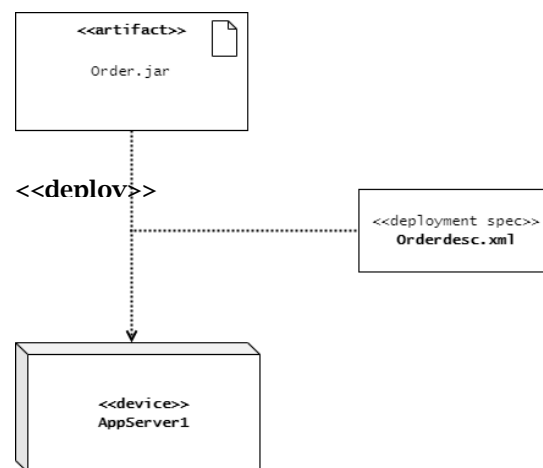


Ilustración 13 Una especificación de implementación para un artefacto implementado, Imagen basada en [26].

2.3.9. Beneficios de un diagrama de despliegue

Los beneficios de utilizar un diagrama de despliegue en el desarrollo de software son múltiples y significativos. Proporcionan una vista general de la arquitectura física de un sistema: muestra cómo se conectan los componentes de hardware y cómo se distribuye el software entre ellos, esto ayuda a entender mejor el sistema en su conjunto [27].

Los diagramas de despliegue muestran las conexiones y relaciones entre los distintos nodos de red como servidores, dispositivos y conexiones, identificación de Relaciones entre Componentes Permite identificar fácilmente las relaciones y conexiones entre los artefactos de software y los nodos de ejecución, lo que facilita la comprensión de cómo funcionan en conjunto [28].

Análisis de Rendimiento y Escalabilidad. facilita la evaluación del rendimiento y la escalabilidad del sistema al mostrar cómo se distribuyen los componentes en diferentes nodos de la red. Esto permite identificar posibles cuellos de botella y diseñar soluciones para mejorar el rendimiento [29].

Facilita la Comunicación entre Equipos, sirve como una herramienta de comunicación efectiva entre equipos de desarrollo, operaciones y gestión al proporcionar una representación visual común del sistema y su infraestructura subyacente [30].

2.3.10. Desafíos de un diagrama de despliegue

Escalabilidad, deben ilustrar cómo la arquitectura puede adaptarse para gestionar incrementos en la carga de trabajo y tráfico. Esto requiere contemplar la incorporación de servidores y recursos adicionales para asegurar un escalado efectivo. Seguridad, deben mostrar dónde se implementarán los controles de seguridad como firewalls, sistemas de detección/prevenición de intrusiones, entre otros. La seguridad debe estar integrada en toda la arquitectura. Mantenibilidad, deben permitir identificar fácilmente los componentes y relaciones entre ellos para simplificar el mantenimiento y resolución de problemas [31].

2.4. Diagrama de componentes

El diagrama de componentes es uno de los principales diagramas UML. Está clasificado como diagrama de estructura y, como tal, representa de forma estática el sistema de información. Habitualmente se utiliza después de haber creado el diagrama de clases, pues necesita información de este diagrama como pueden ser las propias clases, proporciona una vista de alto nivel de los componentes dentro de un sistema. Los componentes pueden ser un componente de software, como una base de datos o una interfaz de usuario; o un componente de hardware como un círculo, microchip, o una unidad de negocio como un proveedor, etc. [32].

2.4.1. Algunos usos del ama de componentes

Se utilizan en desarrollo basado en componentes para describir sistemas con arquitectura orientada a servicios. Mostrar la estructura del propio código. Se puede utilizar para centrarse en la relación entre los componentes mientras se ocultan los detalles de las especificaciones. Ayudar a comunicar y explicar las funciones del sistema que se está construyendo a los interesados o stakeholders [33].

Para su construcción se debe plantear en primer lugar identificar los componentes que utilizará el sistema de información, así como las distintas interfaces. Una forma típica y común para una primera aproximación en sistemas sencillos es utilizar un componente central al que los demás componentes se unen y utiliza como componente gestor del sistema [34].

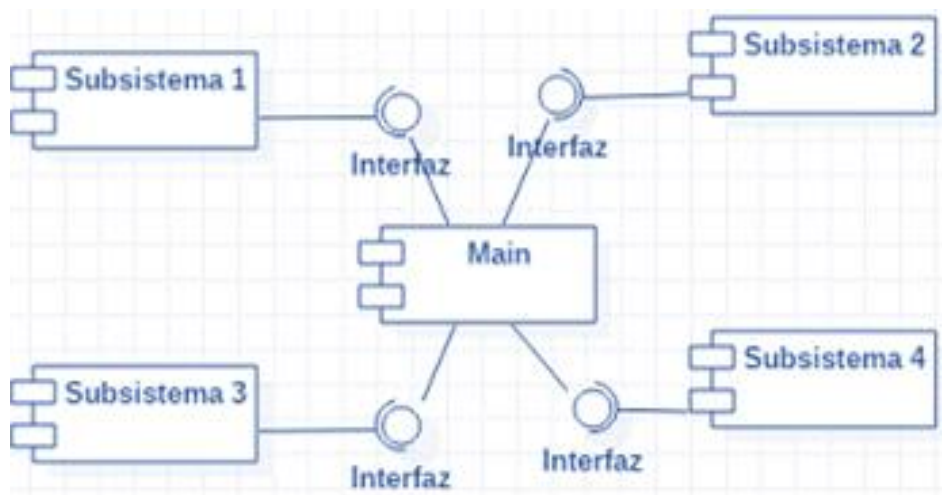


Ilustración 14. Diagrama de componentes.

2.4.2. Componentes en diagramas de componentes

El diagrama de componentes está formado por tres elementos. Componente, Interfaz y Relación de dependencia. Un componente es un bloque de unidades lógicas del sistema, una abstracción ligeramente más alta que las clases. Se representa como un rectángulo más pequeño en la esquina superior derecha con pestañas o la palabra escrita encima del nombre del componente para ayudar a distinguirlo de una clase [35].

Un componente puede representar dos tipos de elementos: componentes lógicos (como por ejemplo componentes de negocio o proceso) o componentes físicos (como componentes .NET, EJB.). Por ejemplo, en una aplicación desarrollada en java habrá, con total seguridad, varios componentes “.java”, que son componentes lógicos del sistema. Es representado a través de un rectángulo que tiene, a su vez, dos rectángulos a la izquierda, tal y como se muestra en la siguiente imagen [36]



Ilustración 15. Notación de Componente.

2.4.3. Paquetes

También es posibles utilizar el diagrama de paquetes para hacer un conjunto de varios módulos. Con esto se consigue representar la unión de esos módulos para un fin concreto, entre algunos de los ejemplos de componentes podrían ser los siguientes: Gestión de E/S, Animal, Personal, Gestión de incidencias, Gestos de workflow, son conceptos muy amplios y que pueden ser más o menos específicos dependiendo de la profundidad que se puede dar al diagrama [37].



Ilustración 16. Ejemplo de componentes.

2.4.4. Interfaz

La interfaz esta siempre asociada a un componente y se utiliza para representar la zona del módulo que es utilizada para la comunicación con otro de los componentes. Se representa con una línea que tiene al final un círculo no relleno. Otros módulos pueden conectarse a una interfaz. Esto se hace cuando un componente requiere o utiliza al otro componente mediante su interfaz, que son las operaciones externas que ofrece el componente. Se representa con una línea que termina en un semicírculo que rodea la interfaz del otro componente. En el diagrama se vería de la siguiente manera [38].

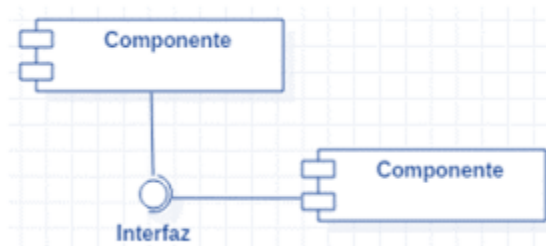


Ilustración 17. Utilización de interfaz.

2.4.5. Relación de dependencia

La relación de dependencias representa que un componente requiere de otro para ejecutar su trabajo. Es diferente a la interfaz, pues identifica que un componente ofrece una serie de operaciones. En cualquier caso, en ocasiones para simplificar el diagrama no se usan las interfaces, sino que solamente se utilizan relaciones de dependencia. Una relación de dependencia se representa mediante una flecha discontinua que va desde el componente que requiere de otro componente hasta el requerido [39].

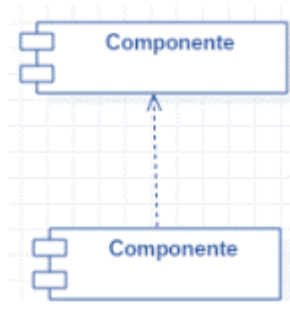


Ilustración 18 Notación de una relación de dependencia.

2.5. Diagrama de clases

El diagrama de clases en UML es una representación visual estática que describe los tipos de objetos en un sistema, así como las relaciones entre ellos. Es una herramienta fundamental para el modelado conceptual en desarrollo de software orientado a objetos, muestra las clases, interfaces y las relaciones de asociación, dependencia y generalización entre ellas, puede incluir paquetes o subsistemas para organizar los elementos del modelo en partes más grandes. Similar al modelo Entidad-Relación (E/R), puede incluir notas explicativas y restricciones que proporcionan información adicional sobre las clases y las relaciones, ofrece una visión integral y detallada de la arquitectura y el diseño del sistema [40].

2.5.1. Componentes

El diagrama de clases, una herramienta esencial en el modelado de sistemas orientados a objetos, se estructura alrededor de elementos cruciales que permiten representar de manera clara y precisa la arquitectura del sistema. Un diagrama de clases está compuesto por los siguientes elementos:

- **Clase:** Atributos, Métodos y Visibilidad.
- **Relaciones:** Asociación, Agregación, Composición y Herencia [41].

2.5.2. Clase

La clase constituye la unidad elemental que encapsula todos los detalles pertinentes de un objeto, siendo este último una instancia específica de dicha clase. Mediante la clase, se puede modelar exhaustivamente el contexto bajo análisis, ya sea este una casa, un automóvil, una cuenta corriente u otro elemento del sistema en cuestión [42].

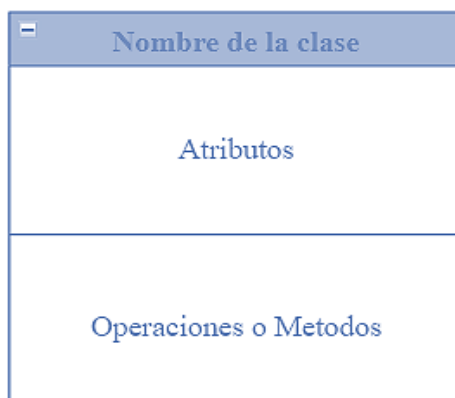


Ilustración 19. Estructura de la clase.

2.5.3. Atributos

Un atributo denota una cualidad o característica inherente a la clase que se manifiesta en todas las instancias de dicha clase. Estos atributos pueden ser representados de diversas formas: mediante su nombre solamente, mostrando tanto el nombre como el tipo, e incluso exhibiendo su valor por defecto, si lo hubiera. Las características o atributos de una clase pueden clasificarse en cuatro tipos distintos, los cuales determinan el nivel de acceso y visibilidad que tienen hacia el entorno, estos son [43]

Visibilidad	Significado	Símbolo
Publica	Se puede acceder al miembro de la clase desde cualquier lugar.	+
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	#
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	-
Paquete local	Se puede acceder a los miembros de una clase desde cualquier clase en el mismo paquete	~

Los atributos son elementos que definen la estructura tanto de una clase como de los objetos asociados a ella, estableciendo el valor de un dato para todas las instancias de esa clase. Los nombres de los atributos deben ser únicos dentro de una clase, aunque pueden repetirse en diferentes clases. Los atributos carecen de identidad propia, a diferencia de los objetos, lo que significa que dos atributos con el mismo nombre pueden tener valores diferentes en diferentes instancias de la clase. Este tipo de atributo se conoce también como "atributo básico" [44].

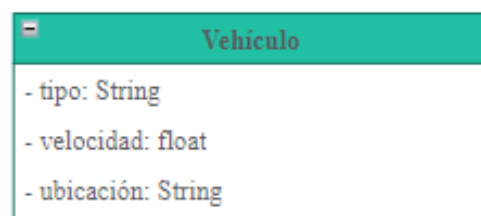


Ilustración 20 Atributos de una clase.

2.5.4. Identificadores

Los identificadores son elementos utilizados exclusivamente para propósitos de implementación en un sistema, y deben diferenciarse de los atributos, que reflejan las características de los objetos en el mundo real. Por ejemplo, el número de Seguro Social o la licencia de conducir son identificadores del mundo real, mientras que un identificador interno para distinguir entre objetos de tipo persona no debe ser considerado como un atributo en el diagrama de clases. Es esencial evitar incluir estos identificadores internos como atributos en la representación de la clase. En la figura siguiente se muestra cómo se debe evitar (forma incorrecta) y cómo se debe representar adecuadamente (forma correcta) un identificador en el diagrama de clases [45].

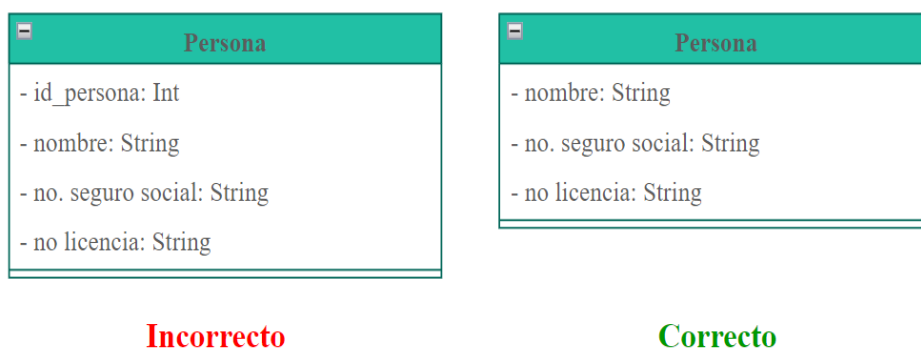


Ilustración 21. Forma correcta de los Identificadores

2.5.5. Métodos

Un método u operación en una clase representa la implementación de un servicio específico, mostrando un comportamiento que es común a todas las instancias de esa clase, es una función que indica a las instancias de la clase qué acciones llevar a cabo, las operaciones son funciones o transformaciones aplicadas a todos los objetos de una clase particular, pudiendo ser acciones ejecutadas por el objeto o acciones realizadas sobre el objeto mismo cada operación debe ser única dentro de una misma clase, aunque no necesariamente entre diferentes clases. Es fundamental evitar utilizar el mismo nombre para operaciones que tengan significados distintos [46].

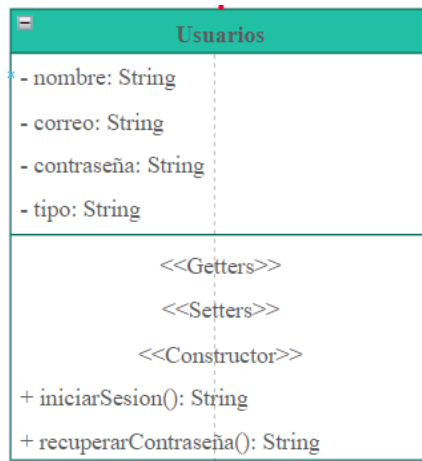


Ilustración 22. Ejemplo de clase con métodos.

2.5.6. Relaciones

Existen cuatro tipos distintos de relaciones entre clases en UML: Herencia, Agregación, Composición y Asociación. En estas relaciones, se identifica una clase de origen y una clase de destino. La clase de origen es aquella desde la cual se establece la relación, es decir, de donde parte la flecha; mientras que la clase de destino es aquella que recibe la flecha [46].

2.5.7. Multiplicidad

La cardinalidad de una relación indica el grado y nivel de dependencia entre las clases, es decir, especifica cuántas instancias de una clase pueden relacionarse con una sola instancia de otra clase, se anotan en cada extremo de la relación [47].

Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

Ilustración 23 Cardinalidad

2.5.8. Interfaces

Describe operaciones abstractas que pueden ser implementadas por elementos como paquetes, componentes o clases. A diferencia de las clases, las interfaces no contienen atributos ni métodos con implementaciones concretas, pueden participar en relaciones de asociación, dependencia y generalización, representa como un círculo pequeño con su nombre debajo, conectado mediante una línea continua a los elementos que la implementan, se establece una relación de asociación. Si una clase utiliza una interfaz implementada en otra clase, la clase dependiente solo tiene acceso a las operaciones definidas en la interfaz, sin depender de la implementación específica de la clase. [42].

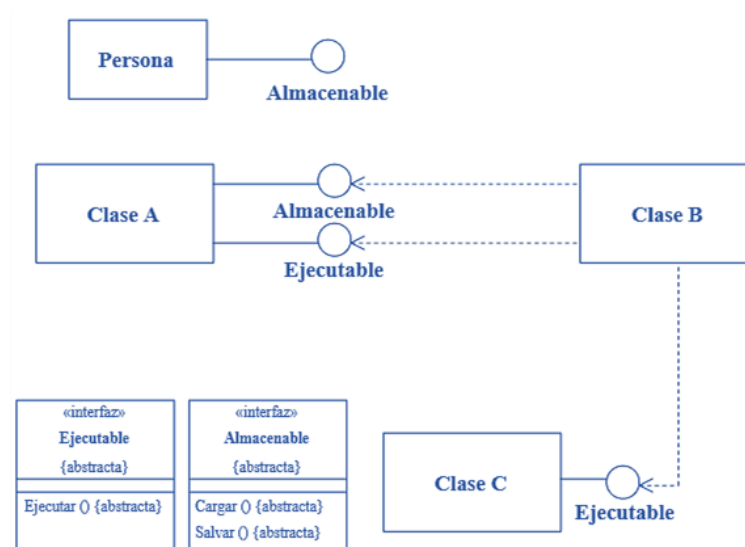


Ilustración 24. Implementación de la interfaz.

2.5.9. Asociación

Conexión estructural que indica que los objetos de una clase están relacionados con los objetos de otra clase. Si esta conexión involucra solo dos clases, se llama asociación binaria, aunque también es posible tener asociaciones n-arias que conecten más de dos clases, es bidireccional, lo que significa que se puede navegar en ambas direcciones. Se representa como una línea sólida entre las clases conectadas. Sin embargo, también puede ser unidireccional, indicado por una flecha al final de la línea, que señala la dirección permitida de la asociación [44].

Para detallar la relación, una asociación puede tener elementos adicionales:

- **Nombre:** Describe la naturaleza de la relación y se indica la dirección de lectura con un triángulo.
- **Rol:** Identifica el papel específico que una clase desempeña en la asociación.
- **Multiplicidad:** Define la cantidad de objetos conectados en una instancia de la asociación.
- **Restricciones:** Proporcionan significado adicional a la asociación mediante restricciones específicas.

La figura siguiente se ilustra dos tipos de navegación en una asociación entre las clases "Persona" y "Coche". En el primer caso, un coche puede tener uno o más propietarios, mientras que una persona puede ser propietaria de cero o más coches. En el segundo caso, se indica que una persona puede poseer varios coches o ninguno, pero no se especifica cuántas personas pueden ser propietarias de un coche [44].

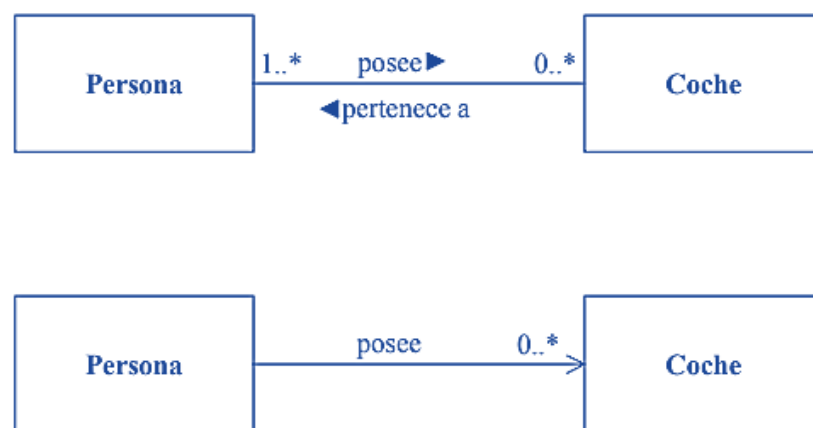


Ilustración 25. Asociación en el diagrama de clases [44].

2.5.10. Agregación

Variante específica de asociación que denota un vínculo más estrecho entre clases, donde una de ellas desempeña un papel más crucial que la otra en la relación. Tanto la agregación normal como la compartida representan relaciones menos fuertes, compartiendo características similares con la asociación en términos de implementación, los ciclos de vida de la clase "todo" y sus partes no están sincronizados [42].

En la siguiente ilustración se presentan dos ejemplos ilustrativos de los dos tipos de agregaciones: normal, compartida. En el primer ejemplo, se muestra una agregación normal donde una flota contiene múltiples barcos de guerra. Los barcos pueden ser agregados o eliminados, pero la flota sigue existiendo, es característico de una agregación normal, donde las partes (los barcos de guerra) componen el todo (la flota). En el segundo ejemplo, se ilustra una agregación compartida. Un remix está conformado por varias bandas sonoras, y una misma banda sonora puede estar presente en varios remixes, la multiplicidad en el lado del "todo" es mayor que 1, se observa que los objetos de las clases "Remix" y "Banda Sonora" están ordenados, lo cual se indica mediante la restricción {ordenado} [46].

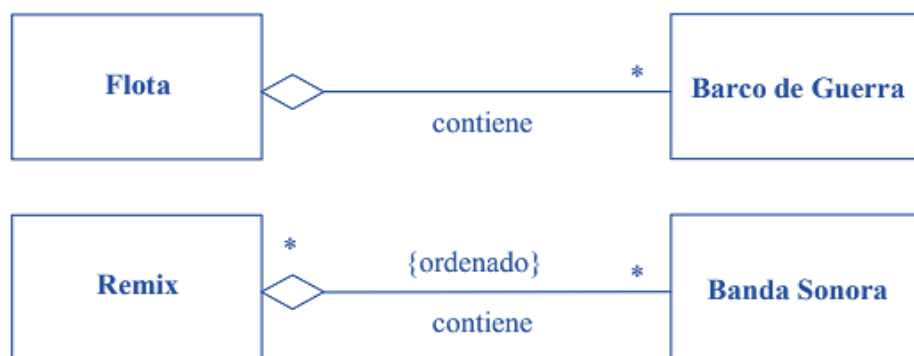


Ilustración 26. Agregación en el diagrama de clases [43].

2.5.11. Composición

La agregación de composición representa una conexión muy sólida, implicando una dependencia existencial, donde la clase dependiente desaparece cuando se elimina la clase contenedora, y una pertenencia fuerte, donde la clase contenida es esencial para la existencia de la clase contenedora, una forma común de implementar la agregación de composición es incluir físicamente las partes dentro de la clase "todo" (encapsulamiento). Aquí es donde se distinguen los tipos de agregaciones en términos de implementación. En la agregación de composición, la clase "todo" controla el ciclo de vida de sus partes, lo que significa que cuando se destruye la clase "todo", también se eliminan sus partes [45].

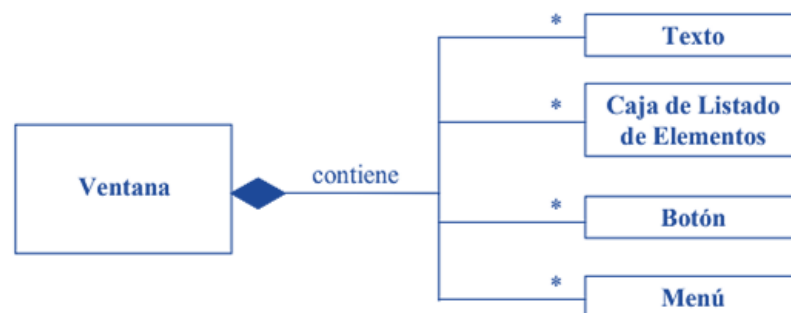


Ilustración 27. Composición en el diagrama de clases.

2.5.12. Generalización o Herencia

La generalización normal representa una relación de herencia donde una clase específica, llamada subclase, hereda todos los atributos, operaciones y asociaciones de una clase más general, llamada superclase. Los atributos y operaciones con visibilidad pública en la superclase se heredan como públicos en la subclase, mientras que los que tienen visibilidad privada se heredan, pero no son accesibles desde la subclase, a menos que se declaren como protegidos. Esta relación se representa visualmente como una línea continua desde la subclase hacia la superclase, con un triángulo vacío en el extremo de la superclase [46].

En este ejemplo, la clase "Vehículo" representa características comunes entre coches y barcos, y también tiene una asociación con la clase "Persona". "Vehículo" es una clase abstracta que no puede tener instancias directas, mientras que las clases "Coche" y "Barco" son clases concretas. Estas subclases heredan el atributo "color" y la operación "Conducir" de la superclase "Vehículo", la operación "Conducir" se redefine en las clases "Coche" y "Barco", está marcada como abstracta en la clase "Vehículo" y su implementación varía dependiendo del tipo de vehículo [47].

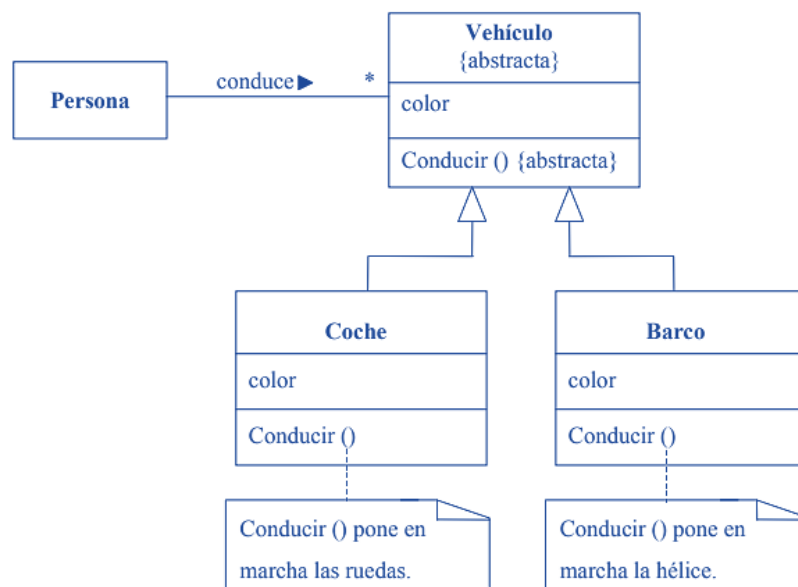


Ilustración 28. Herencia en el diagrama de clases.

2.5.13. Notas

Comentario que proporciona información adicional sobre elementos específicos del diagrama. Puede estar vinculada a uno o varios elementos mediante líneas punteadas. Se representa como un rectángulo con el borde superior derecho doblado, se utilizan para ofrecer aclaraciones sobre el diagrama o para expresar restricciones relacionadas con los elementos específicos. Por ejemplo, el texto entre "{" y "}" puede indicar restricciones sobre los elementos relacionados [48].

2.6. Diagramas BPMN

Los Diagramas BPMN son una abreviatura para el Modelado de Procesos de Negocio y Notación, son una herramienta crucial para representar visualmente los procesos internos de una empresa. Es decir, su objetivo principal es proporcionar una representación comprensible de los procedimientos empresariales para todas las partes interesadas. En este contexto, los diagramas BPMN facilitan la comunicación estandarizada entre los usuarios, simplificando así el diseño y la implementación de procesos empresariales [71].

2.6.1. Conceptos básicos de BPMN

BPMN fue desarrollado originalmente por la Iniciativa de Gestión de Procesos de Negocio (BPMI) y actualmente mantenido por el Grupo de Gestión de Objetos (OMG), el estándar BPMN ha evolucionado a lo largo de los años, con la versión más reciente, BPMN 2.0, lanzada en 2011. Esta iteración introdujo nuevas características como eventos de interrupción, subprocesos más detallados, entre otros. Esto, permitió una representación aún más precisa de los procesos de negocio [72].

La especificación de BPMN representa la combinación de las mejores prácticas dentro de la comunidad del modelado empresarial para definir la notación y semántica de los diagramas de Colaboración, Procesos y Coreografía. Su intención es estandarizar un modelo y notación de procesos de negocio frente a muchas notaciones y puntos de vista de modelado diferentes, proporcionando un medio simple para comunicar información sobre procesos a otros usuarios comerciales, clientes y proveedores [73].

2.6.2. Usos comunes de BPMN

El modelado BPMN se utiliza comúnmente para comunicar una amplia variedad de información a diversos públicos. BPMN está diseñado para cubrir muchos tipos de modelado y permite la creación de procesos empresariales de principio a fin. Sus elementos estructurales facilitan al espectador la diferenciación entre secciones de un diagrama BPMN [73]. Dentro de un modelo BPMN, existen tres tipos básicos de submodelos:

1. Procesos (Orquestación), que incluyen:

- Procesos de Negocio Privados no Ejecutables (internos)
 - Procesos de Negocio Ejecutables Privados (internos)
 - Procesos Públicos
2. Coreografías
 3. Colaboraciones, que pueden incluir Procesos y/o Coreografías
 - Una vista de Conversaciones

2.6.3. Modelamiento con BPMN en sistemas distribuidos

El modelado con BPMN nos brinda una herramienta poderosa para comprender y optimizar sistemas distribuidos de manera eficiente y efectiva. Al utilizar esta notación, podemos trazar visualmente los flujos de trabajo y las interacciones entre los diversos componentes de nuestros sistemas, identificando áreas de mejora y oportunidades para la innovación [74].

Desde la simplificación de procesos hasta la identificación de cuellos de botella potenciales, el modelado con BPMN nos permite tomar decisiones informadas y estratégicas que impulsan el rendimiento y la eficiencia de nuestros sistemas distribuidos. En un mundo cada vez más conectado, el modelado con BPMN se convierte en una herramienta invaluable para construir puentes entre la visión y la implementación, permitiéndonos crear sistemas distribuidos que sean robustos, adaptables y centrados en el usuario [75].

2.6.4. Elementos de BPMN

Uno de los objetivos principales del desarrollo de BPMN es crear un mecanismo simple y comprensible para crear modelos de procesos de negocio, y al mismo tiempo ser capaz de manejar la complejidad de los procesos de negocio. Es así, como el enfoque adoptado para manejar estos dos requisitos fue organizar los aspectos gráficos de la notación en categorías específicas [76]. Estas categorías básicas de elementos son:

1. Objetos de flujo
2. Datos
3. Conectando objetos
4. Carriles

5. Artefactos

También, tenemos los objetos de flujo que son los principales elementos gráficos para definir el comportamiento de un proceso de negocio [77]. Existen tres flujos de objetos:

1. Eventos
2. Actividades
3. Puertas de enlace

2.6.5. Notación y símbolos utilizados en BPMN

Otros de las notaciones y símbolos que se utilizan en el diagrama de BPMN para representar los procesos de negocio son los siguientes [78]:

Para representar la notación y modelo de procesos de negocio. Los datos se representan con los cuatro elementos siguientes:

1. Objetos de datos
2. Entradas de datos
3. Salidas de datos
4. Almacenes de datos

Hay cuatro formas de conectar los objetos de flujo entre sí o con otra información. Estos objetos de conexión son:

1. Flujos de secuencia
2. Flujos de mensajes
3. Asociaciones
4. Asociaciones de datos

Hay dos formas de agrupar los elementos primarios de modelado a través de “Carriles”:

1. Pool
2. Carriles

Por otro lado, tenemos los artefactos se utilizan para proporcionar información adicional sobre el Proceso. Existen dos artefactos estandarizados. El conjunto actual de artefactos incluye:

1. Grupo
2. Anotación de texto

2.6.6. Eventos

Los eventos representan un evento en un proceso de negocio [79].

Un Evento se define como un suceso que ocurre durante el desarrollo de un Proceso o Coreografía, y ejerce influencia sobre el flujo del modelo, generalmente asociado con una causa o un efecto. Estos Eventos se representan mediante círculos con centros abiertos para permitir la diferenciación de distintos desencadenantes o resultados. Según el momento en que impactan en el flujo, existen tres tipos de eventos: de inicio, intermedios y finales [80].

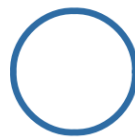


Ilustración 29. Eventos [79].

2.6.7. Eventos de inicio

El evento de inicio indica el punto donde un proceso o coreografía específica comenzará. En el contexto de los flujos de secuencia, el inicio de un proceso es marcado por el evento inicial, el cual no cuenta con flujos de secuencia entrantes debido a su función de iniciar el flujo del proceso. Es importante señalar que ningún flujo de secuencia puede estar conectado directamente a este evento. Además, el evento inicial presenta una estructura visual similar a la de los eventos intermedios y finales, consistente en un círculo con un centro abierto, que permite la inclusión de marcadores para indicar distintas variaciones del evento [81].



Ilustración 30. Evento inicio [81].

2.6.8. Eventos intermedios

Los eventos intermedios ocurren entre un evento de inicio y un evento de fin. Afectarán el flujo del proceso, pero no iniciarán ni terminarán directamente el proceso. El evento intermedio comparte la misma forma básica del evento de inicio y evento final, un círculo con un centro abierto para que se puedan colocar marcadores dentro del círculo para indicar variaciones del evento [82].

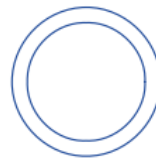


Ilustración 31. Evento intermedio [82].

2.6.9. Eventos finales

El evento de fin señala el punto donde un proceso o coreografía llegará a su término. El evento final comparte la misma forma básica del evento inicial y el evento intermedio, un círculo con un centro abierto para que se puedan colocar marcadores dentro del círculo para indicar variaciones del evento [83].



Ilustración 32. Evento final [83].

2.6.10. Actividades

Una actividad es el trabajo que se realiza dentro de un proceso empresarial. Esta puede ser atómica o no atómica (compuesta). Los tipos de actividades que forman parte de un proceso son: tarea, subproceso y actividad de llamada, que permite la inclusión de tareas y procesos reutilizables en el diagrama. Sin embargo, un proceso no es un objeto gráfico específico. En cambio, es un conjunto de objetos gráficos. Las actividades representan puntos en un flujo de proceso donde se realiza trabajo [84].



Ilustración 33. Actividades [84].

2.6.11. Puertas de enlace.

En un proceso de orquestación, las puertas de enlace se utilizan para crear caminos alternativos y/o paralelos para ese proceso. La coreografía tiene el mismo requisito de caminos alternativos y paralelos. Es decir, las interacciones entre los participantes pueden ocurrir en secuencia, en paralelo o a través de una selección exclusiva. Aunque los caminos de la coreografía siguen los mismos patrones básicos que los de un proceso de orquestación, la falta de un mecanismo central para mantener la visibilidad de los datos y que no hay una evaluación central [85].



Ilustración 34. Puertas de enlace [85].

2.6.12. Flujo de secuencia

Un flujo de secuencia se utiliza para mostrar el orden de los elementos de flujo en un proceso o una coreografía. Cada flujo de secuencia tiene solo una fuente y solo un objetivo. La fuente y el objetivo deben ser del conjunto de los siguientes elementos de flujo: eventos (inicio, intermedio y final), actividades (tarea y subproceso; para procesos), actividades de coreografía (tarea de coreografía y sub-coreografía; para coreografías) y puertas de enlace [86].



Ilustración 35. Flujo de secuencia [86].

2.6.13. Flujo de mensajes

Un flujo de mensajes se utiliza para mostrar el flujo de mensajes entre dos participantes que están preparados para enviarlos y recibirlos. Un flujo de mensajes debe conectar dos grupos separados. Se conectan ya sea al límite del grupo o a objetos de flujo dentro del límite del grupo. No deben conectar dos objetos dentro del mismo grupo [87].

Un flujo de mensajes es una línea con un inicio de línea de círculo abierto y un extremo de línea de punta de flecha abierta que debe dibujarse con una línea discontinua simple [87].



Ilustración 36. Flujo de mensajes [87].

2.6.14. Asociación

Una asociación se utiliza para asociar información y artefactos con objetos de flujo. Texto y objetos gráficos no relacionados con el flujo pueden asociarse con los objetos de flujo y el flujo. Una asociación también se utiliza para mostrar la actividad utilizada para compensación [88].



Ilustración 37. Asociación [88].

2.6.15. Pool

Un pool es la representación gráfica de un participante en una colaboración. Un participante puede ser una entidad asociada específica (por ejemplo, una empresa) o puede ser un rol de asociado más general (por ejemplo, un comprador, vendedor o fabricante). Un pool puede o no puede hacer referencia a un proceso. Un pool no está obligado a contener un proceso, es decir, puede ser una "caja negra" [89].



Ilustración 38. Pool [89].

2.6.16. Carril

Un carril es una sub-partición dentro de un proceso (a menudo dentro de un pool) y se extenderá a lo largo de toda la longitud del nivel del proceso, ya sea verticalmente u horizontalmente. El texto asociado con el carril (por ejemplo, su nombre y/o el de cualquier atributo del elemento del proceso) se puede colocar dentro de la forma, en cualquier dirección o ubicación, según la preferencia del modelador o del proveedor de herramientas de modelado. Nuestros ejemplos colocan el nombre como un banner en el lado izquierdo (para pool horizontales) o en la parte superior (para pool verticales) del otro lado de la línea que separa el nombre del pool, sin embargo, esto no es un requisito [90].

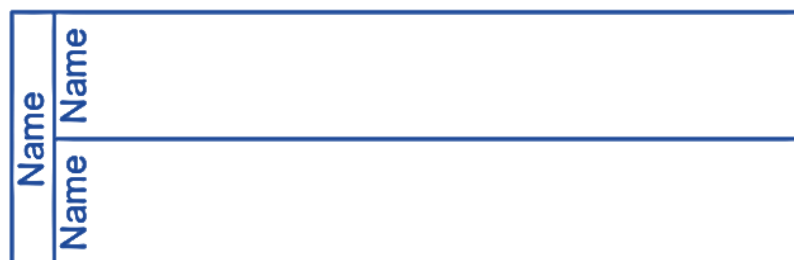


Ilustración 39. Carril [90].

2.6.17. Objeto de datos

La clase de objeto de datos es un elemento consciente de los elementos. Los elementos de objeto de datos deben estar contenidos dentro de elementos de proceso o subprocesso. Los elementos de objeto de datos se muestran visualmente en un diagrama de proceso. Las referencias de objeto de datos son una forma de reutilizar objetos de datos en el mismo

diagrama. Pueden especificar diferentes estados del mismo objeto de datos en diferentes puntos de un proceso. La referencia de objeto de datos no puede especificar definiciones de elementos y los objetos de datos no pueden especificar estados [91].



Ilustración 40. Objeto de datos [91].

2.6.18. Mensaje

Un mensaje representa el contenido de una comunicación entre dos participantes. En BPMN 2.0, un mensaje es un decorador gráfico (era un elemento de soporte en BPMN 1.2). Se utiliza una definición de elemento para especificar la estructura del mensaje. Cuando se muestra en un diagrama: en un mensaje es un rectángulo con líneas diagonales convergentes en la mitad superior del rectángulo para dar la apariencia de un sobre. Debe dibujarse con una sola línea delgada [75].



Ilustración 41. Mensaje [75].

2.6.19. Grupo (un cuadro alrededor de un grupo de objetos dentro la misma categoría)

Los grupos son una forma en que las categorías de objetos pueden mostrarse visualmente en el diagrama. Es decir, un grupo es una representación visual de un único valor de categoría. Los elementos gráficos dentro del grupo se asignarán al valor de categoría del grupo. El valor del valor de categoría, opcionalmente precedido por el nombre de la categoría y el delimitador ":", aparece en el diagrama como la etiqueta del

grupo. Una sola categoría puede usarse para varios grupos en un diagrama [76].



Ilustración 42. Grupo [76].

2.6.20. Símbolos de eventos de BPMN

Es posible personalizar el aspecto de todos estos eventos para reflejar los detalles particulares de tus procesos. Los ejemplos siguientes se aplican a los símbolos de eventos de inicio, pero son aplicables a cualquier tipo de evento [77]. Los símbolos de eventos más comunes representan las situaciones siguientes:

Símbolo de mensaje - El símbolo de mensaje inicia el proceso, facilita las etapas intermedias o finaliza el proceso [77].



Ilustración 43. Símbolo de mensaje [77].

Símbolo de temporizador: Una fecha, hora o una combinación de ambas que activa, asiste o concluye el proceso en etapas intermedias [78].



Ilustración 44. Símbolo de temporizador [78].

Símbolo de escalación: Un paso que reacciona ante una escalada y se dirige hacia otro rol en la organización. Este evento se utiliza exclusivamente en un subproceso de evento. Una escalación ocurre cuando alguien con un nivel superior de responsabilidad dentro de la organización participa en un proceso [79].



Ilustración 45. Símbolo de escalación [79].

Símbolo condicional: El proceso se inicia o continúa cuando se cumple una condición o regla de negocio [80].



Ilustración 46. Símbolo condicional [80].

Símbolo de enlace: Un subproceso que forma parte de un proceso más amplio [81].



Ilustración 47 Símbolo de enlace [81].

Símbolo de error: Un error detectado al principio, durante o al final de un proceso. Un subproceso de evento con un activador de error siempre interrumpe el proceso que lo contiene [82].



Ilustración 48. Símbolo de error [82].

Símbolo de cancelación: Reacciona a una transacción que se ha cancelado dentro de un subproceso. En un evento de finalización, indica que se ha activado la cancelación de un proceso [83].



Ilustración 49. Símbolo de cancelación [83].

Símbolo de compensación: Un reembolso activado cuando las operaciones fallan parcialmente [84].



Ilustración 50. Símbolo de compensación [84].

Símbolo de señal: Una señal comunicada en varios procesos, que puede iniciar, asistir o completar un proceso [85].



Ilustración 51. Símbolo de señal [85].

Símbolo múltiple: Activadores múltiples que inician un proceso [86].



Ilustración 52. Símbolo múltiple [86].

Símbolo de paralelas múltiples: Una instancia de proceso que no comienza, continúa o finaliza hasta que se hayan realizado todos los eventos posibles [87].



Ilustración 53. Símbolo de paralelas múltiples[87].

Símbolo de finalización: Activa la finalización inmediata de un paso del proceso, donde todas las instancias relacionadas finalizan simultáneamente [88].



Ilustración 54. Símbolo de finalización[88].

2.6.21. Ejemplo del diagrama BPMN

Se muestra un ejemplo, aplicando el diagrama de BPMN a un contexto de un sistema compra y venta de productos [90].

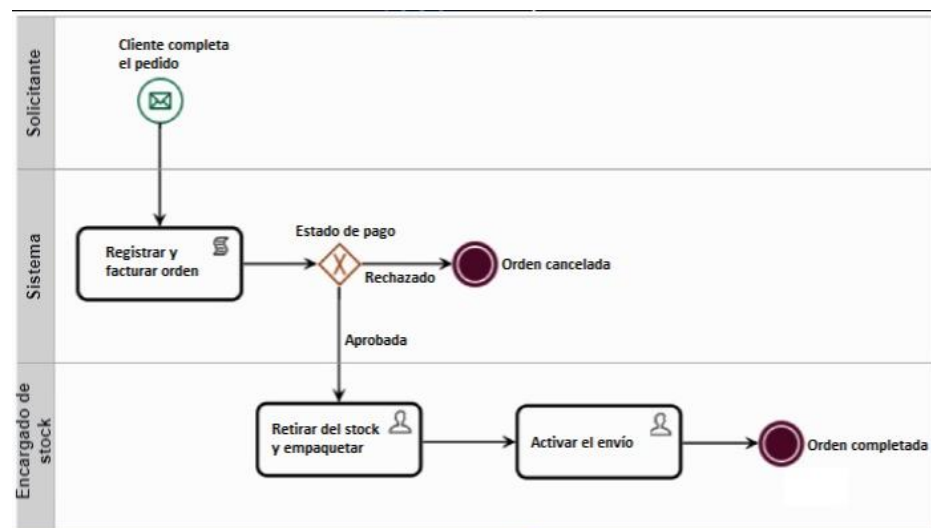


Ilustración 55. Ejemplo de diagrama BPMN [90].

2.7. Redes de Petri

Las redes de Petri son un método formal empleado para representar transiciones entre estados en un marco de tiempo discreto. Han encontrado aplicación en diversos ámbitos como sistemas concurrentes, software y modelos epidemiológicos. Este formalismo permite modelar sistemas simultáneos y detectar posibles fallos antes de su ejecución. Además, pueden contribuir a optimizar programas computacionales al establecer la máxima concurrencia permitida según la estructura de la red de Petri [92].

2.7.1. Importancia de la modelación con redes de Petri

La utilización de redes de Petri proporciona un sólido marco teórico que facilita la representación gráfica y matemática de los sistemas. Esta combinación permite a investigadores y desarrolladores examinar aspectos críticos como la vivacidad, la seguridad y la prevención de condiciones conflictivas, entre otros. La capacidad de las redes de Petri para representar y analizar tanto el flujo de información como el control en sistemas distribuidos y paralelos las posiciona como una herramienta esencial en la ingeniería de software y sistemas [93].

2.7.2. Conceptos de redes de Petri

Las Redes de Petri constituyen una categoría de modelos matemáticos y gráficos empleados para representar sistemas dinámicos que son concurrentes y distribuidos. Estas redes se componen de un grafo bipartito y dirigido, que consta de dos tipos de nodos: plazas y transiciones. Las plazas representan los estados del sistema donde se almacenan los recursos o tokens, mientras que las transiciones simbolizan los eventos o acciones que pueden tener lugar en el sistema. Estos nodos se conectan a través de arcos que indican las relaciones de dependencia entre ellos [94].

El estado de una Red de Petri se caracteriza por su marcaje, que describe la distribución de tokens en las plazas en un momento determinado. Este marcaje refleja la situación actual del sistema y cómo se distribuyen los recursos o información en ese instante. La evolución de la red se lleva a cabo mediante el disparo de transiciones, donde una transición puede activarse si cuenta con los tokens necesarios en sus plazas de entrada, consumiendo dichos tokens y generando nuevos en las plazas de salida [95].

2.7.3. Elementos de una red de Petri

Las redes Petri están compuestas por varios elementos importantes que la componen, a continuación se los muestra con un concepto cada uno:

Plazas y Transiciones: En una Red de Petri, las plazas representan los estados del sistema donde se guardan recursos, tokens o información pertinente. Por otro lado, las transiciones denotan eventos o acciones que

pueden suceder en el sistema, como la realización de una tarea o la disponibilidad de un recurso [96].

Arcos: Los arcos en una Red de Petri son las conexiones entre plazas y transiciones, estableciendo las relaciones de dependencia entre ellas. Un arco dirigido de una plaza a una transición indica que la transición necesita que haya tokens en esa plaza para poder activarse. En contraste, un arco de una transición a una plaza señala que al activarse la transición, se generan tokens en esa plaza [97].

Tokens: Los tokens son unidades discretas presentes en las plazas, simbolizando la disponibilidad de recursos, la ocurrencia de eventos o el estado del sistema en un instante específico. La presencia o ausencia de tokens en una plaza determina el estado del sistema en ese momento [98].

Disparo de Transiciones: Una transición en una Red de Petri puede dispararse si cuenta con al menos un token en cada una de sus plazas de entrada. Al activarse, la transición consume los tokens de entrada y produce tokens en las plazas de salida, reflejando un cambio de estado en el sistema y la ocurrencia del evento asociado a la transición [99].

Marcaje: El marcaje de una Red de Petri representa la distribución de tokens en las plazas en un momento determinado, reflejando así el estado actual del sistema y cómo se distribuyen los recursos o la información en ese momento. Con cada disparo de transiciones, el marcaje evoluciona y el sistema experimenta cambios en su estado [100].

Regla de Disparo: La regla de disparo de una transición especifica las condiciones que deben cumplirse para que una transición se active. Por lo general, una transición está lista para activarse si dispone de todos los tokens necesarios en sus plazas de entrada, cumpliendo así con los requisitos previos para su ejecución [101].

2.7.4. Propiedades y características de las redes de Petri

Las redes de Petri constituyen un modelo matemático y visual empleado para explicar sistemas distribuidos, concurrentes y paralelos. Poseen

atributos y peculiaridades particulares que las convierten en herramientas valiosas para el análisis y la planificación de sistemas [102].

2.7.5. Propiedades

Bipartitas: Una red de Petri se compone de un grafo bipartito, lo que implica que está conformada por dos conjuntos de nodos: lugares (o estados) y transiciones. Los lugares albergan tokens que simbolizan el estado del sistema, mientras que las transiciones representan eventos que pueden ocurrir para cambiar de un estado a otro [103].

No determinismo: Las redes de Petri tienen la capacidad de modelar sistemas no deterministas, donde múltiples transiciones pueden estar habilitadas simultáneamente y la elección de cuál activar puede depender de reglas específicas [104].

Disparo de transiciones: Las transiciones en una red de Petri se activan cuando cuentan con la cantidad suficiente de tokens en sus lugares de entrada. Este mecanismo permite modelar la evolución del sistema a medida que se producen eventos [105].

Estado del sistema: La representación del estado global del sistema se logra mediante la distribución de tokens en los lugares de la red de Petri. Esta representación facilita la visualización y comprensión del comportamiento del sistema [106].

2.7.6. Características

Modelado de sistemas concurrentes: Las redes de Petri demuestran una utilidad significativa en el modelado de sistemas concurrentes, donde múltiples eventos pueden tener lugar simultáneamente [107].

Análisis de propiedades: Facilitan el análisis de propiedades cruciales del sistema, como la accesibilidad de ciertos estados, la prevención de bloqueos y la conservación de recursos [107].

Expresividad: Las redes de Petri exhiben un nivel de expresividad tal que les permite modelar una amplia variedad de sistemas, desde los más simples hasta los complejos sistemas de tiempo real [108].

Verificación y validación: Son aptas para verificar y validar el comportamiento de sistemas previo a su implementación, lo cual ayuda a detectar posibles fallos y problemas de diseño [108].

2.7.7. Modelamiento de sistemas distribuidos

El modelado de sistemas distribuidos conlleva el desarrollo y la validación de sistemas complejos a gran escala. Estos sistemas se distinguen por su naturaleza distribuida, donde los componentes son autónomos y se comunican de forma asincrónica. Los formalismos de modelado tradicionales derivados de sistemas centralizados resultan poco realistas para abordar sistemas distribuidos, por lo que se ha propuesto un modelo integrado y sincronizado para sistemas distribuidos [109]. Este enfoque aborda la asincronía, la autonomía y la localidad del comportamiento en sistemas distribuidos, y puede ser traducido en autómatas temporizados para su verificación. La adopción de este formalismo facilita la verificación automatizada de propiedades como la ausencia de bloqueos y la eficiencia de la terminación. Además, los sistemas de aprendizaje profundo distribuidos pueden ser modelados y optimizados para anticipar los beneficios de rendimiento de la capacitación en sistemas distribuidos en contraste con máquinas individuales [110].

2.7.8. Modelamiento con redes de Petri en sistemas distribuidos

El modelado mediante redes de Petri presenta varias ventajas en el contexto de sistemas distribuidos. En primer lugar, estas redes ofrecen una notación gráfica que permite representar sistemas complejos donde múltiples dispositivos operan secuencial o simultáneamente. Esto simplifica la comprensión y la construcción de modelos del sistema [110]. Además, las redes de Petri comparten similitudes con otros lenguajes de modelado, como SFC (GRAFCET), lo que facilita la generación de algoritmos de control discreto implementados con PLCs. Asimismo, estas redes pueden emplearse para evaluar el rendimiento del sistema modelado, identificar formas de mejorar su eficiencia y verificar el correcto funcionamiento de los sistemas de control. La capacidad de simular y verificar el comportamiento del sistema utilizando redes de Petri resulta

especialmente útil para garantizar la ausencia de cuellos de botella y para optimizar la transmisión y el procesamiento de datos [111].

2.7.9. Evaluación y validación de sistemas distribuidos representados mediante con redes de Petri

El análisis y la validación de sistemas distribuidos modelados con redes de Petri constituyen un enfoque integral para abordar los problemas de seguridad en redes dentro de sistemas descentralizados. Este enfoque se centra en la evaluación y verificación de modelos de ataques a la red, comenzando desde la generación automatizada de modelos de componentes hasta la mejora continua de simulaciones de ataques y estrategias defensivas. La combinación de métodos que incluyen el procesamiento automatizado de bases de datos, la selección y construcción de modelos, junto con una validación y verificación rigurosas, garantiza la precisión y confiabilidad de los modelos desarrollados [112].

Este enfoque ejemplifica un método para desarrollar sistemas robóticos celulares cooperativos con un enfoque particular en la seguridad y la corrección. Se destaca la importancia del desarrollo de software basado en componentes y el uso de middleware ROS, así como la utilización de redes de Petri como un enfoque basado en modelos. La adopción de modelos formales puede reducir significativamente el esfuerzo de prueba durante el desarrollo, proporcionando un método eficaz para abordar problemas de complejidad y garantizar la seguridad de sistemas descentralizados mediante el uso de redes de Petri [113].

2.7.10. Ejemplo de Petri

Se muestra un ejemplo, aplicando el modelado de Petri a un contexto de un sistema ensamblaje de piezas [113].

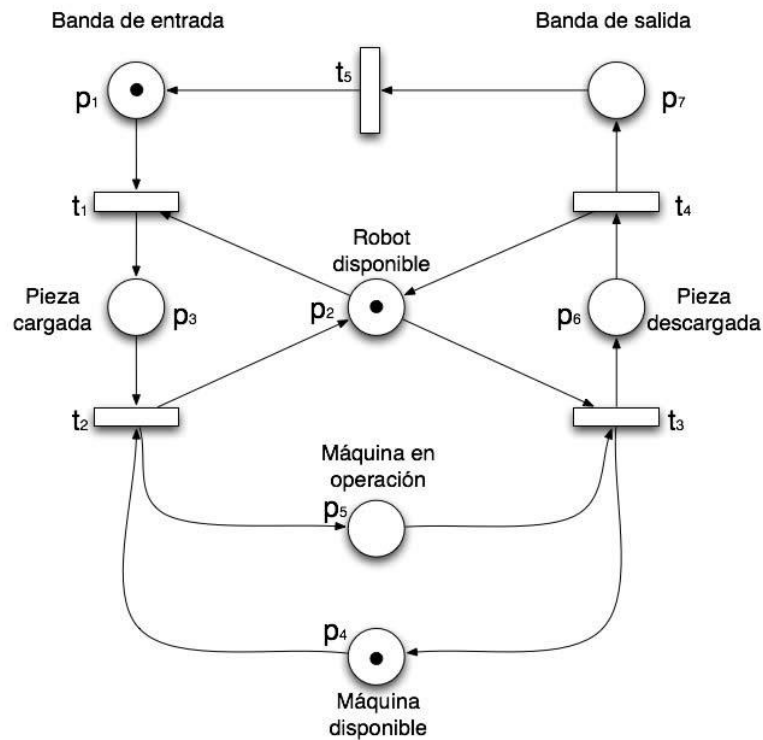


Ilustración 56. Ejemplo de Petri [113].

3. Desarrollo de Sistemas basados en IoT

El Internet de las Cosas (IoT) se ha convertido en una de las áreas tecnológicas con mayores tasas de crecimiento en la era moderna. Se estima que para 2025 habrá aproximadamente 75 millones de dispositivos conectados a IoT en todo el mundo . El termo IoT significa la conexión de dispositivos de computadora integrados en objetos cotidianos a través de internet, lo que les permite enviar y recibir datos [49]. El IoT está modificando todas las áreas industriales, desde la logística y fabricación hasta el cuidado de los edificios inteligentes y la salud.

Sin embargo, el desarrollo de soluciones de IoT confiables y escalables plantea desafíos importantes. Se dice que los sistemas de Internet de las Cosas integran una amplia variedad de tecnologías, desde dispositivos, conectividad y seguridad de red, hasta análisis de datos y servicios en la nube. Además, a medida que interactúan con el mundo físico, factores como la privacidad, la seguridad y la interoperabilidad son cruciales [50].

En esta investigación examinaremos dos temas clave en el desarrollo de sistemas de Internet de las cosas: Métodos de desarrollo, dando énfasis en la selección de métodos, adquisición de datos e interconexión de dispositivos. También exploraremos soluciones

alternativas, investigando opciones de plataformas de servicios de nube y hardware para IoT.

3.1. Metodología de Desarrollo

Para el desarrollo de sistemas IoT, existen diversas metodologías de desarrollo orientado a estos, pasando por metodologías ágiles como Scrum o Kanban, por metodologías tradicionales como el modelo en cascada hasta metodologías híbridas:

3.1.1. Metodologías ágiles

Las metodologías ágiles son enfoques iterativos y flexibles para gestionar proyectos de desarrollo de software. Se centran en la entrega oportuna y continua de software utilizable, la adaptación al cambio y la estrecha colaboración entre el cliente y los equipos autoorganizados encargados del proyecto [51].

Metodología Scrum

Scrum es una metodología ágil que permite adaptarse rápidamente a los cambios en los requisitos, algo importante para los proyectos de Internet de las cosas (IoT) debido a la continua evolución tecnológica. Se concentra en entregas incrementales de software funcional en sprints, que suelen ser de 2 a 4 semanas [52].

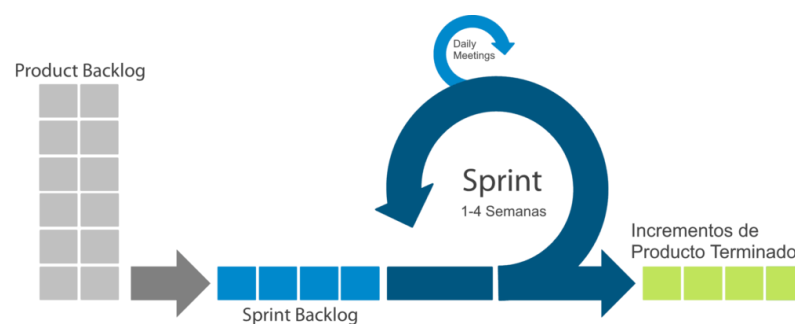


Ilustración 57. Metodología Scrum.

Características

- Autoorganización de equipos pequeños y diversos: desarrolladores, testers y DevOps.
- Entregas continuas e incrementales del producto al final de cada sprint.

- Adaptación a cambios incluso en etapas avanzadas.
- Ritmo de trabajo sostenible sin sobrecargas.
- Transparencia sobre progreso diario dentro del equipo [52], [53].

Desarrollo

Las historias de usuario son priorizadas en el backlog del producto por el propietario del producto. Luego, el equipo selecciona los artículos a completar en el próximo sprint en la planificación de sprints. Scrums diarios de 15 minutos se realizan durante el sprint para sincronizar avances, bloqueos, etc. Finalmente, hay una retrospectiva para identificar mejoras y una revisión de Sprint para validar los resultados [54].

Ventajas

- Rápida obtención de feedback y fail fast (reducción de riesgos).
- Mayor calidad y valor temprano.
- Equipos comprometidos y productividad mejorada.

Metodología Kanban

Kanban, que se centra en la entrega just-in-time de software, es una metodología ágil para la gestión del flujo de trabajo. Ayuda a visualizar y optimizar los flujos de desarrollo en proyectos de Internet de las cosas, especialmente cuando se integran componentes digitales y físicos [55].



Ilustración 58. Metodología Kanban.

Características

- Tablero con columnas para visualizar estado de tareas (Por hacer, En progreso, Completo).
- Límites de trabajo en progreso (WIP) para evitar cuellos de botella.

- Workflow pull-based, se comienza una nueva tarea cuando hay capacidad disponible.
- Entrega continua de software funcionando.
- Mejora incremental centrada en optimizar el flujo.

Desarrollo

Se escriben las tareas en tarjetas y se mueven de izquierda a derecha por el tablero. El equipo de desarrollo se autogestiona al monitorear continua y visualmente su progreso para identificar y eliminar obstáculos [55].

Ventajas

- Rápida identificación de fallas y riesgos para corregir.
- Reduce sobre procesamiento y desperdicio de recursos y tiempo.
- Promueve el trabajo en equipo, la transparencia y la entrega continua.

3.1.2. Metodologías tradicionales

Los enfoques secuenciales y estructurados, que han estado presentes en la ingeniería de software desde la década de 1970, se conocen como metodologías tradicionales o en cascada. Se basan en un ciclo de vida lineal de desarrollo que se divide en etapas como la planificación, el análisis de requisitos, el diseño, la codificación e integración, las pruebas y el mantenimiento [56].

Es importante recalcar que, aunque este tipo de metodologías son aplicables para el desarrollo de sistemas IoT, la rigidez que existe al momento de llevar el flujo de manera unidireccional se puede considerar una limitante en entornos dinámicos como IoT [57].

Modelo cascada

El modelo en cascada, también conocido como metodología en cascada, es un enfoque de desarrollo de software secuencial que organiza el proceso en una secuencia lineal de pasos [57]. Se ha utilizado ampliamente en proyectos de ingeniería de sistemas con requisitos estables.

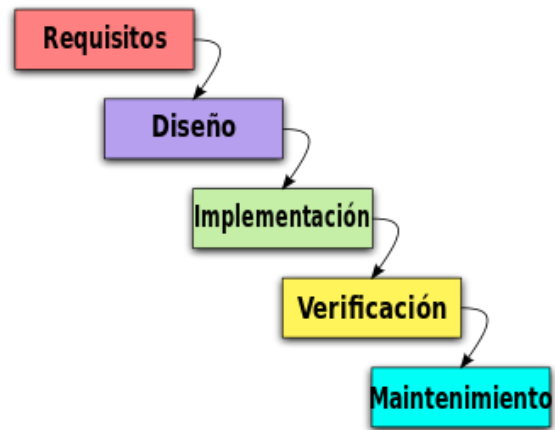


Ilustración 59. Modelo en cascada.

Características

- Progresión secuencial por etapas predefinidas.
- Salidas de cada etapa se convierten en entradas para la siguiente.
- Enfoque meticuloso, basado en documentación y revisiones.

Etapas

Requisitos: En esta etapa se llevan a cabo el análisis y obtención de los requisitos del sistema, esta etapa es la más importante al momento de crear la planificación sobre todo al trabajar con IoT.

Diseño: Se realiza el diseño arquitectónico, de componentes y base de datos.

Implementación: programación e integración de módulos IoT según el diseño.

Verificación: Pruebas del sistema para determinar si cumple requisitos.

Mantenimiento: Correcciones, actualizaciones o mejoras tras el despliegue.

La rigidez del flujo unidireccional es una limitación en entornos dinámicos como IoT [57].

3.1.3. Metodologías híbridas

Las metodologías híbridas combinan técnicas ágiles y predictivas o tradicionales para gestionar proyectos tecnológicos. Adoptan

características de metodologías ágiles como sprints o entregas rápidas, pero mantienen un cierto nivel de pre-planeación y documentación formal de cascada. El objetivo de los métodos híbridos es lograr un equilibrio que combine las mejores características de ambos paradigmas. Por ejemplo, utilizar prácticas ágiles para mantener la flexibilidad para cambios rápidos en los requisitos, pero siempre dentro de una estructura general más formal paso a paso cuando sea necesario [56].

Metodología DevOps

DevOps combina las operaciones de TI con los procesos y herramientas de desarrollo de software (Dev) para permitir la entrega y el despliegue confiable y rápido de actualizaciones en sistemas IoT [58], [59].

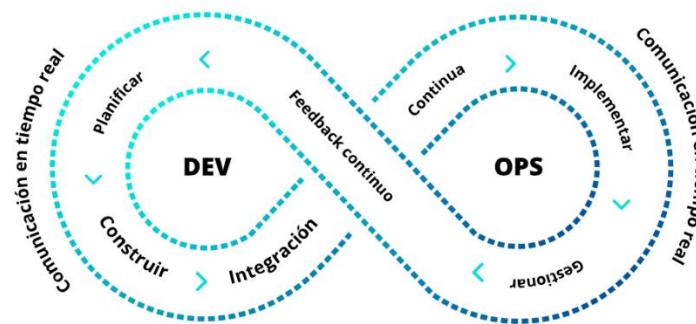


Ilustración 60. Metodología DevOps.

Características

- Ciclos rápidos de desarrollo, testing y lanzamiento.
- Integración y colaboración continua de equipos Dev y Ops.
- Infraestructura y procesos automatizados.
- Monitoreo y mejora continua.

Etapas

Planificación: Definir historias de usuario, prioridades y plan de versiones.

Desarrollo: Programación colaborativa e integración continua de código.

Pruebas: Testing automatizado a diferentes niveles.

Release: Despliegue automatizado en ambientes stage/producción.

Operaciones: Monitoreo, log analysis y tuning post-despliegue [59].

3.1.4. Consideraciones

Para garantizar el éxito del proyecto, el desarrollo de sistemas basados en IoT (Internet de las cosas) presenta una serie de consideraciones fundamentales que deben abordarse cuidadosamente. El procesamiento en la nube, la interconexión de dispositivos y la adquisición de datos son elementos cruciales que se destacan en estas consideraciones. Estos componentes no solo son necesarios para que los sistemas IoT funcionen de manera eficiente y efectiva, sino que también tienen un gran impacto en la escalabilidad, la seguridad y la capacidad de adaptarse a los cambios [60].

Adquisición de datos

La adquisición de datos es esencial en los sistemas IoT para recopilar información de los dispositivos conectados. Esto implica tanto elegir los sensores apropiados para capturar los datos necesarios como establecer protocolos de comunicación efectivos para transmitir esos datos a través de la red. Para optimizar la adquisición de datos en entornos de Internet de las cosas, también es importante pensar en cosas como la gestión de la energía, la frecuencia de muestreo y la calidad de los datos [61].

Procesamiento en la nube

Es común que los dispositivos IoT transfieran los datos a la nube para su procesamiento y análisis. Esto facilita el almacenamiento, el procesamiento en tiempo real, el análisis predictivo y la generación de informes, entre otras tareas. La escalabilidad, la seguridad, la latencia y el costo son factores importantes al seleccionar plataformas de procesamiento en la nube para sistemas IoT [62].

Interconexión de dispositivos

La interconexión de dispositivos es esencial para permitir que los diferentes componentes de un sistema Internet de las cosas se comuniquen entre sí. Para facilitar la interoperabilidad entre los dispositivos y la integración con otros sistemas, se requiere el uso de protocolos de comunicación convencionales como MQTT, CoAP o HTTP. Además, es importante tener en cuenta la topología de red y los mecanismos de seguridad apropiados para proteger la comunicación entre los dispositivos [62].

3.2. Alternativas de Solución

El desarrollo e implementación de proyectos IoT requiere la consideración cuidadosa de una variedad de opciones de solución, que abarcan desde la elección de las plataformas de hardware hasta la elección de los servicios en la nube adecuados.

3.2.1. Plataforma de hardware

El término "plataforma de hardware" se refiere a un conjunto de piezas físicas y dispositivos conectados que sirven como base para la creación y operación de sistemas electrónicos y computacionales. Estas plataformas brindan una estructura y funcionalidad básica que permite la creación de aplicaciones y soluciones personalizadas, lo que facilita la creación de dispositivos y sistemas específicos [63].

Arduino

La mayoría de las personas utilizan la plataforma de hardware de código abierto Arduino para desarrollar proyectos electrónicos y sistemas embebidos. Consta de múltiples placas de circuito impreso que integran un microcontrolador y una serie de pines de entrada/salida (E/S), lo que permite a los usuarios interactuar fácilmente con sensores, actuadores y otros dispositivos electrónicos [64].

Las placas Arduino vienen en una variedad de tamaños y formas, desde las más básicas hasta las más avanzadas, y todas vienen con funciones adicionales como conectividad Wi-Fi o Bluetooth. Además, el entorno de desarrollo integrado (IDE) de Arduino proporciona una interfaz fácil de

usar para escribir y cargar programas en las placas Arduino, utilizando un dialecto de C/C++ simplificado [63], [64].

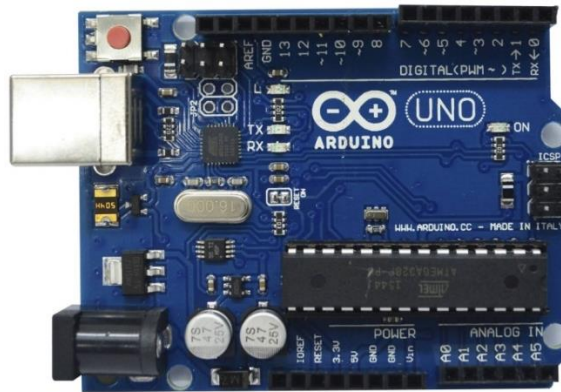


Ilustración 61. Arduino UNO R3.

Los Arduino se utilizan en una amplia gama de aplicaciones, incluyendo domótica, robótica, arte interactivo, prototipado rápido y educación.

Raspberry Pi

La Fundación Raspberry Pi creó una serie de computadoras de placa única (SBC) llamadas Raspberry Pi. Estas computadoras pequeñas y accesibles están destinadas a fomentar la enseñanza de la programación y la informática, así como a servir como una plataforma versátil para una amplia gama de proyectos de informática y electrónica [65].

Las Raspberry Pi tienen un procesador ARM, memoria RAM, puertos USB, HDMI, Ethernet y GPIO (Ingreso/Salida General Purpose), lo que les permite conectar una variedad de periféricos y dispositivos externos. Además, ejecutan sistemas operativos basados en Linux, como Raspbian (derivado de Debian), lo que brinda a los desarrolladores y usuarios un entorno familiar [65], [66].



Ilustración 62, Raspberry Pi4 2GB.

Raspberry Pi es adecuada para una amplia gama de aplicaciones, como servidores web, estaciones de trabajo, centros multimedia, proyectos de automatización del hogar y sistemas de control, debido a su versatilidad. Su popularidad y constante desarrollo se deben a su bajo costo y a su comunidad activa de usuarios y desarrolladores [65].

3.2.2. Servicios en la nube para IoT

Los servicios en la nube para Internet de las cosas (IoT) son plataformas y servicios alojados en la nube que proporcionan una infraestructura escalable y flexible para el desarrollo, implementación y gestión de sistemas y aplicaciones de Internet de las cosas (IoT) [67].



Ilustración 63. Plataformas IoT en la nube.

AWS IoT

Una plataforma de servicios en la nube desarrollada por Amazon Web Services (AWS), AWS IoT facilita la conectividad, el manejo y la interacción segura y escalable con dispositivos IoT. Esta plataforma ofrece una variedad de productos y herramientas destinados a ayudar a los desarrolladores a desarrollar y administrar de manera efectiva aplicaciones de Internet de las cosas [68].

Características

Entre las principales características que nos brinda AWS IoT tenemos:

Tabla 1. Principales características de AWS IoT [68], [69].

Aspecto	Descripción
Gestión de dispositivos	AWS IoT facilita el registro, autenticación y gestión segura de un gran número de dispositivos, simplificando su incorporación y control en aplicaciones IoT.
Comunicación y mensajería	Ofrece un servicio de mensajería basado en el protocolo MQTT (Mensaje Queuing Telemetry Transport) para establecer una comunicación bidireccional confiable y escalable entre dispositivos.
Almacenamiento y análisis	Permite la utilización de servicios como Amazon Kinesis y Amazon QuickSight para realizar análisis avanzados y almacenar grandes volúmenes de datos generados por dispositivos IoT.
Integración con AWS	Se integra con diversos servicios de AWS, incluyendo aprendizaje automático, bases de datos y seguridad, permitiendo crear soluciones IoT completamente personalizadas.
Seguridad	Proporciona características avanzadas de seguridad, incluyendo autenticación de dispositivos y encriptación de datos, para garantizar la confidencialidad y protección de la información.

Azure IoT

Una plataforma de servicios en la nube desarrollada por Microsoft llamada Azure IoT ofrece una amplia gama de herramientas y servicios que ayudan a los desarrolladores a crear, implementar y administrar soluciones basadas en el Internet de las Cosas (IoT). Para conectar, supervisar y controlar una variedad de dispositivos IoT, así como para analizar y actuar sobre los datos generados por estos dispositivos, Azure IoT proporciona una infraestructura escalable y segura [70].

Tabla 2. Principales características de Azure IoT [70].

Aspecto	Descripción
---------	-------------

Gestión de dispositivos	Azure IoT Hub facilita el registro, autenticación y seguimiento de dispositivos conectados, permitiendo la gestión centralizada de millones de dispositivos IoT.
Comunicación segura	Proporciona protocolos de comunicación seguros como MQTT, AMQP y HTTP para garantizar una comunicación segura y confiable entre dispositivos y la nube.
Análisis de datos	Las herramientas de Azure IoT, como Azure Stream Analytics y Azure Time Series Insights, permiten el análisis de datos en tiempo real y la extracción de datos útiles para la toma de decisiones.
Integración con servicios de Azure	Se integra estrechamente con otros servicios de Azure, como Azure Machine Learning, Azure Functions y Azure Cosmos DB, para brindar capacidades avanzadas de procesamiento de datos y aplicaciones IoT enriquecidas.
Seguridad	Azure IoT protege la integridad y la privacidad de los datos de IoT mediante medidas de seguridad avanzadas como la autenticación de dispositivos, la encriptación de datos y el control de acceso.

Para garantizar la eficiencia, la seguridad y la escalabilidad de los sistemas IoT, la adquisición de datos, el procesamiento en la nube y la integración de dispositivos deben abordarse de manera cuidadosa.

Además, el éxito de los proyectos de Internet de las cosas depende de la selección de metodologías de desarrollo, plataformas de servicios en la nube y hardware adecuados.

4. Desarrollo de la práctica

4.1.1. Problema planteado:

Diseño del prototipo de una fábrica en donde se simule la línea de proceso de producción de un producto de todas sus fases, que parta desde un inicio con la materia prima, sea procesado por las máquinas y que después de cierto tiempo determinado (el cual es el necesario para que la máquina transforme la materia prima en producto final) devuelva el producto final para ser empacado como pedido y que pueda ser enviado, es decir que haya cumplido con cada una de sus fases, cabe mencionar que el procedimiento se debe de realizar de forma lineal, de manera consecutiva, es decir que las actividades no se deben de realizar sin antes verificar que la materia prima haya pasado por cada una de las fases correspondientes de manera completa, de tal forma que se pueda verificar que el producto ha realizado el proceso de producción de manera completa y correcta. Para llegar a un producto terminado se necesitará de dos materias primas y un ensamblado. La línea de producción debe de verse de la siguiente manera: materia prima -> tarea 1 – tarea 2 -> almacén -> tarea 3 -> Producto. Considerando el siguiente contexto: existe una máquina por cada tarea, las máquinas solo podrán procesar una unidad al tiempo incluye un almacén entre tareas con la capacidad de uno.

4.1.2. Solución

Se implementará una metodología de tipo Pull, propio de redes de Petri, la cual nos permite pasar de una acción a otra por medio de una señal, por lo que, si no se envía la señal de terminado en un área, no se podrá continuar al siguiente proceso y se plantearán de dicha forma para verificar que el procedimiento se lleve a cabo de manera correcta y completa. Cuando un pedido se genera la información se da a través de una señal a todas las fases de trabajo. Para representar una máquina: tendrá un inicio y un fin que vendrán marcadas como “transiciones” y la tarea será representada por dos “plazas” en donde una de ellas tendrá un token indicando que la máquina está disponible para procesar materia prima.

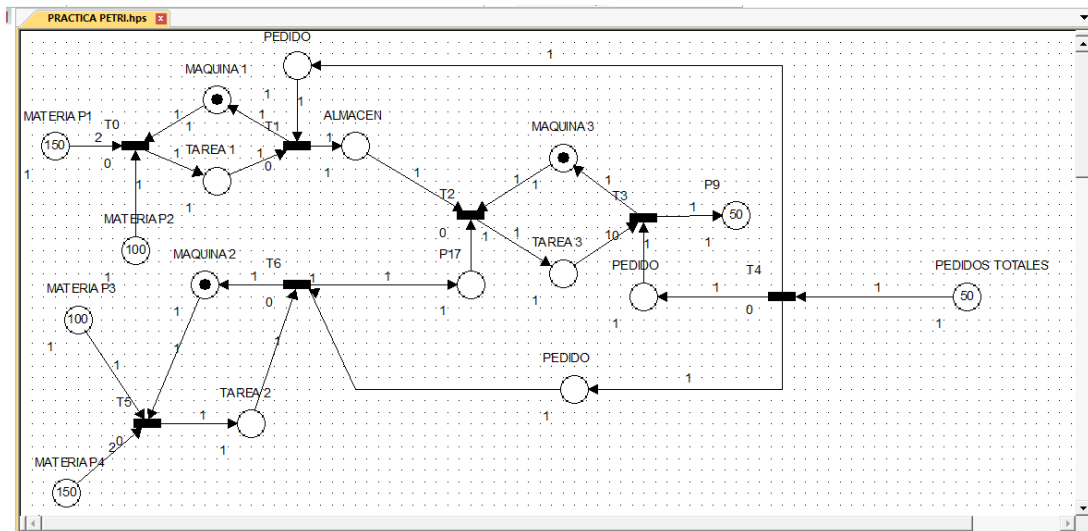


Ilustración 64. Diagrama Petri 1.

Se ha considerado comenzar con materia prima de: 150, 100, 100 y 150, existen 3 plazas con el nombre de pedido, estos cumplen con el proceso de trabajar de manera coordinada, de esta manera cada vez que se realiza un pedido se pueda fabricar un terminado, como se puede observar todos los “pedidos” se conectan en una sola hacia “pedidos totales”, una vez el pedido pasa en la transición y se reparta a las demás plazas se pueden realizar los productos terminados

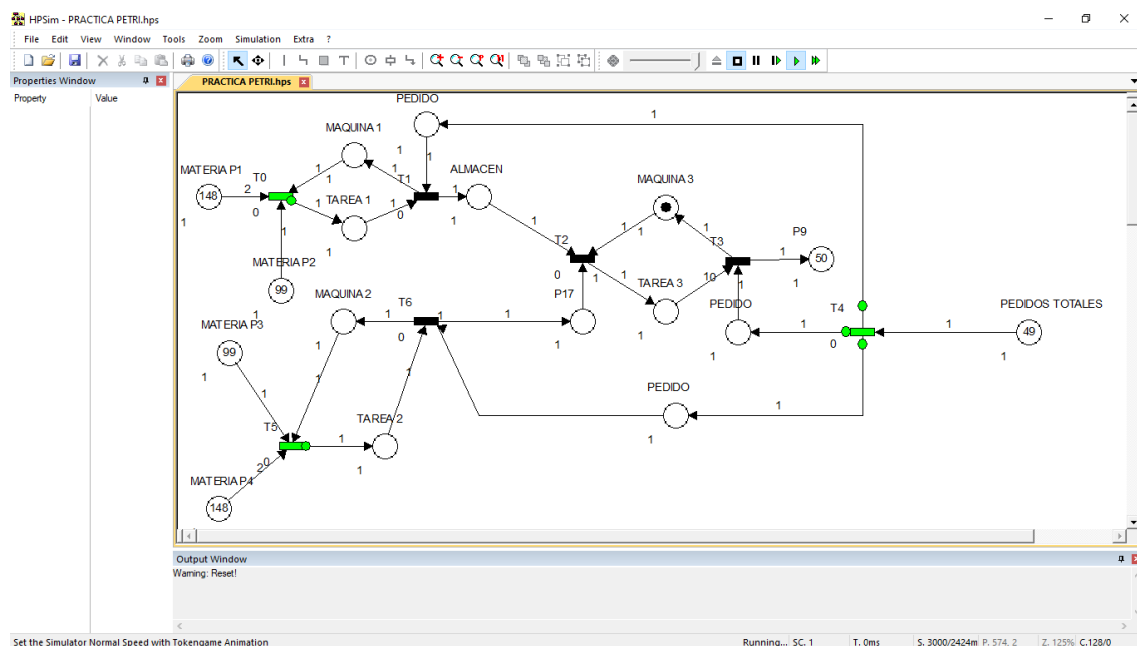


Ilustración 65. Diagrama Petri 2.

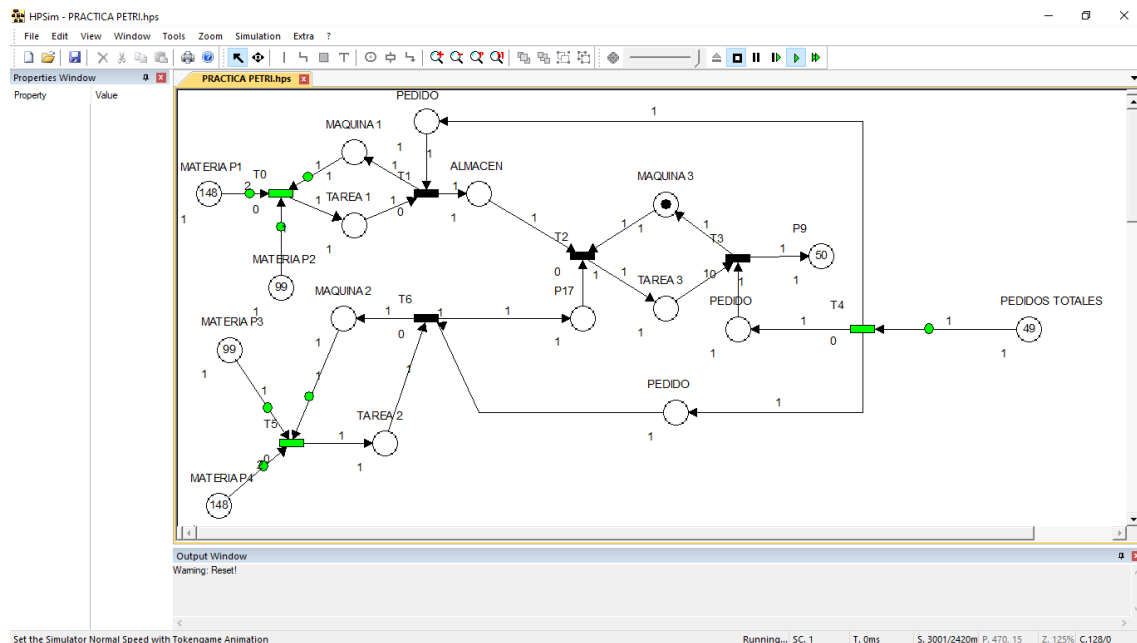


Ilustración 66. Diagrama Petri 3.

Como se puede observar, una vez que se finalizan los tres procesos o fases se envía la señal para que lleguen a la transición los 3 al mismo tiempo para que devuelva el producto finalizado y de la orden para que comience a salir la demás materia prima y se realicen las respectivas fases y así consecutivamente hasta que la plaza “pedidos totales llegue a 0”, en este caso se ha considerado trabajar con materia prima necesaria para que termine con los “pedidos totales” si fuera el caso de que la materia prima sea insuficiente para la cantidad de pedidos solicitados el proceso se realizara hasta lo que de para producir con dicha cantidad de materia prima.

5. Conclusiones

En primer lugar, examinamos técnicas de modelado como diagramas UML, BPMN y redes de Petri. Se evidenció la importancia del uso de estas herramientas para representar la estructura, el comportamiento y los procesos de un sistema distribuido. Estos diagramas facilitan la comunicación y la colaboración entre los miembros del equipo de desarrollo y nos permiten visualizar y comprender mejor la complejidad de los sistemas distribuidos.

Es importante saber elegir y aplicar las metodologías correctas para el desarrollo de sistemas IoT considerando elementos importantes como la adquisición de datos, el procesamiento en la nube y la interconexión de dispositivos. Además, examinamos una variedad de soluciones para la implementación de sistemas IoT, que incluyen el uso de plataformas y tecnologías como Arduino, Raspberry Pi y servicios en la nube de proveedores líderes como AWS IoT, Azure IoT y Google Cloud IoT.

6. Bibliografia

- [1] L. Wang et al., "On-line Simulation System of Urban Road Traffic Signal Control Based on Scene Driven," 2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS), Dali, China, 2019, pp. 1213-1218, doi: 10.1109/DDCLS.2019.8908950.
- [2] A. Shaikh, A. Hafeez, A. A. Wagan, M. Alrizq, A. Alghamdi and M. S. A. Reshan, "More Than Two Decades of Research on Verification of UML Class Models: A Systematic Literature Review," in IEEE Access, vol. 9, pp. 142461-142474, 2021, doi: 10.1109/ACCESS.2021.3121222.
- [3] D. N. H. Weerasinghe, K. A. T. Thiwanka, H. B. C. Jayasith, P. A. D. Onella Natalie, U. U. Samantha Rajapaksha and A. Karunasena, "Smart UML - Assignment Management Tool for UML Diagrams," 2022 4th International Conference on Advancements in Computing (ICAC), Colombo, Sri Lanka, 2022, pp. 114-119, doi: 10.1109/ICAC57685.2022.10025080.
- [4] E. Cariou, L. Brunschwig, O. Le Goaer and F. Barbier, "A software development process based on UML state machines," 2020 International Conference on Advanced Aspects of Software Engineering (ICAASE), Constantine, Algeria, 2020, pp. 1-8, doi: 10.1109/ICAASE51408.2020.9380117.
- [5] H. E. Jihad, M. Adedjouma and M. Morelli, "Automated Fault Tree generation in Open-PSA from UML Models," 2021 28th Asia-Pacific Software Engineering Conference (APSEC), Taipei, Taiwan, 2021, pp. 578-579, doi: 10.1109/APSEC53868.2021.00076.
- [6] P. Ardimento, L. Aversano, M. L. Bernardi, V. A. Carella, M. Cimitile and M. Scalera, "UML Miner: A Tool for Mining UML Diagrams," 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Västerås, Sweden, 2023, pp. 30-34, doi: 10.1109/MODELS-C59198.2023.00014.

- [7] D. Apostol, P. Rusovan and M. Marcu, "UML to code, and code to UML, a view inside implementation challenges and cost," 2022 26th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2022, pp. 140-145, doi: 10.1109/ICSTCC55426.2022.9931871.
- [8] D. Torre, Y. Labiche, M. Genero, M. Elaasar and C. Menghi, "UML Consistency Rules: a Case Study with Open-Source UML Models," 2020 IEEE/ACM 8th International Conference on Formal Methods in Software Engineering (FormaliSE), Seoul, Korea, Republic of, 2020, pp. 130-140.
- [9] D. Apostol, R. Bogdan and M. Marcu, "UML Diagrams in Teaching Software Engineering Classes. A Case Study In Computer Science Class," 2024 IEEE 22nd World Symposium on Applied Machine Intelligence and Informatics (SAMI), Stará Lesná, Slovakia, 2024, pp. 000327-000332, doi: 10.1109/SAMI60510.2024.10432905.
- [10] J. Farinha and A. R. da Silva, "UML Templates Distilled," in IEEE Access, vol. 10, pp. 8709-8727, 2022, doi: 10.1109/ACCESS.2022.3143898.
- [11] P. J. Escamilla-Ambrosio, D. A. Robles-Ramírez, T. Tryfonas, A. Rodríguez-Mota, G. Gallegos-García and M. Salinas-Rosales, "IoTsecM: A UML Extension for Internet of Things Security Modeling," in IEEE Access, vol. 9, pp. 15411-15413, 2021, doi: 10.1109/ACCESS.2021.3125979.
- [12] L. Carnevali, R. German, F. Santoni and E. Vicario, "Compositional Analysis of Hierarchical UML Statecharts," in IEEE Transactions on Software Engineering, vol. 48, no. 12, pp. 4762-4788, 1 Dec. 2022, doi: 10.1109/TSE.2021.3125720.
- [13] M. Alshayeb, H. Mumtaz, S. Mahmood and M. Niazi, "Improving the Security of UML Sequence Diagram Using Genetic Algorithm," in IEEE Access, vol. 8, pp. 62738-62761, 2020, doi: 10.1109/ACCESS.2020.2981742.

- [14] M. Alshayeb, H. Mumtaz, S. Mahmood and M. Niazi, "Improving the Security of UML Sequence Diagram Using Genetic Algorithm," in IEEE Access, vol. 8, pp. 62738-62761, 2020, doi: 10.1109/ACCESS.2020.2981742.
- [15] F. Sechi, B. A. Gran, P. -A. Jørgensen and O. Kilyukh, "Better Security Assessment Communication: Combining ISO 27002 Controls with UML Sequence Diagrams," 2022 IEEE/ACM 3rd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS), Pittsburgh, PA, USA, 2022, pp. 49-56, doi: 10.1145/3524489.3527304.
- [16] R. G. Tiwari, A. Pratap Srivastava, G. Bhardwaj and V. Kumar, "Exploiting UML Diagrams for Test Case Generation: A Review," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), London, United Kingdom, 2021, pp. 457-460, doi: 10.1109/ICIEM51511.2021.9445383.
- [17] Z. A. Hamza and M. Hammad, "Generating Test Sequences from UML Use Case Diagram: A Case Study," 2020 Second International Sustainability and Resilience Conference: Technology and Innovation in Building Designs(51154), Sakheer, Bahrain, 2020, pp. 1-6, doi: 10.1109/IEEECONF51154.2020.9319979.
- [18] S. W. Haga, W. -M. Ma and W. S. Chao, "Inconsistency Checking of UML Sequence Diagrams and State Machines Using the Structure-Behavior Coalescence Method," 2022 International Conference on Engineering and Emerging Technologies (ICEET), Kuala Lumpur, Malaysia, 2022, pp. 1-6, doi: 10.1109/ICEET56468.2022.10007160.
- [19] N. F. Setiyawan, Y. Priyadi and W. Astuti, "Development of Class Diagrams Based on Use Case, and Sequence Diagrams Using a Text Mining Approach in SRS Penguin," 2023 IEEE World AI IoT Congress (AllIoT),

Seattle, WA, USA, 2023, pp. 0070-0076, doi: 10.1109/AlloT58121.2023.10174287.

- [20] A. Mohan and S. Jayaraman, "Enabling Granular Exploration of Sequence Diagrams with Activity Diagrams," 2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS), Kochi, India, 2022, pp. 1-6, doi: 10.1109/IC3SIS54991.2022.9885450.
- [21] H. Yang, X. Dong, T. Wang and A. Xiao, "The Microservice Architecture of Airline's Group-CRM Based on UML," 2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE), Shenzhen, China, 2020, pp. 133-136, doi: 10.1109/AEMCSE50948.2020.00035.
- [22] F. Buzaid, F. Albaloooshi and W. Elmedany, "The Use of UML Diagrams to Enhance Dynamic Feature Location Techniques," 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), Sakheer, Bahrain, 2022, pp. 285-292, doi: 10.1109/3ICT56508.2022.9990768.
- [23] R. G. Mohammadi and A. A. Barforoush, "Enforcing component dependency in UML deployment diagram for cloud applications," 2014 7th International Symposium on Telecommunications, IST 2014, pp. 412–417, Dec. 2014, doi: 10.1109/ISTEL.2014.7000739.
- [24] C. Britton and J. Doake, "Design," in A Student Guide to Object-Oriented Development, Elsevier, 2005, pp. 221–247. doi: 10.1016/B978-075066123-2/50009-8.
- [25] T. Górski and J. Bednarski, "Transformation of the UML Deployment Model into a Distributed Ledger Network Configuration," 2020 IEEE 15th International Conference of System of Systems Engineering (SoSE), Budapest, Hungary, 2020, pp. 255-260, doi: 10.1109/SoSE50414.2020.9130492.

- [26] A. R. Viesca and M. Al Lail, "Automated Mitigation of Frame Problem in UML Class Diagram Verification," 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Västerås, Sweden, 2023, pp. 841-850, doi: 10.1109/MODELS-C59198.2023.00133.
- [27] H. Liu et al., "A Fault Injection and Formal Verification Framework Based on UML Sequence Diagrams," 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), Florence, Italy, 2023, pp. 45-50, doi: 10.1109/ISSREW60843.2023.00045.
- [28] A. Pişirgen and S. Peker, "A UML-Based Conceptual Model for Appointment Booking Systems," 2021 6th International Conference on Computer Science and Engineering (UBMK), Ankara, Turkey, 2021, pp. 812-817, doi: 10.1109/UBMK52708.2021.9558929.
- [29] P. Jain and D. Soni, "A Survey on Generation of Test Cases using UML Diagrams," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-6, doi: 10.1109/ic-ETITE47903.2020.395.
- [30] X. Li and X. Wu, "Code Generation from UML State Chart Diagrams," 2022 International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS), Wuhan, China, 2022, pp. 15-20, doi: 10.1109/ICMSS55574.2022.00010.
- [31] D. Shrestha, T. Wenan, S. Maharjan, B. Gaudel, J. Chun and S. R. Jeong, "A UML based approach for analysis and design of tourism web portal," 2020 International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2020, pp. 236-243, doi: 10.1109/ICOSEC49089.2020.9215380.

- [32] A. Kunnappilly, P. Backeman and C. Seceleanu, "UML-based Modeling and Analysis of 5G Service Orchestration," 2020 27th Asia-Pacific Software Engineering Conference (APSEC), Singapore, Singapore, 2020, pp. 129-138, doi: 10.1109/APSEC51365.2020.00021.
- [33] A. Souri, M. A. Sharifloo, y M. Norouzi, «Formalizing class diagram in UML», 2011 IEEE 2nd International Conference On Software Engineering And Service Science, jul. 2011, doi: 10.1109/icsess.2011.5982368.
- [34] D. Faitelson y S. Tyszberowicz, «UML Diagram Refinement (Focusing on Class-and Use Case Diagrams)», 2017 IEEE/ACM 39th International Conference On Software Engineering (ICSE), may 2017, doi: 10.1109/icse.2017.73.
- [35] P. Ardimento, L. Aversano, M. L. Bernardi, V. A. Carella, M. Cimitile, y M. Scalera, «UML Miner: A Tool for Mining UML Diagrams», ACM/IEEE International Conference On Model Driven Engineering Languages And Systems Companion (MODELS-C), oct. 2023, doi: 10.1109/models-c59198.2023.00014.
- [36] K.-H. Doan y M. Gogolla, «Quality Improvement for UML and OCL Models Through Bad Smell and Metrics Definition», ACM/IEEE 22nd International Conference On Model Driven Engineering Languages And Systems Companion (MODELS-C), sep. 2019, doi: 10.1109/models-c.2019.00121.
- [37] P. Muñoz, J. Troya, y A. Vallecillo, «Using UML and OCL Models to Realize High-Level Digital Twins», 2021 ACM/IEEE International Conference On Model Driven Engineering Languages And Systems Companion (MODELS-C), oct. 2021, doi: 10.1109/models-c53483.2021.00037.
- [38] C. Vidal-Silva y R. Villarroel, «JPI UML: UML Class and Sequence Diagrams Proposal for Aspect-Oriented JPI Applications», 33rd

International Conference Of The Chilean Computer Science Society (SCCC), nov. 2014, doi: 10.1109/sccc.2014.23.

- [39] S. Kaneda y I. Akio, «Understanding of Class Diagrams Based on Cognitive Linguistics and Functional Dependency», IIAI 3rd International Conference On Advanced Applied Informatics, ago. 2014, doi: 10.1109/iaai-aai.2014.126.
- [40] A. Souri, M. A. Sharifloo, y M. Norouzi, «Formalizing class diagram in UML», IEEE 2nd International Conference On Software Engineering And Service Science, jul. 2011, doi: 10.1109/icsess.2011.5982368.
- [41] L. Carnevali, R. German, F. Santoni and E. Vicario, "Compositional Analysis of Hierarchical UML Statecharts," in IEEE Transactions on Software Engineering, vol. 48, no. 12, pp. 4762-4788, 1 Dec. 2022, doi: 10.1109/TSE.2021.3125720.
- [42] T. Baar, A. Strohmeier, A. Moreira, y S. J. Mellor, < <UML> > 2004 - The Unified Modeling Language. Modelling Languages and Applications. 2004. doi: 10.1007/b101232.
- [43] H. Ishikawa, "A Case Study of Refactoring with UML Editor Plug-in for Eclipse – Replace Type Code with State/Strategy –, " 2020 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Nagoya, Japan, 2020, pp. 49-53.
- [44] L. Holý, K. Ježek, J. Šnajberk, y P. Brada, «Lowering Visual Clutter in Large Component Diagrams», 2012 16th International Conference On Information Visualisation, jul. 2012, doi: 10.1109/iv.2012.17.
- [45] B. Nadira, C. Bouanaka, M. Bendjaballah and A. Djarri, "Towards an UML-based SoS Analysis and Design Process," 2020 International Conference on Advanced Aspects of Software Engineering (ICAASE), Constantine, Algeria, 2020, pp. 1-8, doi: 10.1109/ICAASE51408.2020.9380112.

- [46] D. Suárez, H. Posadas and V. Fernández, "UML-Based Design Flow for Systems with Neural Networks," 2023 38th Conference on Design of Circuits and Integrated Systems (DCIS), Málaga, Spain, 2023, pp. 1-6, doi: 10.1109/DCIS58620.2023.10335992.
- [47] A. R. Mohammed and S. S. Kassem, "UML Modeling of Online Public Bus Reservation System in Egypt," 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI), Sakheer, Bahrain, 2020, pp. 1-6, doi: 10.1109/ICDABI51230.2020.9325604.
- [48] J. Hagar and M. -F. Wendland, "Defining Software Test Architectures with the UML Testing Profile," 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Dublin, Ireland, 2023, pp. 271-280, doi: 10.1109/ICSTW58534.2023.00056.
- [49] A. McEwen y H. Cassimally, *Designing the internet of things*. Nashville, TN, Estados Unidos de América: John Wiley & Sons, 2013.
- [50] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, y J. Henry, *IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things*. Indianápolis, IN, Estados Unidos de América: Cisco Press, 2016.
- [51] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6. Elsevier Inc., pp. 1213–1221, 2012. doi: 10.1016/j.jss.2012.02.033.
- [52] J. Sutherland, *Scrum: The art of doing twice the work in half the time*. Currency, 2014.

- [53] G. Watts, *Scrum mastery: From good to great servant leadership*. Cheltenham, Inglaterra: Inspect & Adapt, 2013.
- [54] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, IEEE Computer Society, 2013, pp. 9–16. doi: 10.1109/SEAA.2013.28.
- [55] I. Sommerville, *Software Engineering: United States Edition*, 9a ed. Upper Saddle River, NJ, Estados Unidos de América: Pearson, 2010.
- [56] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7a ed. Nueva York, NY, Estados Unidos de América: McGraw-Hill Professional, 2009.
- [57] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Softw*, vol. 33, no. 3, pp. 94–100, May 2016, doi: 10.1109/MS.2016.68.
- [58] A. Mishra and Z. Otaiwi, "DevOps and software quality: A systematic mapping," *Computer Science Review*, vol. 38. Elsevier Ireland Ltd, Nov. 01, 2020. doi: 10.1016/j.cosrev.2020.100308.
- [59] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet Things J*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.
- [60] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of Internet-of-Things platforms," *Comput Commun*, vol. 89–90, pp. 5–16, Sep. 2016, doi: 10.1016/j.comcom.2016.03.015.

- [61] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, y J. Henry, IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things. Indianápolis, IN, Estados Unidos de América: Cisco Press, 2016.
- [62] M. Banzi y M. Shiloh, Getting started with arduino: The open source electronics prototyping platform, 3a ed. Sebastopol, CA, Estados Unidos de América: Maker Media, 2015.
- [63] M. Margolis, Arduino Cookbook. Sebastopol, CA, Estados Unidos de América: O'Reilly Media, 2011.
- [64] Simon, Raspberry pi cookbook, 4E: Software and hardware problems and solutions, 4a ed. Sebastopol, CA, Estados Unidos de América: O'Reilly Media, 2022.
- [65] D. Molloy, Exploring raspberry pi: Interfacing to the real world with embedded Linux. Nashville, TN, Estados Unidos de América: John Wiley & Sons, 2016.
- [66] M. Kranz, Building the internet of things: Implement new business models, disrupt competitors, transform your industry. Nashville, TN, Estados Unidos de América: John Wiley & Sons, 2016.
- [67] A. Tali, A. Bouslama, M. Eddabbah, y Y. Laaziz, "AWS and IoT for real-time remote medical monitoring", Int. J. Intell. Enterp., vol. 6, núm. 2/3/4, p. 369, 2019.

- [68] J. Yang, A. Sharma, and R. Kumar, "IoT-based framework for smart agriculture," *International Journal of Agricultural and Environmental Information Systems*, vol. 12, no. 2, pp. 1–14, Apr. 2021, doi: 10.4018/IJAEIS.20210401.0a1.
- [69] [13] H. R. Trabelsi y M.-A. Laniel, *A Developer's Guide to Cloud Apps Using Microsoft Azure A Developer's Guide to Cloud Apps Using Microsoft Azure: Migrate and modernize your cloud-native applications with containers on Azure using real-world case studies*. Birmingham, Inglaterra: Packt Publishing, 2023.
- [70] S. Klein, *IoT Solutions in Microsoft's Azure IoT Suite: Data acquisition and analysis in the real world*. Nueva York, NY, Estados Unidos de América: APRESS, 2017.
- [71] M. Von Rosing, S. A. White, F. Cummins, and H. De Man, "Business process model and notation-BPMN," *Comple. Bus. Process Handb. Body Knowl. from Process Model. to BPM*, vol. 1, no. January, pp. 429–453, 2014, doi: 10.1016/B978-0-12-799959-3.00021-5.
- [72] E. Garcia-Maldonado, A. Cristóbal-Salas, y B. Santiago-Vicente, «Interactive BPMN Diagrams for Developing Under Scrum and DevOps», 2018 6th International Conference In Software Engineering Research And Innovation (CONISOFT), oct. 2018, doi: 10.1109/conisoft.2018.8645910.
- [73] P. Von Olberg y L. Strey, «Approach to Generating Functional Test Cases from BPMN Process Diagrams», 2022 IEEE 30th International Requirements Engineering Conference Workshops (REW), ago. 2022, doi: 10.1109/rew56159.2022.00042.
- [74] B. Hitpass y J. C. Leiva, «Modeling a Retailer B2B Integration Portal as Choreography Intermediary with BPMN 2.0 Choreography Diagrams»,

2012 31st International Conference Of The Chilean Computer Science Society, nov. 2012, doi: 10.1109/sccc.2012.36.

- [75] B. Hitpass y J. C. Leiva, «Modeling a Retailer B2B Integration Portal as Choreography Intermediary with BPMN 2.0 Choreography Diagrams», 2012 31st International Conference Of The Chilean Computer Science Society, nov. 2012, doi: 10.1109/sccc.2012.36.
- [76] U. I. Hernandez, F. J. Á. Rodríguez, y M. V. Martín, «Use processes — modeling requirements based on elements of BPMN and UML Use Case Diagrams», 2010 2nd International Conference On Software Technology And Engineering, oct. 2010, doi: 10.1109/icste.2010.5608758.
- [77] D. Gagné y A. Trudel, «Time-BPMN», 2009 IEEE Conference On Commerce And Enterprise Computing, jul. 2009, doi: 10.1109/cec.2009.71.
- [78] M. Chinosi y A. Trombetta, «Modeling and Validating BPMN Diagrams», 2009 IEEE Conference On Commerce And Enterprise Computing, jul. 2009, doi: 10.1109/cec.2009.48.
- [79] I. Abouzid, Y. K. Bekali, y R. Saidi, «Modelling IoT Behaviour in Supply Chain Business Processes with BPMN: A Systematic Literature Review», Journal Of ICT Standardisation, ago. 2022, doi: 10.13052/jicts2245-800x.1035.
- [80] C. Dechsupa, W. Vatanawood, y A. Thongtak, «Hierarchical Verification for the BPMN Design Model Using State Space Analysis», IEEE Access, vol. 7, pp. 16795-16815, ene. 2019, doi: 10.1109/access.2019.2892958.
- [81] U. Campos, A. Lopes, y T. Conte, «Evolution of BPMN Models through e-VOL BPMN», SBQS '20: Proceedings Of The XIX Brazilian Symposium On Software Quality, dic. 2020, doi: 10.1145/3439961.3439983.

- [82] F. Hasić, E. Serral, y M. Snoeck, «Comparing BPMN to BPMN + DMN for IoT process modelling», SAC '20: Proceedings Of The 35th Annual ACM Symposium On Applied Computing, mar. 2020, doi: 10.1145/3341105.3373881.
- [83] M. Kurz, «BPMN Model Interchange», S-BPM '16: Proceedings Of The 8th International Conference On Subject-oriented Business Process Management, abr. 2016, doi: 10.1145/2882879.2882886.
- [84] P. Polak, «Improving C2G Relation Using BPMN Diagrams», ICEEG '19: Proceedings Of The 3rd International Conference On E-commerce, E-Business And E-Governme, ene. 2019, doi: 10.1145/3340017.3340020.
- [85] D. W. Sorgatto, D. M. B. Paiva, y M. I. Cagnin, «How to elicit and specify software requirements from BPMN diagrams?», SBSI '18: Proceedings Of The XIV Brazilian Symposium On Information Systems, jun. 2018, doi: 10.1145/3229345.3229403.
- [86] A. Rachdi, A. En-Nouaary, y M. Dahchour, «Short Paper: BPMN Process Analysis: A Formal Validation and Verification Eclipse Plugin for BPMN Process Models», en Lecture Notes in Computer Science, 2019, pp. 100-104. doi: 10.1007/978-3-030-05529-5_7.
- [87] P. y. H. Wong y J. Gibbons, «A Process Semantics for BPMN», en Lecture Notes in Computer Science, 2008, pp. 355-374. doi: 10.1007/978-3-540-88194-0_22.
- [88] G. Aagesen y J. Krogstie, «BPMN 2.0 for Modeling Business Processes», en Springer eBooks, 2014, pp. 219-250. doi: 10.1007/978-3-642-45100-3_10.
- [89] R. Braun y W. Esswein, «Towards Multi-perspective Modeling with BPMN», en Lecture notes in business information processing, 2015, pp. 67-81. doi: 10.1007/978-3-319-19297-0_5.

- [90] F. Kossak et al., «A Rigorous Semantics for BPMN 2.0 Process Diagrams», en Springer eBooks, 2014, pp. 29-152. doi: 10.1007/978-3-319-09931-6_4.
- [91] M. A. Habri, R. Esbai, y Y. L. el M. Nadori, «BPMN to UML Class Diagram Using QVT», en Smart innovation, systems and technologies, 2021, pp. 593-602. doi: 10.1007/978-981-16-3637-0_42.
- [92] W. Reisig, Understanding Petri nets: Modeling techniques, analysis methods, case studies, vol. 9783642332784. Springer-Verlag Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-33278-4.
- [93] D. Palko et al., "Cyber Security Risk Modeling in Distributed Information Systems," Applied Sciences, vol. 13, no. 4, p. 2393, Feb. 2023, doi: 10.3390/app13042393.
- [94] J. Hao, Y. Yang, C. Xu, and X. Du, "A comprehensive review of planning, modeling, optimization, and control of distributed energy systems," Carbon Neutrality, vol. 1, no. 1, p. 28, Dec. 2022, doi: 10.1007/s43979-022-00029-1.
- [95] Y. Zeng, B. Chen, P. Pan, K. Li, and G. Chen, "Modeling the Training Iteration Time for Heterogeneous Distributed Deep Learning Systems," International Journal of Intelligent Systems, vol. 2023, pp. 1–15, Feb. 2023, doi: 10.1155/2023/2663115.
- [96] W. B. Daszczuk, "Modeling and Verification of Asynchronous Systems Using Timed Integrated Model of Distributed Systems," Sensors, vol. 22, no. 3, p. 1157, Feb. 2022, doi: 10.3390/s22031157.
- [97] D. Xin and L. Shi, "Trajectory Modeling by Distributed Gaussian Processes in Multiagent Systems," Sensors, vol. 22, no. 20, p. 7887, Oct. 2022, doi: 10.3390/s22207887.

- [98] D. Clavel, C. Mahulea, and M. Silva, "On Liveness Enforcement of Distributed Petri Net Systems," *IEEE Trans Automat Contr*, vol. 68, no. 6, pp. 3776–3782, Jun. 2023, doi: 10.1109/TAC.2022.3198328.
- [99] O.-G. DUCA, E. MINCA, M.-A. PAUN, I. V. GURGU, O. E. DRAGOMIR, and C. BIDICA, "PETRI NET MODELING OF A PRODUCTION SYSTEM WITH PARALLEL MANUFACTURING PROCESSES," *Journal of Science and Arts*, vol. 23, no. 1, pp. 305–318, Mar. 2023, doi: 10.46939/J.Sci.Arts-23.1-c05.
- [100] M. Djahafi and N. Salmi, "Using Stochastic Petri net modeling for self-adapting Publish/Subscribe IoT systems," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, IEEE, May 2023, pp. 1–4. doi: 10.1109/NOMS56928.2023.10154382.
- [101] L. Wells, "Performance analysis using CPN tools," in *Proceedings of the 1st international conference on Performance evaluation methodologies and tools - valuetools '06*, New York, New York, USA: ACM Press, 2006, p. 59. doi: 10.1145/1190095.1190171.
- [102] N. Kokash, C. Krause, and E. de Vink, "Reo + mCRL2 : A framework for modelchecking dataflow in service compositions," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 187–216, Mar. 2012, doi: 10.1007/s00165-011-0191-6.
- [103] M. A. Riwanto and W. N. Budiarti, "Development of Digital Science Comics for Elementary School as a Support for Digital Literacy in Online Learning," in *Proceedings of the 4th International Conference on Learning Innovation and Quality Education*, New York, NY, USA: ACM, Sep. 2020, pp. 1–4. doi: 10.1145/3452144.3452221.
- [104] A. Arcoverde, G. Alves, and R. Lima, "Petri nets tools integration through Eclipse," in *Proceedings of the 2005 OOPSLA workshop on Eclipse*

technology eXchange - eclipse '05, New York, New York, USA: ACM Press, 2005, pp. 90–94. doi: 10.1145/1117696.1117715.

- [105] M. Jurikovič, P. Čičák, and K. Jelemenská, “Parallel controller design and synthesis,” in Proceedings of the 7th FPGAWorld Conference, New York, NY, USA: ACM, Sep. 2010, pp. 35–40. doi: 10.1145/1975482.1975486.
- [106] J. D. F. de la Hoz, J. S. Sosa, and N. Cardozo, “Distributed Context Petri Nets,” in Proceedings of the Workshop on Context-oriented Programming - COP '19, New York, New York, USA: ACM Press, 2019, pp. 24–31. doi: 10.1145/3340671.3343359.
- [107] A. Bouillard and B. Gaujal, “Backward coupling in petri nets,” in Proceedings of the 1st international conference on Performance evaluation methodolgies and tools - valuetools '06, New York, New York, USA: ACM Press, 2006, p. 33. doi: 10.1145/1190095.1190137.
- [108] E. Tavares, P. Maciel, B. Silva, and M. Oliveira, “A time petri net-based approach for hard real-time systems scheduling considering dynamic voltage scaling, overheads, precedence and exclusion relations,” in Proceedings of the 20th annual conference on Integrated circuits and systems design, New York, NY, USA: ACM, Sep. 2007, pp. 312–317. doi: 10.1145/1284480.1284563.
- [109] W. Qinghao and Y. Dengkai, “A New Method for Evaluating Air Traffic Control Safety,” in Proceedings of the 2017 VI International Conference on Network, Communication and Computing, New York, NY, USA: ACM, Dec. 2017, pp. 217–221. doi: 10.1145/3171592.3171640.
- [110] B. Farwer and M. Leuschel, “Model checking object petri nets in prolog,” in Proceedings of the 6th ACM SIGPLAN international conference on Principles and practice of declarative programming, New York, NY, USA: ACM, Aug. 2004, pp. 20–31. doi: 10.1145/1013963.1013970.

- [111] E. Gianniti, A. M. Rizzi, E. Barbierato, M. Gribaudo, and D. Ardagna, "Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, no. 4, pp. 23–36, May 2017, doi: 10.1145/3092819.3092824.

- [112] V. Lesi, Z. Jakovljevic, and M. Pajic, "Reliable industrial IoT-based distributed automation," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, New York, NY, USA: ACM, Apr. 2019, pp. 94–105. doi: 10.1145/3302505.3310072.

- [113] M. D. Petty et al., "Modeling cyberattacks with extended Petri nets," in *Proceedings of the ACM Southeast Conference*, New York, NY, USA: ACM, Apr. 2022, pp. 67–73. doi: 10.1145/3476883.3520209.