UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de software para el comisionamiento virtual de instalaciones y máquinas complejas

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Oscar Villafán Tapia

ASESOR DE INFORME

Dr. Hoover Mujica Ortega

Ciudad Universitaria, Cd. Mx., 2025

Jurado asignado

Presidente:	M.F. Gabriel Hurtado Chong
Secretario:	M.I. María del Socorro Guevara Rodríguez
Vocal:	Dr. Hoover Mujica Ortega
1^{er} suplente:	Ing. David Quintanar Villalba
2^{do} suplente:	M.I. Carlos Alberto Pérez Juárez

Ciudad Universitaria, Departamento de Control y Robótica, Laboratorio de Automatización.

Ciudad de México.

Asesor de informe

Dr. Hoover Mujica Ortega

Dedicatoria

A mis padres Marcela y Fernando, porque gracias a su apoyo y amor, siempre incondicionales, hoy puedo culminar esta etapa.
A mi hermana María Fernanda por ser mi confidente y amiga, y porque con su cariño, me motiva a ser mejor cada día.
A mis abuelos, porque con su ejemplo, enseñanzas y anécdotas me mostraron la persona que quería llegar a ser.

Agradecimientos

A la Universidad Nacional Autónoma de México, por todo lo aprendido tanto dentro como fuera de sus aulas, por todas las experiencias vividas en estos años, pero sobre todo por brindarme las herramientas necesarias para lograr una formación profesional integral.

A la Facultad de Ingeniería, por permitirme aprender de la mano de grandes profesores y por darme la oportunidad de utilizar sus excelentes instalaciones.

A mi profesor y asesor de tesis, el Dr. Hoover Mujica Ortega, por haber compartido su experiencia y conocimientos, por la amistad, por sus acertadas observaciones y por su enorme compromiso, gracias a los cuales pudo culminarse este trabajo.

A todos los miembros del Laboratorio de Automatización, los cuales han sido para mi grandes mentores y amigos.

A Pedro, Eduardo, Miguel y Vicente, por tantos años de amistad, por los consejos brindados y por poder contar con ellos en cualquier circunstancia.

A mis compañeros de trabajo por la buena disposición mostrada al responder mis dudas y por sus valiosos comentarios.

A la Dirección General de Asuntos del Personal Académico (DGAPA) de la UNAM por el apoyo y la beca otorgada en el Proyecto UNAM-PAPIME PE109121.

Resumen

En este trabajo, se describe la estructura de un software utilizado en el comisionamiento virtual de proyectos industriales de automatización, el cual cuenta con diversas herramientas digitales para la construcción de estaciones o plantas virtuales que permiten simular el comportamiento real de estas basándose en el uso de gemelos digitales. Con este enfoque se consigue que las puestas en marcha consuman menos tiempo y recursos, sobre todo económicos, pues los errores de diseño, las fallas debidas a una incorrecta estimación o los problemas con la compatibilidad de equipos pueden ser observados desde una etapa temprana del desarrollo y corregidos una vez encontrados.

El concepto de Gemelo Digital data de 2003, introducido en la Universidad de Michigan por el Dr. Michael Grieves, y es ampliamente utilizado en diversos sectores, tanto industriales como científicos y socioeconómicos. Forma parte importante del paradigma tecnológico actual llamado la Industria 5.0, basado en las tecnologías habilitadoras y la economía circular, que defienden un esquema económico y tecnológico en el que los productos y procesos industriales sean sustentables, utilicen energías limpias, sean responsables con los desechos que generan e involucren asertivamente el quehacer humano.

El software de simulación tiene una estructura modular, que lo hace bastante flexible y le permite adecuarse fácilmente a los requerimientos de diversos sectores de la industria, por ejemplo, automotriz, de logística, de la construcción, entre muchos otros. Cuenta con dos aplicaciones principales y una gran variedad de módulos que otorgan herramientas para el comisionamiento virtual. Existen módulos con propósitos generales, pero incluso esto pueden ser creados por los usuarios del software según sus necesidades y ensamblados con el núcleo de este.

Debido a la complejidad del software y de los proyectos que con él se crean, es preciso identificar y reparar las fallas que puedan surgir durante su uso. De igual forma, es necesario implementar nuevas soluciones, tanto solicitadas por los usuarios como las planeadas por el equipo de trabajo, también, es importante brindar una interfaz de usuario intuitiva y consistente y, además, evaluar constantemente que las correcciones y nuevas implementaciones funcionen correctamente.

Existen muchas aplicaciones del software, en la industria automotriz se usa en la producción de componentes, la construcción de chasis, pintado y ensamblado; en el área de logística brinda herramientas como bandas transportadoras, depósitos automáticos y sistemas de transporte automáticos; en ingeniería mecánica permite detectar las colisiones de maquinaria; mientras que en área de manufactura ayuda a que el *hardware* para automatización funcione sin errores y que el software de controladores y robots este correctamente programado. Para mostrar las aplicaciones del software de simulación, se describe un caso de uso en el transporte y ensamblado de conexiones para bancos de baterías usadas en autos eléctricos de una importante empresa automotriz.

Índice general

Índice de figuras

Acrónimos

PLC Programmable Logic Controller. xiii, 3, 6, 13, 14, 15, 25, 26, 29, 31, 32, 39, 43, 45
HMI Human Machine Interface. 1
IoT Internet of Things. 2
AI Artificial Intelligence. 2, 4
CPPS Cyber-Physical Production Systems. 2, 25
CAD Computer-Aided Design. 6, 9, 29, 31, 41, 42
TCP Transmission Control Protocol. 6, 32, 33, 43
PCR Point Cloud Representation. 9
XML eXtensible Markup Language. 10, 20, 35, 36, 41
OOP Object-Oriented Programming. 15, 16, 21
SoC Separation of Concerns. 18
Protobuf Protocol Buffer. 20
RPC Remote Procedure Call. 20, 33
ECS Entity Component System. 21, 34
DDD Data Driven Design. 21
GPU Graphics Processing Unit. 22
PBR Physically Based Rendering. 23
BRDF Bidirectional Reflectance Distribution Function. 23
XAML eXtensible Application Markup Language. 27, 35
SDK Software Development Kit. 29, 30
API Application Programming Interface. 30, 31, 33
VR Virtual Reality. 31
NC Numerical Control. 40

Capítulo 1

Introducción

En la actualidad, las empresas encuentran cada vez más difícil satisfacer las necesidades de sus clientes, pues ahora se les demanda entregar productos y soluciones tecnológicas inteligentes, interconectadas, amigables con el medio ambiente y adaptables a los contextos específicos donde son requeridas. Además de esto, las regulaciones que algunos países les han impuesto para poder comercializar sus productos son cada vez más estrictas. Por lo tanto, las compañías se han visto en la necesidad de implementar sistemas de producción que puedan lidiar con esta creciente complejidad [CIMdata, 2021].

Como consecuencia de estas nuevas condiciones, existe un enorme interés por migrar al plano digital todos aquellos elementos que puedan servir como fuentes de información para una operación óptima y eficiente. Se desea que exista un flujo de información entre el mundo físico y el digital, en el que este último permita procesar de forma inteligente los datos provenientes de los sistemas físicos y, de esta manera, tomar decisiones pertinentes para reducir el consumo de energía y recursos materiales, teniendo como consecuencia que el producto final sea ecológica y económicamente viable. Esto genera, a su vez, la necesidad de contar con tecnología distribuida en la que cualquier miembro dentro de la empresa pueda tener acceso a los datos, aplicaciones y software [Explainers, 2023].

Lo anterior requiere de un cambio profundo en la estructura y organización de las empresas, repensando cuales son las prioridades dentro de sus procesos y en el manejo de sus recursos. A esto se le conoce como transformación digital, y si bien se apuntala sobre recursos digitales y software especializados, no son las tendencias tecnológicas por si solas los que determinan estas mejoras, es en cambio, la reelaboración de estrategias y procesos de la empresa considerando todas las relaciones, tanto internas (entre los miembros de la empresa) como externas (con proveedores y clientes) [Mauricio y Alberto, 2022].

Existen diversas tendencias tecnológicas que se han desarrollado en los últimos años conocidas como tecnologías disruptivas, gracias a las cuales ha sido posible implementar la antes mencionada transformación digital (Figura 1.1). Con ellas, se dispone de grandes cantidades de datos provenientes de las diferentes etapas y componentes de los procesos industriales. Apoyándose de sensores, Interfaz Humano-Máquina (HMI, por sus siglas en inglés *Human Machine Interface*) y sistemas de supervisión; en combinación con el aumento en el poder computacional de los equipos, redes de comunicación de largo alcance y bajo consumo energético, nuevas formas de interacción humano-máquina (realidad aumentada), software para simulación y Analytical and Business Intelligence permiten desarrollar

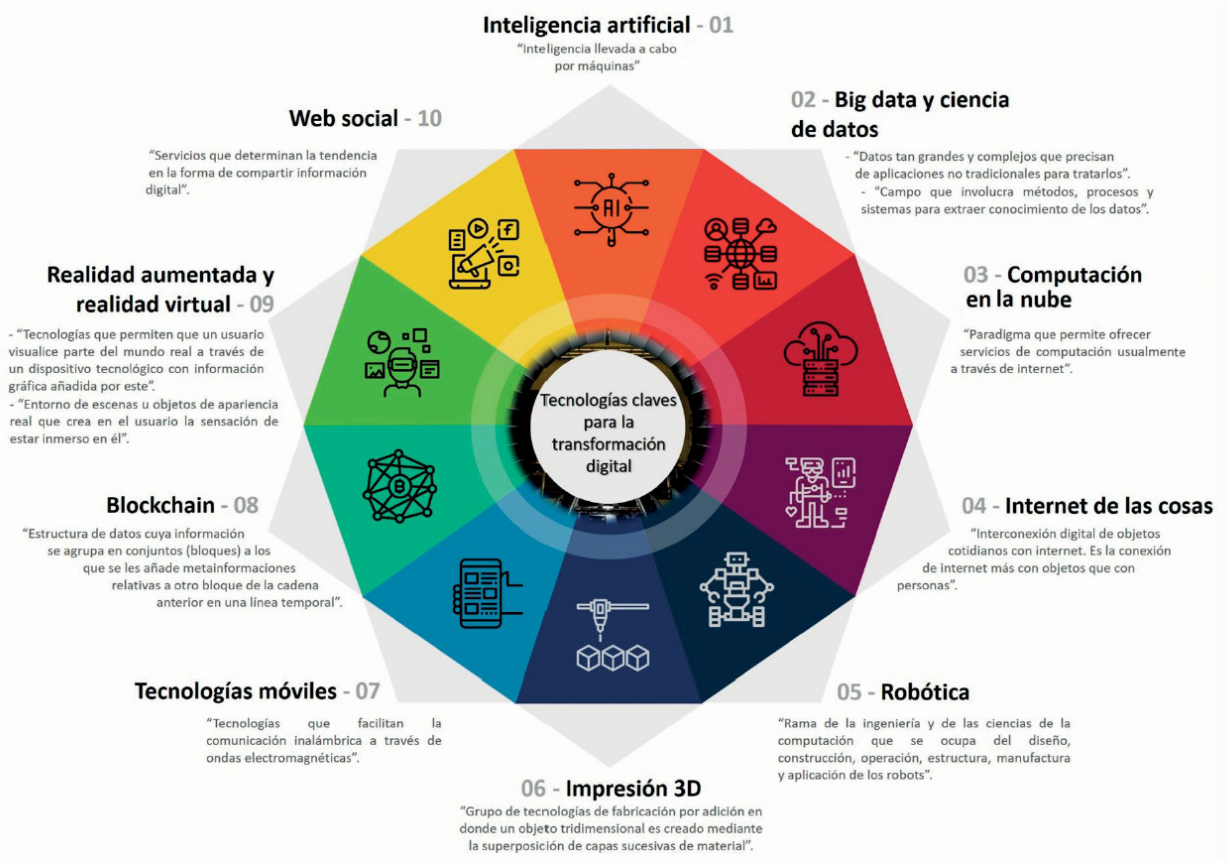

Figura 1.1 Tecnologías clave para la transformación digital.[1]

sistemas que son cada vez más inteligentes, autónomos y eficientes. [Baur y Wee, 2015]

En resumen, se desea construir sistemas cuyos componentes mantengan un alto grado de intercomunicación entre ellos, que brinden datos que puedan ser procesados y cargados en modelos digitales complejos, y que a su vez, dicha información pueda retroalimentar el control de los procesos industriales.

Para poder cubrir estas necesidades se han desarrollado tecnologías como el Internet de las cosas (IoT, por sus siglas en inglés *Internet of Things*), la Interligencia Artificial (AI, por sus siglas en inglés *Artificial Intelligence*), la robótica, los software para simulación, la computación en la nube, entre otros, que pueden ser identificados con el nombre de *Cyber-Physical Production Systems* (CPPS). Estas nuevas áreas de conocimiento representan el siguiente paso en el desarrollo de la llamada Manufactura Inteligente, la cual posibilita una mayor adaptabilidad y "autoconciencia" de los sistemas, permitiendo un mejor control de calidad y poder de predicción del comportamiento de las variables involucradas en los procesos, mejor control operacional y un monitoreo más detallado de las condiciones del sistema. [Möller*, et al.*, 2022]

Si bien es cierto que algunas de estas tecnologías ya se encuentran actualmente disponibles en muchos ámbitos de la industria, también es verdad que se requieren ciertas condiciones, muchas veces difíciles de alcanzar, para implementarlas en una mayor escala. Uno de los principales requisitos es

[1]Tomada de [Mauricio y Alberto, 2022].

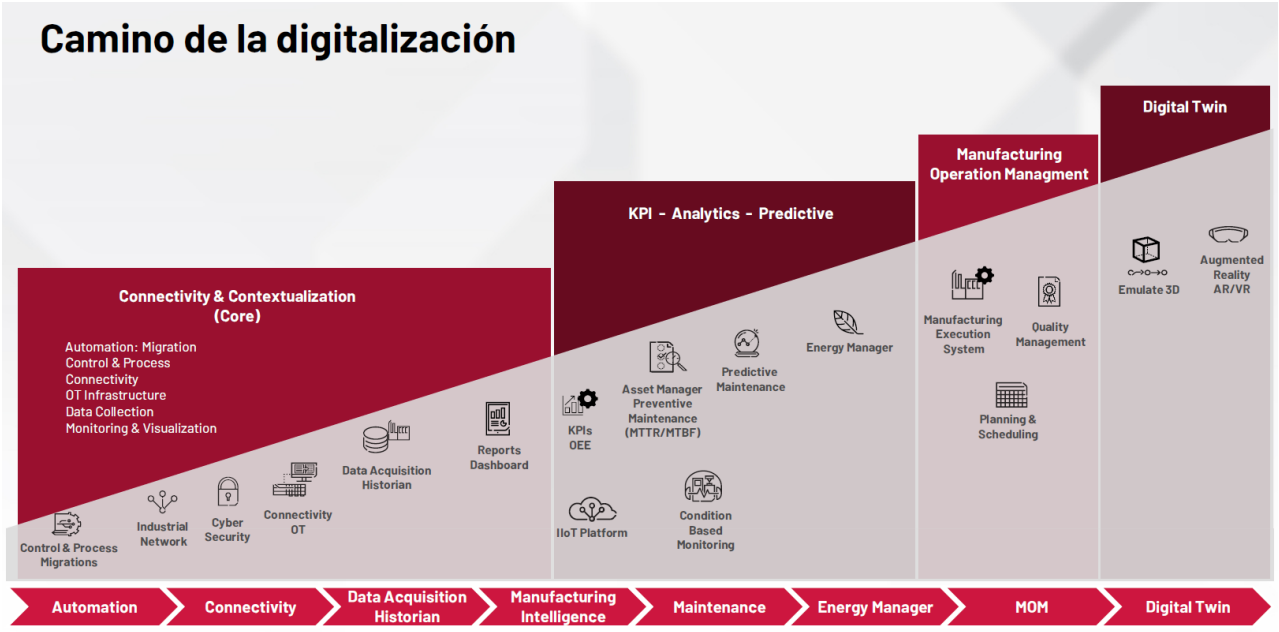

Figura 1.2 Camino hacia la digitalización.[2]

el de contar con personal altamente calificado que pueda hacer uso de ellas, y que haya adquirido suficientes habilidades técnicas y personales para adaptarse al mercado tecnológico que se encuentra en constante evolución. Como se muestra en la Figura 1.2, desde que se implementaron los sistemas de automatización en la industria hasta la época actual donde se integró la digitalización, se han desarrollado tecnologías innovadoras que van desde la adquisición histórica de datos, pasando por el manejo eficiente de la energía y la optimización en la planeación de proyectos, las cuales traen consigo desafíos para los especialistas pero también grandes oportunidades para la mejora continua de productos y procesos ([Rockwell-Automation, 2024]).

El paradigma que ha sido descrito hasta ahora es llamado Industria 5.0, siguiendo la progresión histórica de las cuatro revoluciones industriales anteriores. La primera de ellas, la Industria 1.0, dio inicio con la creación de la máquina de vapor al aumentar la capacidad de producción de las empresas de manufactura y la eficiencia del transporte. La Industria 2.0 fue impulsada gracias al desarrollo de las líneas de manufactura en serie alimentadas con energía eléctrica, permitiendo por primera vez una producción en grandes volúmenes. La Industria 3.0 se caracterizó por el uso de computadoras conocidas como PLC, lo cual significó un primer paso hacia la digitalización de los procesos industriales.

Con la Industria 4.0 la información obtenida a partir de sensores, así como, sistemas de comunicación, control y supervisión puede ser manipulada para darle un sentido aún más amplio, pues es posible procesar las señales eléctricas recibidas en los equipos electrónicos y llevarlas del plano analógico al digital, hasta convertirlas en cadenas de bits y desde ahí abstraerlas en servicios digitales, con lo cual es posible tener software cada vez más especializado.[Gaiardelli, *et al.*, 2021]

[2]Tomada de [Rockwell-Automation, 2024].

A pesar de que la Industria 4.0 presentaba enormes beneficios no estaba exenta de problemáticas, en particular existen cuatro áreas donde se puede registrar un impacto negativo [Chaudhari*, et al.*, 2021]:

- Aspecto ecológico. El balance ecológico, en general, no es tomado en cuenta. Se pone énfasis en el incremento de la eficiencia de la producción y la rapidez con que se pueden comercializar los productos, dejando de lado el esfuerzo para preservar el medio ambiente.
- Sustentabilidad. El consumo de energía responsable y el manejo de desechos no son temas considerados como primordiales, y a pesar de que existe trabajo en el área de fuentes de energía renovables, estos ocupan un lugar secundario en el panorama económico mundial.
- Desempleo y costo humano. El desempleo es una gran amenaza que puede traer inestabilidad en varios sectores de la población. Y puede ser alentado por el desplazamiento de empleados en el sector de la transformación por sistemas o equipos inteligentes, así como por reducciones de personal debido a los requerimientos de capacitación cada vez más elevados.
- Impacto psicológico. Existe una percepción de falta de "propósito" en la sociedad, por lo que hay incremento en los problemas de salud mental.

El esquema económico seguido por la Industria 4.0 ponía énfasis en el producto final listo para el consumo, dejando de lado la preocupación por el cuidado del medio ambiente y por el bienestar de los individuos, tanto de aquellos involucrados en la producción como los usuarios finales. En el ámbito industrial el enfoque estaba puesto, sobre todo, en la intercomunicación entre los equipos de las fábricas inteligentes y sus modelos digitales, dejando al ser humano con un papel de supervisión menos activo.

Como respuesta a las problemáticas anteriormente planteadas se establecieron los fundamentos para que emergiera una nueva revolución industrial, la Industria 5.0. El interés principal de este paradigma es el de hacer los procesos de manufactura sustentables desde una perspectiva económica, ecológica y social. Este nuevo punto de vista está basado en una aproximación sistemática a las tecnologías habilitadoras, es decir, aquellas que por sí solas o en combinación con otras generan una mejora significativa en el rendimiento, competencias y bienestar de sus usuarios.

Algunas de las tecnologías habilitadoras claves son [Möller*, et al.*, 2022]:

- Materiales avanzados. El interés se centra en soluciones que combinen la energía y los recursos naturales de tal forma que sean mejor aprovechados.
- Procesos y manufactura inteligentes. Utiliza Big Data análisis, AI, Machine Learning (ML) y simulaciones para mejorar el rendimiento de los procesos de manufactura, minimizar el desperdicio y reducir el consumo de energía y recursos.
- Nanotecnología. Se enfoca en crear materiales y dispositivos en una escala en el orden de los nanómetros, es esencial en la electrónica digital, y busca mejorar la eficiencia energética.
- Manufactura sustentable. Se caracteriza por desarrollar una manufactura inteligente con procesos de producción mejorados y más limpios.

Figura 1.3 Diferencias entre la Industria 4.0 y la Industria 5.0.[3]

Contrario a los objetivos de gran parte de la industria actual, basados en producir y consumir *ad infinitum*, para la Industria 5.0 la mayor preocupación es la de hacer la industria manufacturera sustentable, buscando la mejor manera de aprovechar los recursos naturales, haciendo más eficiente el consumo y producción de energía y dando un mejor tratamiento a los desechos.

Existe también una preocupación por poner a los seres humanos como parte activa y creativa en los procesos industriales, dejando que los robots y sistemas inteligentes se encarguen de tareas repetitivas o muy complejas, pero en estrecha colaboración con el quehacer humano. La Figura 1.3 muestra algunos puntos de discordancia entre el paradigma industrial que está en desarrollo (Industria 5.0) y el anterior (Industria 4.0).

Este nuevo paradigma se encuentra fundamentando en la llamada economía circular, que plantea la posibilidad de optimizar el uso de materiales y energía, así cómo la producción de bienes que regresen al sistema al final de su vida. Es decir, que después de la etapa de producción y consumo sean manejados de manera responsable como deshechos, incluyendo un marco legal y un mercado para materiales de segunda mano. Esto significaría un enorme cambio, considerando que actualmente la industria se rige por un esquema de producir, usar y desechar. [Möller, *et al.*, 2022]

Una herramienta muy importante para conseguir los objetivos de esta nueva perspectiva es el uso de gemelos digitales, pues a partir de ellos se pueden comprender los fenómenos involucrados en los diferentes procesos mediante su simulación, además de permitir tener una visión más transparente de la manera en que funcionan los elementos dentro del sistema. Esto tendrá como consecuencia

[3]Tomada de [Rockwell-Automation, 2024].

una reducción en los costos y tiempo de la puesta en marcha o comisionamiento de los proyectos industriales, y a su vez, podría ayudar a reducir el impacto energético y los desechos generados en el mismo, ya que se usarían solo los materiales, la energía y los equipos estrictamente necesarios.

Los gemelos digitales también pueden ser importantes en la consecución de una operación óptima y segura, ayudando a las diferentes industrias a procurar la existencia de normas y procedimientos basados en los datos arrojados por estos, es decir, los rangos y parámetros deseables durante la ejecución del proceso. Obteniendo esta información se podrán especificar métodos de mantenimiento preventivo y correctivo mediante el análisis del comportamiento del modelo virtual. Con la ausencia de estos procedimientos se corre el riesgo de presentar problemas de diferente índole, que pueden ir desde deterioro o averías en los equipos hasta accidentes con consecuencias fatales.

A su vez, es indispensable contar con herramientas que presenten información sobre las diferentes variables físicas de interés y de otros datos importantes como conteos y medidas de tiempo, pues solo así es posible tomar decisiones y acciones que contribuyan a obtener los mejores resultados. Para este propósito, la industria se vale de medios tradicionales como la obtención de datos directamente del sistema físico para luego enviarlos a un medio digital para su almacenamiento y posterior procesamiento. Con la ayuda del gemelo digital se pueden, además, proyectar escenarios que responden a la pregunta: ¿qué pasaría sí...?.

Se puede afirmar que si el concepto de Gemelo Digital en conjunto con las demás tecnologías habilitadoras (Figura 1.4) se inscribe dentro de un esquema como el descrito por la Industria 5.0 pueden promover un crecimiento armónico y de cooperación mutua en la industria, la economía y la sociedad en general.

1.1. Objetivo

El presente informe de trabajo profesional tiene como objetivo general explicar el desarrollo y mantenimiento de un software modular diseñado para realizar comisionamientos virtuales de instalaciones industriales complejas. Se trata de una aplicación de escritorio que hace uso de gemelos digitales para modelar los entornos industriales reales, pudiendo a su vez realizar conexiones con PLC y robots, generar lógicas internas e importar y exportar archivos de tipo Diseño Asistido por Computadora (CAD, por sus siglas en inglés *Computer-Aided Design*).

Este software tiene la finalidad de poner a prueba la lógica escrita para los PLC antes de su implementación en sistemas reales. De igual forma, sirve para comunicarse con robots mediante el protocolo Protocolo de Control de Transmisión (TCP, por sus siglas en inglés *Transmission Control Protocol*) y, de esta manera, verificar si estos equipos cumplen los requerimientos del usuario antes de adquirirlos en grandes volúmenes o siquiera adquirirlos.

Lo anterior se realiza mediante diferentes herramientas y objetos virtuales especiales que permiten simular la disposición física y el comportamiento de los equipos, así como generar sistemas que pueden

[4]Realizada con información tomada de [Qi, *et al.*, 2021].

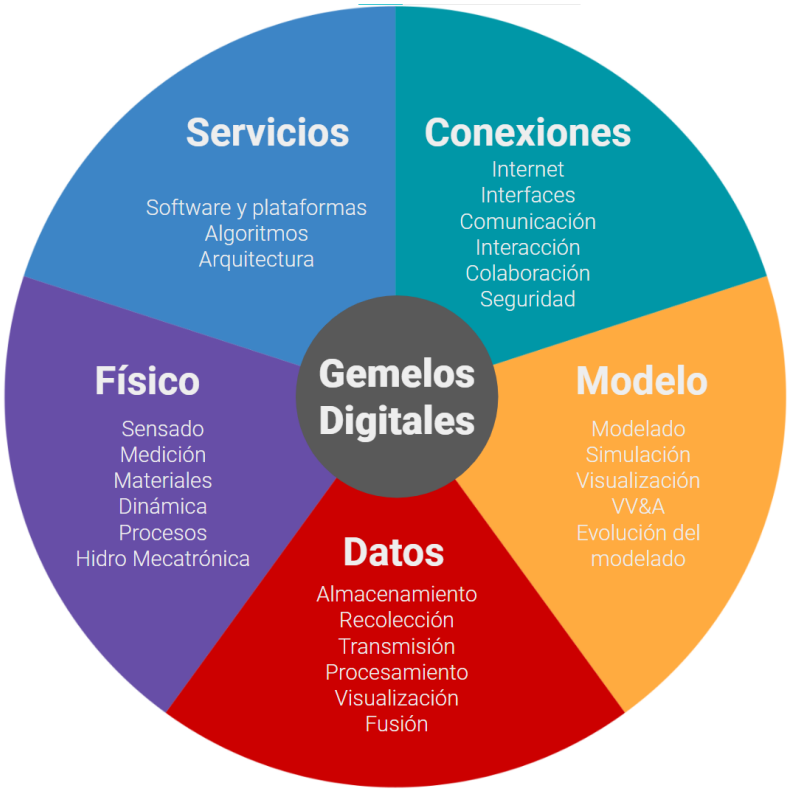

Figura 1.4 Marco de tecnologías habilitadoras para gemelos digitales.[4]

controlarse dentro de la simulación con lógicas escritas en diferentes lenguajes de programación. Por otro lado, puede ser vinculado a algunas aplicaciones externas para intercambiar datos para su renderizado o procesamiento.

Este software permite que se detecten y corrijan los errores en una etapa temprana del desarrollo del proyecto y se evite comprar equipos que no serán de utilidad, dando como resultado un ahorro significativo en tiempo y dinero.

1.1.1. Objetivos particulares

- Explicar el funcionamiento general del software y sus diferentes alcances.
- Mostrar la forma en que se agregan funcionalidades a pedido de los usuarios para cumplir con los requisitos y estándares por ellos solicitados.
- Comprender el proceso para dar mantenimiento al software para que funcione en condiciones óptimas y no arroje resultados equivocados.

Capítulo 2

Antecedentes

2.1. Gemelos digitales

El concepto de Gemelo Digital fue introducido en 2003 en la Universidad de Michigan por el Dr. Michael Grieves en un curso sobre el ciclo de vida de los productos industriales. Su principal característica consiste en ser un objeto digital basado y virtualmente indistinguible de un objeto físico real. La información intercambiada entre ambos y la obtenida de cada uno individualmente permite predecir comportamientos y estados del objeto físico a partir de comportamientos y estados del objeto virtual.

Muchas empresas e instituciones actualmente utilizan gemelos digitales como parte del desarrollo de su tecnología, por ejemplo, la NASA cuenta con copias digitales de sus naves espaciales para monitorear su estado en tiempo real. General Electric y Chevron los utilizan para rastrear la operación de sus turbinas eólicas. Siemens utiliza modelos matemáticos y representaciones virtuales tridimensionales, así como análisis de elementos finitos, para rastrear temperaturas, tensiones y deformaciones en sus productos. [Tao y Qi, 2019] Por otro lado, en Singapur se está desarrollando un gemelo digital de toda la ciudad para ayudar a mejorar la calidad de vida de sus habitantes. [Walker, 2023]

Se distinguen tres partes fundamentales en el concepto de Gemelo Digital: 1) productos físicos en un Espacio Físico; 2) productos virtuales en un Espacio Virtual; 3) la información y datos que conectan a ambos y les permiten interactuar.

Actualmente, los avances en la tecnología permiten obtener modelos tridimensionales bastante fidedignos de los objetos físicos, con la posibilidad de ajustar el nivel de detalle que se quiere dar al modelo virtual. Es decir, se puede elegir solo la geometría y atributos necesarios, haciendo que dichos modelos sean más ligeros y con un procesamiento más rápido, dando como consecuencia que el costo y tiempo durante la transferencia de datos sea menor. [Grieves, 2015]

Un gemelo digital debe de contar con las siguientes tres características [Yuchen Jiang y Kaynak, 2021]:

- *Mirroring*: Se refiere a la proyección de un elemento físico en un modelo virtual, ayudándose de CAD y herramientas como *Point Cloud Representation* (PCR), tomografías e inspección por ultrasonido, por mencionar algunas.

- *Shadowing*: Es la capacidad de obtener el estado actual del sistema físico y sincronizar el elemento físico y su contraparte virtual. Es una actualización en tiempo real del estado del objeto virtual ante cambios del objeto físico.
- *Threading*: Se refiere a la capacidad de conectar las diferentes etapas de la operación y las diferentes instancias de gemelo digital existentes. Permitiendo que un componente conozca el estado actual de otro, a través del flujo de información creado por todas las interconexiones, y que pueda realizar monitoreo y control en tiempo real de toda la planta.

Las empresas que cuentan con una gran cantidad de datos históricos pueden encontrar poco útil esta información si no es llevada al plano digital. De igual forma, los profesionales enfocados en el diseño tridimensional, simulaciones y procesamiento de datos pueden encontrar dificultades al analizar los datos provenientes de los procesos físicos. En el caso de los departamentos encargados de la planeación y las finanzas también pueden existir dificultades a la hora de mitigar costos innecesarios y reducir los tiempos de entrega.

Dadas las características del concepto de Gemelo Digital, se presenta como un elemento con el que las problemáticas mencionadas anteriormente pueden ser resueltas al reunir estas diversas áreas en un entorno digital. Sin embargo, durante esta integración se deben tomar en cuenta una gran cantidad de consideraciones.

Por ejemplo, es importante elegir los tipos de datos adecuados, esto implica elegir correctamente las variables físicas esenciales para modelar el proceso, las unidades de las variables medidas, la resolución de los datos, así como las interfaces, formatos y software utilizados por las diferentes marcas de los elementos involucrados en el proceso.

En este sentido, es necesario plantear la estandarización de los formatos empleados en la comunicación entre el mundo físico y virtual. Por ejemplo, el uso de Lenguaje de Marcado Extensible (XML, por sus siglas en inglés *eXtensible Markup Language*), que actualmente se puede encontrar en áreas muy distintas. O el COMTRADE (common format for transient data exchange), que es utilizado en el sector eléctrico.

Otro problema emerge cuando se desea que diferentes gemelos digitales interactúen entre sí. En este caso, pueden surgir dificultades al intentar hacer compatibles programas escritos para modelos virtuales diferentes, por lo que, la existencia de una plataforma en la que pueda ejecutarse cualquier modelo, bajo un estándar universal de diseño y desarrollo debería ser una meta a seguir.

La necesidad de los gemelos digitales con respecto al panorama actual de la industria se puede ejemplificar con los siguientes puntos (tal como menciona [Yuchen Jiang y Kaynak, 2021]):

- El diseño de los productos no toma en cuenta las condiciones reales en la práctica manufacturera (existe una brecha entre el diseño y la manufactura).
- No es posible saber la calidad final del producto durante la manufactura (existe una brecha entre la manufactura y el control de calidad).
- El mejoramiento del diseño y los procesos de manufactura debería de ser guiado por los lotes de productos anteriores (existe una brecha ente el producto y los lotes terminados).

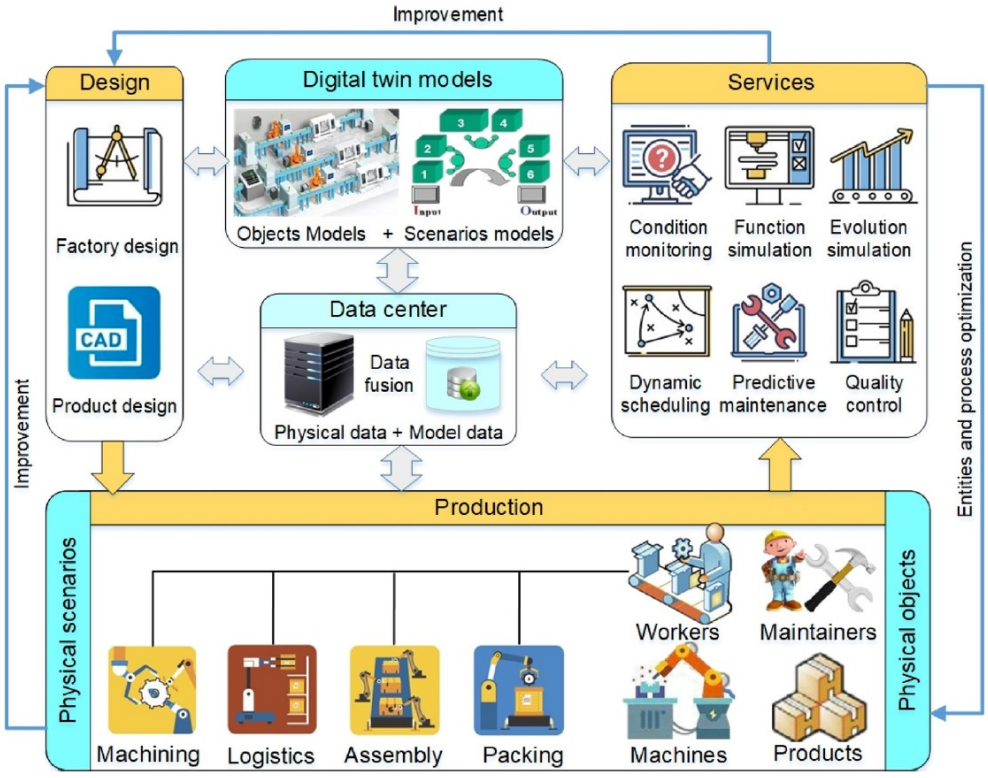

Figura 2.1 Composición y aplicación de un gemelo digital.[1]

- Es difícil controlar las fluctuaciones en los costos de los procesos. Existe una falta de información en tiempo real a través de las diferentes etapas de manufactura del producto.

La Figura 2.1 refleja el esquema general en el que se inscribe un gemelo digital dentro de un proceso industrial. Se puede observar como la etapa de diseño se nutre de la etapa de producción, pero también de los datos proporcionados por el modelo digital, pudiendo visualizar mejor los datos complejos del sistema. De igual forma, la calidad del diseño puede ser evaluada en el gemelo digital mediante la simulación de diferentes escenarios, detectando defectos y áreas de oportunidad.

Por otro lado, desde el punto de vista administrativo el gemelo digital permite la óptima (re)configuración de los recursos, equipos y mano de obra en sitio. Además, desde la perspectiva de control se puede rastrear todo lo ocurrido en el sistema físico para optimizar las estrategias de control [Qi, *et al.*, 2021].

En 2020 CIMdata realizó un estudio global entre diversas compañías industriales para conocer el estado en el que se encontraba el uso de gemelos digitales (Figura 2.2), y arrojó que en ese momento el 11 % lo utilizaba en producción, 20 % estaba corriendo pruebas pilotos. Sin embargo, para 2023 se esperaba que incrementaran a 34 % y 43 % respectivamente. En ese momento, para el 22 % no eran aplicables y 1 % no lo veía aplicable ni siquiera 3 años después [CIMdata, 2020].

[1]Tomada de [Qi, *et al.*, 2021].
[2]Tomada de [CIMdata, 2020].

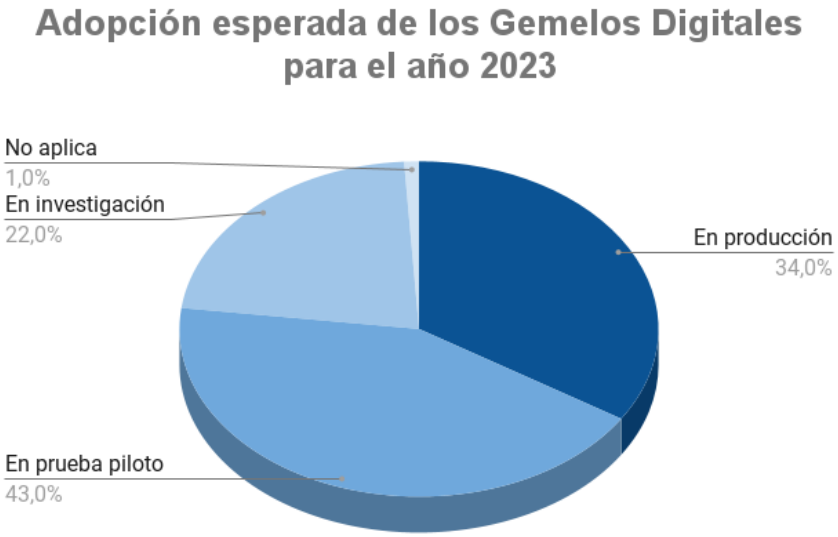

Figura 2.2 Adopción esperada de los gemelos digitales para el año 2023.[2]

2.2. Comisionamiento virtual

Para que una compañía manufacturera pueda ser exitosa en el mercado un factor importante es la elaboración de productos con altos estándares de calidad. No obstante, un buen producto no es sinónimo de rentabilidad financiera en cambio, y debido a que los precios de los productos son regulados por el mercado, para mantener las ganancias positivas es necesario que las empresas puedan reducir lo más posible sus costos de producción [Lee y Park, 2014].

A pesar de ello, a medida que los procesos industriales se vuelven más complejos resulta cada vez más complicado realizar la puesta en marcha de plantas completas en tiempo y forma. Además, debido a que se requiere de una gran inversión económica se vuelve necesario que el diseño de las plantas pueda mantener rentable el modelo de negocios en el largo plazo.

En este sentido, se vuelve de gran relevancia el contar con software especializado que permita examinar los diferentes resultados y posibles fallas dentro de un proceso. De otro modo, la estabilidad de los sistemas recaería solamente en el comisionamiento de plantas físicas reales, lo cual puede resultar muy costoso y consumir mucho tiempo.

El software para simular estos procesos ha incrementado significativamente sus alcances gracias a las tecnologías habilitadoras y, sobre todo, al uso de gemelos digitales creados en equipos con gran poder computacional.

El comisionamiento virtual utiliza dicho software especializado para ayudar a las compañías a reducir tiempos y costos, y se refiere a la fase del proyecto entre la preparación de programas en lenguajes de bajo nivel para la automatización de los equipos y la puesta en marcha real de estos equipos en la

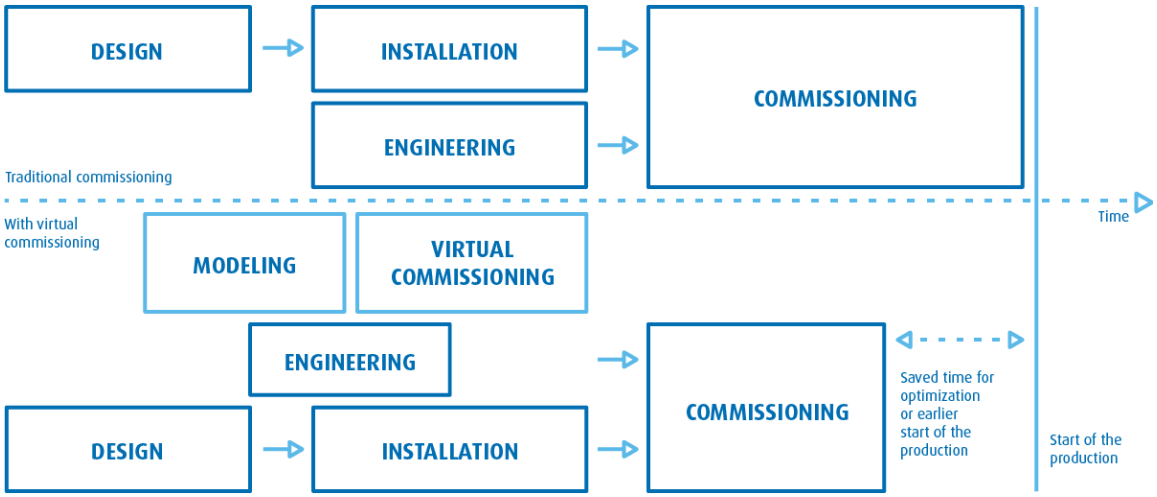

Figura 2.3 Diferencias entre el comisionamiento virtual y el comisionamiento tradicional.[3]

fábrica [Kuc, 2021].

Se estima que 90 % del tiempo utilizado en el comisionamiento es para el desarrollo de sistemas eléctricos y de control, y de este tiempo, el 70 % se puede atribuir a errores de software [Striffler y Voigt, 2023].

La principal tarea durante el comisionamiento virtual es la de realizar inspecciones en estaciones virtuales (modelos virtuales de la planta real), para detectar errores de diseño, de programación o de compatibilidad entre equipos. De esta forma, el comisionamiento virtual puede reducir el tiempo del comisionamiento real hasta en un 75 % gracias a las mejoras en la fabricación en el inicio del proyecto [Lee y Park, 2014]. En la Figura 2.3 se puede observar un esquema que muestra las ventajas del comisionamiento virtual con respecto del comisionamiento tradicional.

Una de las aplicaciones fundamentales del comisionamiento virtual es la de probar la lógica escrita para PLC y robots, con la posibilidad de simular solamente los dispositivos necesarios para inspeccionar que las lógicas de automatización funcionen correctamente o para probar que el comportamiento en modo manual sea adecuado.

Sin embargo, en ocasiones se necesitan simular entornos industriales complejos, compuestos de muchos elementos diferentes, entre los cuales pueden estar PLC y robots, y donde los retrasos en el inicio de las operaciones pueden significar perdidas en el orden de los cientos de miles de euros. [Kuc, 2021] En este sentido, el comisionamiento virtual permite que cada que se desee agregar un nuevo dispositivo este pueda ser analizado mediante un símil virtual en el contexto general de todo el proceso.

2.2.1. Configuraciones para el comisionamiento de proyectos industriales

El comisionamiento o puesta en marcha de proyectos puede seguir diferentes tipos de configuraciones, según se trate de elementos virtuales o reales (Figura 2.4). A continuación, se enumeran los

[3]Tomada de [Kuc, 2021].

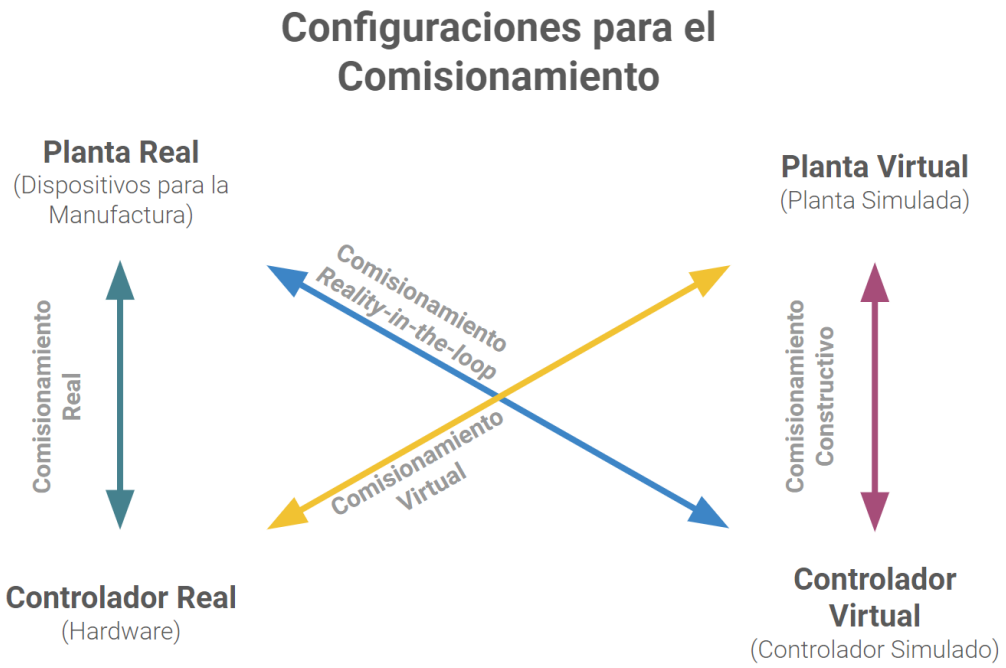

Figura 2.4 Configuraciones para el comisionamiento de proyectos.[4]

posibles escenarios ([Lee y Park, 2014]):

- Comisionamiento real: compuesto de una planta real y un controlador real.
- Comisionamiento virtual (*Hardware in the loop*): compuesto de una planta virtual y un controlador real.
- Comisionamiento *Reality in the loop*: compuesto por una planta real y un controlador virtual.
- Comisionamiento constructivo (*Software in the loop*): compuesto por una planta virtual y un controlador virtual.

En el comisionamiento constructivo, también llamado *Software in the loop*, el programa de control se implementa en un lenguaje común para PLC (e.g., de acuerdo a la IEC61131-1), se carga a un emulador virtual y se pone a prueba en un software de simulación. La comunicación entre ellos se da por medio de algún protocolo virtual. Se usa en etapas tempranas del desarrollo [Striffler y Voigt, 2023].

El comisionamiento *Reality in the loop* tiene como objetivo reducir los daños en la planta y los residuos en la producción [Striffler y Voigt, 2023].

En el caso del comisionamiento *Hardware in the loop* (que es al que se conoce como comisionamiento virtual) la comunicación entre el controlador real y la planta simulada se da por medio de un protocolo por redes físicas (e.g., Profi-bus, Profinet, EtherCAT). Tiene la ventaja de que el sistema de comunicación y el hardware utilizados pueden ser puestos a prueba simultáneamente [Striffler y Voigt, 2023].

[4]Realizada con información tomada de [Lee y Park, 2014].

Por lo general, el controlador que se utiliza para comisionar es un PLC, por lo cual es importante que el software para simular la planta sea compatible con los drivers de las diversas marcas de PLC.

Estos controladores cuentan con tablas de entradas/salidas y un ciclo donde estas se analizan y se determina su estado, de esta forma la lógica escrita se resuelve con cada actualización de los estados en cada ciclo [Lee y Park, 2014]. Por ello, es importante que el software de simulación esté bien sincronizado con los ciclos del PLC y pueda responder a las variaciones del sistema de control en tiempo real.

2.3. Técnicas de desarrollo de software para la creación de entornos virtuales

2.3.1. Programación orientada a objetos

La Programación Orientada a Objetos (OOP, por sus siglas en inglés *Object-Oriented Programming*) es un modelo de programación en el que el diseño se desarrolla alrededor de datos (objetos) en vez de funciones [Gillis, 2021].

Un objeto es un elemento que contiene características y funciones específicas que lo definen como una entidad particular y lo diferencian de otros objetos. Las características se refieren a la información que define cada objeto, mientras que las funciones se refieren a su comportamiento. Por ejemplo, un objeto *persona* puede contener como características: color de pelo, altura, forma de los ojos, etc. y como comportamientos: el respirar, comer, dormir, etc. [Weisfeld, 2009]

Dentro de este paradigma, a las características se les conoce como atributos y a los comportamientos como métodos o funciones. Si un objeto requiere conocer información o acceder a una función de otro objeto, el primero puede invocar o llamar los métodos del otro.

En la Figura 2.5 se observa un objeto llamado *myObject* haciendo llamadas a los métodos del objeto *Math*. Uno de estos métodos pueden ser por ejemplo *sumar*, de esta forma si *myObject* quiere saber la suma de dos números "pregunta" a *Math* invocando dicho método y recibiendo el resultado sin necesariamente saber cómo es que se obtuvo, pues la función de sumar es propia de *Math* y no de *myObject* .

Este paradigma ha estado en uso desde la década de 1960, y se distingue por el hecho de que el flujo de datos dentro de cada programa está definido por la comunicación entre objetos, pues cada objeto contiene su propia información. Esto quiere decir que esta no se encuentra centralizada en un solo lugar donde cualquier función, desde cualquier parte del programa, puede tener acceso a ella. Por lo tanto, se reduce el riesgo de realizar operaciones erróneas y se tiene mayor control sobre la circulación y actualización de dicha información [Weisfeld, 2009].

Además de los objetos, métodos y atributos existen las clases, que pueden entenderse como el plano o esquema con que un objeto individual va a construirse o *instanciarse*. Son tipos de datos definidos por el usuario en los cuales se establecen los atributos y métodos que cada objeto, de ese tipo de dato

[5]Tomada de [Weisfeld, 2009].

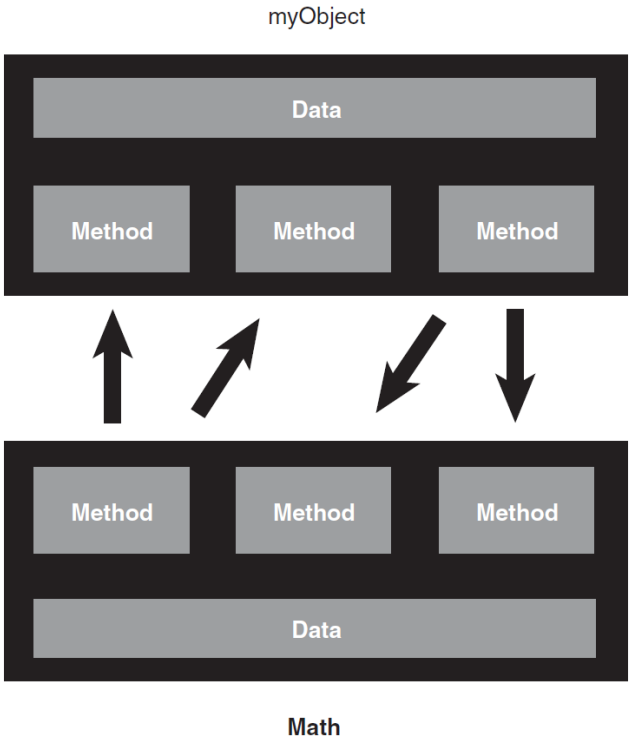

Figura 2.5 Comunicación entre objetos.[5]

particular, va a contener al ser instanciado[Gillis, 2021].

La Figura 2.6 muestra la relación entre clase, objeto, atributos y métodos. Se observa como la clase *Human* (de naturaleza más general) define un humano (objeto) particular que cuenta con diferentes atributos (*Age, Address*) y métodos (*Verify, Send Mail*), los cuales podrían servir, por ejemplo, a un software de verificación de usuarios.

2.3.2. Principios de la programación orientada a objetos

La OOP está basada en los siguientes principios [Gillis, 2021]:

- Encapsulamiento: La información más importante de cada objeto, definida en su clase, permanece inaccesible y solo unas cuantas propiedades son expuestas a otros objetos y clases. Esto provee de seguridad y evita la corrupción en la información.
- Herencia: Es posible reutilizar código entre clases cuando existe una relación jerárquica entre ellas. Se pueden definir subclases a partir de una clase cuando cuentan con una relación del tipo *Y es un X*. Por ejemplo, una clase *Mamíferos* puede tener subclases de tipo *Perros* y *Gatos* (Figura 2.7), pues cada uno de ellos *es un Mamífero* [Weisfeld, 2009]. Las subclases tienen atributos y métodos que son comunes a la superclase de la que heredan, pero a su vez pueden definir atributos y métodos propios.

[6]Tomada de [Gillis, 2021].

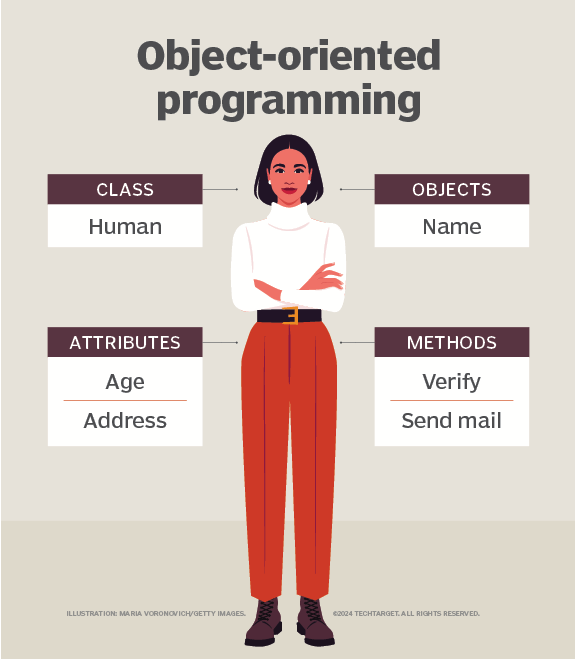

Figura 2.6 Relación entre clase, objeto, atributos y métodos en la programación orientada a objetos.[6]

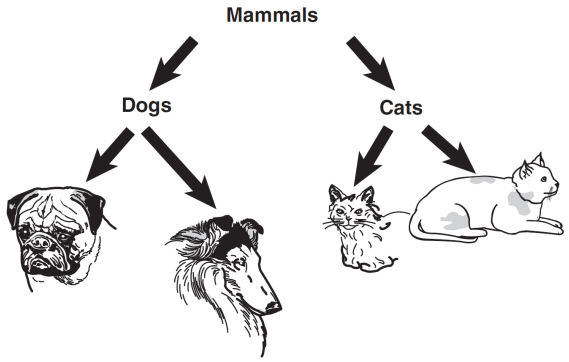

Figura 2.7 Ejemplo de herencia entre objetos.[7]

- Abstracción: Los objetos pueden exponer solo los mecanismos necesarios para que otros objetos los utilicen, escondiendo así complejidad innecesaria en las implementaciones.
- Polimorfismo: El polimorfismo está ligado a la herencia, ya que gracias a ella es posible que distintos objetos puedan compartir procedimientos (cuando heredan de la misma clase), no obstante, cada subclase determinará el comportamiento particular que necesita. Por ejemplo, los *Mamíferos* tienen el comportamiento de *Emitir Sonidos*, sin embargo, los perros *Ladran* y los gatos *Maullan*.

2.3.3. Patrón de diseño modelo - vista - modelo de vista

Al realizar el diseño de un software, sobre todo si es muy complejo, es necesario tomar en cuenta diversos puntos fundamentales como la forma en que se manipulará la información, las normas que se

[7]Tomada de [Weisfeld, 2009].

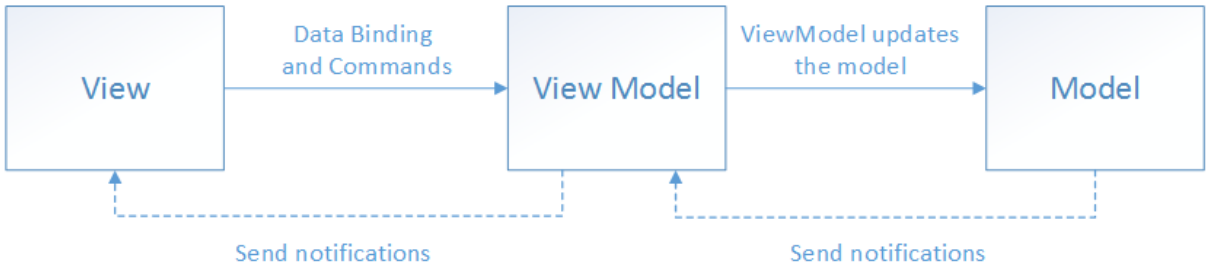

Figura 2.8 Estructura del patrón MVVM.[8]

deben de seguir o la manera en que serán presentados los datos al usuario final.

Uno de los objetivos principales es el de definir y separar de forma efectiva las diferentes partes que conforman el software, estableciendo sus funcionalidades, sus límites y la interacción entre ellos.

Para este propósito, se puede aplicar el principio de Separación de Intereses (SoC, por sus siglas en inglés *Separation of Concerns*) que consiste en organizar en grupos (conocidos en inglés como *concerns*) las actividades y los datos relacionados, y mantenerlos separados unos de otros. De esta forma, es más sencillo comprender las responsabilidades de cada unidad, modificar su comportamiento, encontrar fallas, corregirlas y reutilizar o implementar nuevas funcionalidades. [Kouraklis, 2016]

Una forma concreta de la aplicación de este principio es en las tres capas o niveles principales que componen una aplicación [Kouraklis, 2016]:

- *Presentation layer*: Es la Interfaz de Usuario donde se representan diferentes estados de la información.
- *Business layer*: Esta parte valida los datos y se encarga de las reglas de negocio.
- *Data Access layer*: En esta unidad se conecta la aplicación con el medio de almacenamiento de datos elegido.

Estas capas son una forma de encapsular la lógica que cada parte debe de realizar. A partir de estructura general se pueden desarrollar diferentes patrones de diseño, en los que sus partes están relacionadas en diferente proporción con las capas mencionadas anteriormente.

Uno de ellos es el patrón MVVM, que consta de tres partes: el modelo (*Model*), la vista (*View*) y el modelo de vista (*View-Model*), como muestra el esquema de la Figura 2.8

La vista es responsable de definir la estructura, capas y apariencia de aquello que el usuario ve en su pantalla. Puede contener lógica que se encargue de aspectos visuales, pero no lógica de negocio. [Stonis, 2022]

El modelo de vista implementa propiedades y comandos a los que la vista puede enlazarse, así como notificaciones sobre cualquier cambio en el estado a través de eventos. El modelo de vista define la

[8]Tomada de [Stonis, 2022].

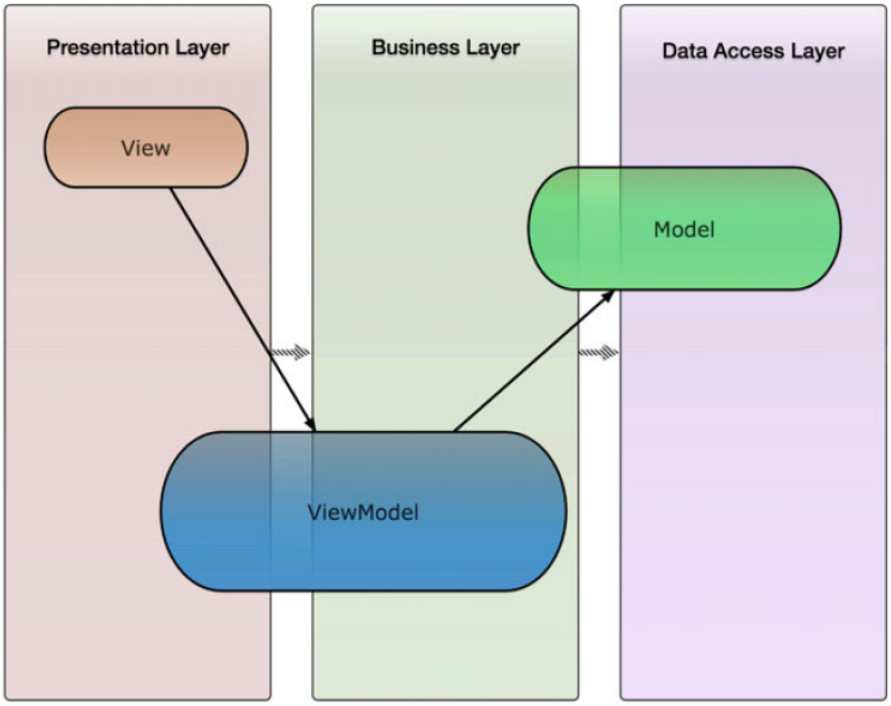

Figura 2.9 Relación entre las tres capas de aplicación y el patrón MVVM.[9]

funcionalidad ofrecida por la vista, y la vista determina como será mostrada. Por otro lado, también se encarga de coordinar la interacción de la vista con los diversos modelos necesarios. [Stonis, 2022]

Por último, los modelos son elementos no visuales que encapsulan todos los datos necesarios consumidos por la vista.

En términos de la arquitectura de tres niveles explicada al inicio de esta sección y representada en la Figura 2.9, el modelo se relaciona casi completamente con la capa de datos, teniendo en ocasiones una pequeña responsabilidad de la lógica de negocio para la conversión o validación de datos. [Kouraklis, 2016]

La vista reside en la capa de presentación, mientras que el modelo de vista se encarga de diversas funciones, sobre todo de aquellas que tienen que ver con la manipulación de los datos brindados por el modelo para presentarlos en la vista, esto es, implementa lógica de negocios pero también define parte de la capa de presentación. [Kouraklis, 2016]

2.3.4. Transmisión y almacenamiento de datos

En el desarrollo de software es importante contar con sistemas que permitan la transmisión de datos de forma eficiente, compacta y sencilla.

Para lograrlo se utiliza el concepto de serialización, que consiste en convertir un objeto en una secuencia de bytes para su almacenamiento o transmisión a la memoria, una base de datos o un archivo. En la

[9]Tomada de [Kouraklis, 2016].

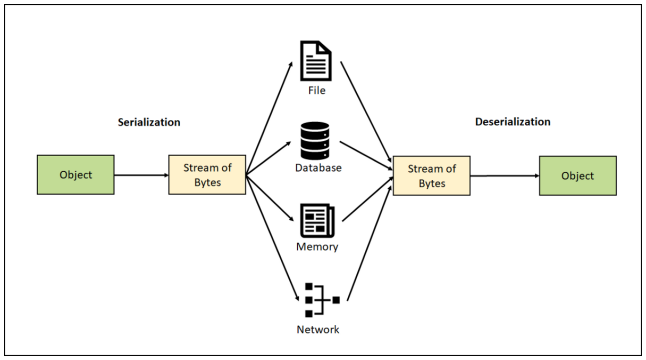

Figura 2.10 Proceso de serialización y deserialización.[10]

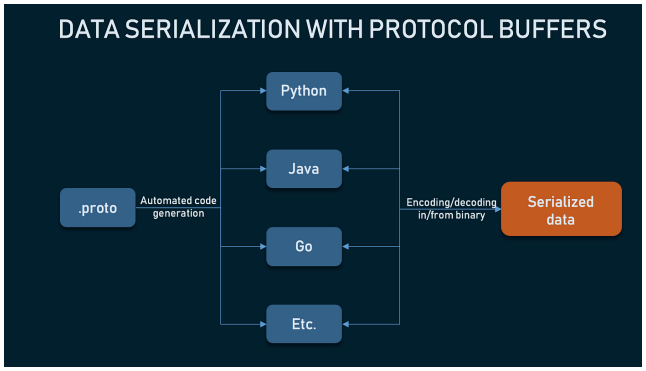

Figura 2.11 Serialización con Protobuf.[11]

secuencia resultante se incluye al objeto e información relevante sobre este como su tipo, su versión, nombre del *assembly* y referencias culturales (Figura 2.10). [Dollard, 2024]

Es posible utilizar serialización binaria o XML, la primera utiliza el código binario para generar una serialización compacta, mientras que la serialización XML serializa las propiedades y campos públicos del objeto en una secuencia XML que se ajusta a un documento específico de este lenguaje.

En el caso de la serialización binaria existe un formato creado por Google llamado *Protocol Buffer (Protobufer)*, el cual es compatible con una amplia gama de lenguajes de programación, admite una gran variedad de tipos de datos (*string, int, float, bool, enum, maps*, entre otras) y su sintaxis es independiente del lenguaje y la plataforma utilizados (Figura 2.11).[Ullah, 2022]

Al ser almacenado de forma binaria es más pequeño que formatos basados en text (XML, JSON), por lo que es muy rápido de transferir y utiliza menos memoria para almacenarse.

Con Protobuf se definen las estructuras de los modelos y mediante un compilador se codifican y convierten los datos hacia el lenguaje utilizado.[Dineshchandgr, 2022]
Otro aspecto importante es que Protobuf puede ser utilizado para implementar comunicación de Llamada de Procedimiento Remoto (RPC, por sus siglas en inglés *Remote Procedure Call*). Este

[10]Tomada de [Isaac, 2020].
[11]Tomada de [Dineshchandgr, 2022].

tipo de comunicación permite llamar funciones de una aplicación desde otra, esto se logra enviando peticiones de una aplicación (conocida como cliente) a otra aplicación (conocida como servidor), esperando la respuesta de la segunda a la primera.[Ullah, 2022]

Esta solicitud se hace mediante paquetes de datos con información relevante como parámetros, valores de retorno y procedimientos o funciones a ejecutar.

2.3.5. Paradigma entidad - componente - sistema

Dentro del diseño de software se utilizan diversos patrones que permiten solucionar problemas específicos en la programación. Un ejemplo de ellos es el Entidad-Componente-Sistema (ECS, por sus siglas en inglés *Entity-Component-System*), un patrón utilizado en aplicaciones complejas como video juegos o sistemas de simulación.

Está basado en el paradigma Diseño Dirigido por Datos (DDD, por sus siglas en inglés *Data-Driven Design*) en el que el punto central es la organización y manipulación de los datos o *data*. A diferencia de la OOP que está fundamentada en abstracciones, jerarquías y comportamientos de los objetos,en DDD la estructura de las aplicaciones está construida alrededor de los datos, sus relaciones y sus patrones de acceso. [Dang, 2023]

ECS se compone de tres conceptos básicos, el primero de ellos es la *entidad*, que se refiere a cualquier "cosa" que pueda ser simulada, por ejemplo, en un juego sería un jugador, un arma, un enemigo, etcétera. La entidad no contiene información más allá de su referencia en memoria, por lo que requiere ser especificada por *componentes*, que no son otra cosa que sus propiedades, por ejemplo, color, tamaño, velocidad, etcétera. Por otro lado, las entidades necesitan de una lógica que determine su funcionamiento, es aquí donde surge el tercer concepto, los *sistemas*, los cuales acceden a los componentes de la entidad para actualizar la información [Muratet y Garbarini, 2020]. La Figura 2.12 ejemplifica la relación entre estos tres elementos, se muestran tres entidades: A, B y C, con componentes que definen su traslación, rotación, locación en el espacio y si son representables gráficamente. Se observa, además, como las entidades se comunican entre sí y con el sistema que ejecuta la lógica para modificar los componentes que las definen.

2.3.6. Motor de físicas

Otro aspecto importante al implementar un software para simulación de gemelos digitales es el motor de físicas, el cual se refiere a un componente de software encargado de reproducir las leyes de la física en un entorno virtual. Se encargan de calcular y mostrar las interacciones físicas y los movimientos entre los objetos, brindando una experiencia realista e inmersiva al usuario.

Cada objeto cuenta con sus propiedades físicas particulares y estas se ven afectadas por la interacción con otros objetos. El motor de físicas usa algoritmos y modelos matemáticos complejos para simular dichas propiedades y sus efectos. Su operación básica se puede resumir en los siguientes puntos [Ipacs, 2023]:

[12]Tomada de [Gombert, 2020].

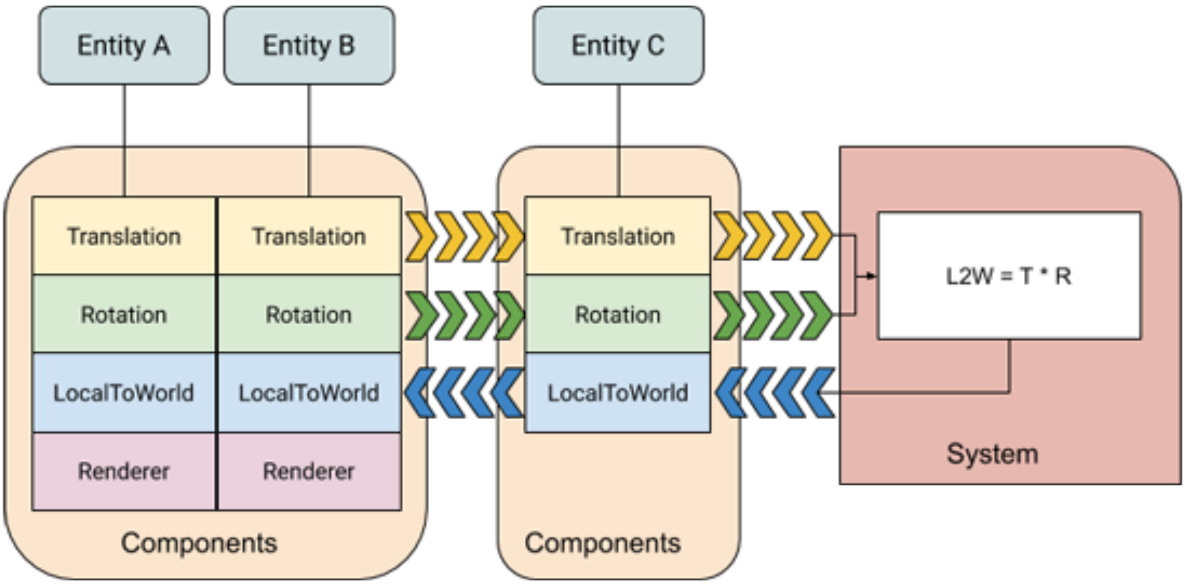

Figura 2.12 Esquema del paradigma entidad - componente - sistema.[12]

- *Input* o Entrada: El motor de físicas recibe datos de cada objeto (como posición, velocidad, masa, entre otras).
- Simulación: Se calcula el movimiento y comportamiento de cada objeto basado en la entrada y las leyes físicas. Se resuelven ecuaciones complejas de variables con respecto al tiempo.
- Detección de colisiones: Se calcula las fuerzas resultantes y dirección cuando los objetos colisionan entre sí.
- Respuesta: Se actualiza el estado de la simulación con la nueva posición, velocidad y orientación de los objetos, resultado de las acciones aplicadas.

Dentro del ciclo de actualización del sistema de simulación están involucradas varias etapas. En la Figura 2.13 se puede observar que el motor de físicas actúa en conjunto con la detección de colisiones y el motor de renderizado[13] para modificar los parámetros del sistema, sus entidades y sus representaciones tridimensionales durante cada ejecución de un ciclo.

2.3.7. Motor de renderizado

El motor de renderizado tiene la responsabilidad de generar la representación gráfica en 3D de los objetos existentes en la simulación. Se vale de la Unidad de Procesamiento Gráfico (GPU, por sus siglas en inglés *Graphics Processing Unit*) para realizar el procesamiento de elementos gráficos mediante programas especiales (*Shaders*) que utilizan los atributos necesarios para presentar el objeto en la pantalla.

Los tres principales atributos para renderizar objetos tridimensionales son: Vértices, Vectores Normales y Coordenadas UV (proyección de objetos 3D en una imagen 2D). Pero además, se necesitan de las

[13]El renderizado se refiere al proceso de generar imágenes fotorrealistas, o no, a partir de un modelo 2D o 3D por medio de programas informáticos.
[14]Tomada de [Serrano, 2019].

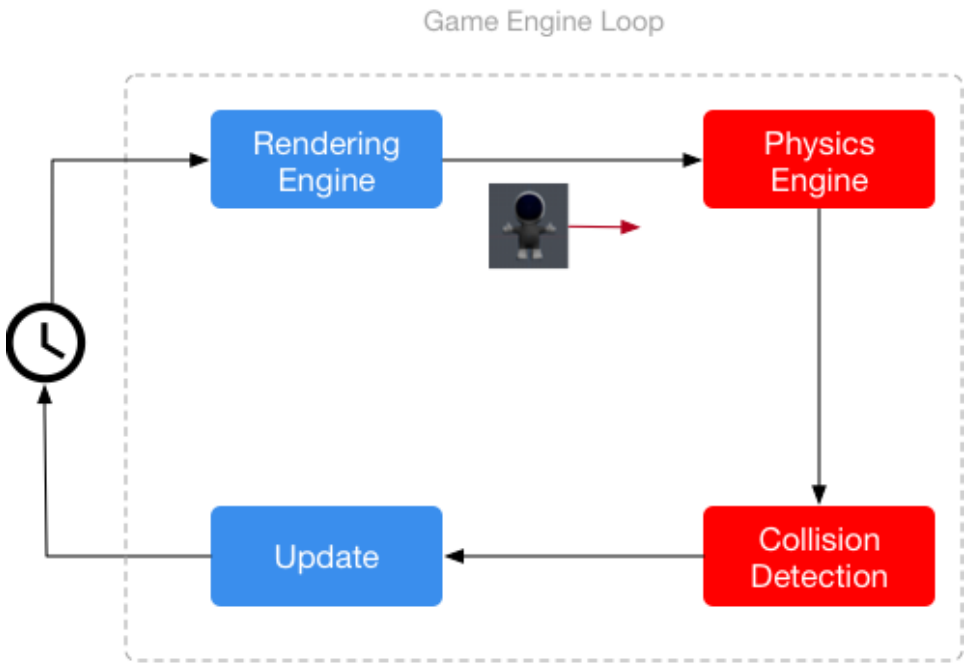

Figura 2.13 Flujo del ciclo de motores para la representación de objetos en la simulación.[14]

coordenadas espaciales para definir la posición del objeto en el mundo.

Las coordenadas espaciales se definen como una matriz 4x4, donde la matriz superior 3x3 contiene la rotación, y el vector en la última columna contiene la posición. Estas coordenadas son conocidas como *Modelo Espacial* [Serrano, 2019].

El renderizado de los modelos tridimensionales se basa en el llamado Renderizado Basado en Física (PBR, por sus siglas en inglés *Physically Based Rendering*), compuesto por una serie de técnicas que intentan replicar el comportamiento de la luz al impactar sobre los cuerpos, aproximándolo lo más posible al comportamiento en el mundo real. Para que un renderizado pueda considerarse basado en la física debe cumplir tres condiciones [Hoffman, 2012].

- Debe de estar basado en el modelo de superficie microfacética. Esta teoría dice que cualquier superficie a escala microscópica está compuesta por pequeños espejos reflejantes. Entre más caótica sea la alineación entre cada uno de ellos más áspera será la superficie.
- Debe respetar la ley de conservación de la energía.
- Usar Función de Distribución de Reflectancia Bidireccional (BRDF, por sus siglas en inglés *Bidirectional Reflectance Distribution Function*). Una función que toma como entrada la dirección de entrada de la luz, la dirección desde donde es observada la superficie (microfacética), el vector normal de la superficie. Y cuyo resultado aproxima cuanto contribuye cada rayo de luz de entrada en la luz reflejada sobre una superficie opaca dadas las propiedades de su material.

[15]Tomada de [Hoffman, 2012].

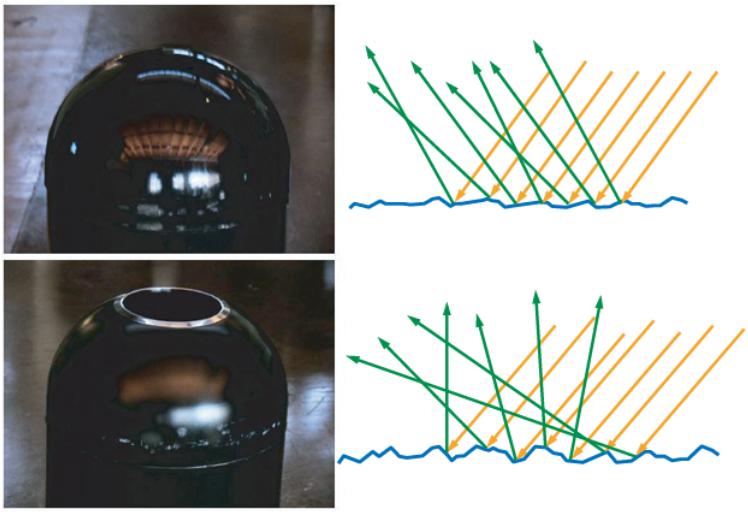

Figura 2.14 Superficie microfacética.[15]

En la Figura 2.14, se observa que la superficie superior es más lisa, por lo que la dirección de la luz reflejada solo tiene pequeñas variaciones, mientras que en la superficie inferior, al ser más rugosa, genera una mayor variación en la dirección de la luz reflejada y por lo tanto, es más borrosa a la vista [Hoffman, 2012].

Capítulo 3

Definición del problema

La industria mundial se ha visto renovada gracias a las tecnologías digitales desarrolladas por la Industria 5.0, sobre todo por aquellos sistemas denominados CPPS con los cuales es posible diseñar y manejar de forma sencilla procesos muy complejos. De esta forma, los elementos del sistema están estrechamente inter-conectados y pueden ser monitoreados en tiempo real, dando como resultado un mayor control y eficiencia durante la operación.

Uno de los principales problemas a que se enfrenta la industria manufacturera en la actualidad, es que durante la puesta en marcha de un proceso industrial hay una alta probabilidad de retrasos y elevación de los costos debido a problemas técnicos no previstos en la etapa de diseño, por ejemplo, errores en la programación de PLC y otros equipos como robots, que no pueden probarse sino hasta que se ha instalado toda o la mayor parte de la planta. Además de esto, se pueden llegar a presentar pérdidas monetarias si la maquinaria adquirida no cumple adecuadamente con los requerimientos del proyecto, debido a una errónea evaluación en la etapa inicial de este (se estima que las pérdidas debidas a estos problemas pueden ser de cientos de miles de euros [Kuc, 2021]).

Por otro lado, los equipos usualmente son herméticos y sus elementos internos son de difícil acceso, lo que también dificulta su mantenimiento y la comprensión de su funcionamiento, sobre todo en lo que respecta a la capacitación de nuevo personal. Dichas actividades implican un alto riesgo para la adecuada operación de plantas y equipos, pues se pueden suscitar accidentes con pérdidas tanto materiales como humanas.

En el caso específico de una empresa de automatización con proyectos de comisionamiento en diversas áreas (principalmente en el sector automotriz, pero también en otras áreas como la construcción, la ingeniería hidroeléctrica, almacenes automatizados, entre otras), se observó que si se contaba con una herramienta para reproducir digitalmente entornos industriales reales sería posible diseñar y probar sistemas de mayor complejidad, asegurando que la operación fuera segura y eficiente en un tiempo reducido, sin tarifas extraordinarias debidas a errores cometidos en la planeación.

Por otro lado, cuando se contrataba nuevo personal era necesario familiarizarlo con los entornos industriales en que se trabaja regularmente, es decir, con los estándares de fabricación de las diversas empresas que requieren servicios de automatización. Resultaba peligroso y costoso enviarlos a proyectos reales, pues se requería de viáticos y la falta de experiencia podría ocasionar problemas, por lo que

contar con un entorno virtual para la capacitación de los nuevos integrantes permite solucionar esta cuestión.

La solución para las problemáticas antes mencionadas fue el uso del comisionamiento o puesta en marcha virtual, que consiste en utilizar gemelos digitales para reproducir procesos industriales antes de implementarlos en el mundo real. Con este software se pueden utilizar modelos paramétricos de plantas y equipos reales y observar el comportamiento de las lógicas diseñadas para el proceso, haciendo posible que se prueben los programas escritos para PLC y robots en entornos virtuales controlados y, de esta forma, detectar errores en una etapa temprana del proyecto.

El recurso elegido para llevar a cabo esta solución fue un software modular de simulación virtual, compuesto por dos aplicaciones principales, una encargada de presentar los objetos tridimensionales y ofrecer la interfaz de usuario y la otra encargada de hacer los cálculos lógicos y matemáticos necesarios para reproducir los comportamientos físicos de los objetos. La comunicación entre ellas se realiza mediante paquetes serializados en código binario, en los cuales se incluyen campos, propiedades y cualquier información útil para la correcta sincronización entre ambas. Esta aplicación puede ser extendida agregando *plugins* o complementos con funcionalidades específicas (por ejemplo, interfaces para comunicar con diferentes marcas de PLC o para trabajar con diferentes estándares de fabricación).

Al ser un software con un alto grado de complejidad y una gran cantidad de proyectos es usual encontrar fallas durante su ejecución, estas pueden ser tan variadas como errores en el almacenamiento de datos importantes para la aplicación, problemas con la interfaz de usuario, errores de multiprocesamiento o tareas paralelas (*rise conditions*), fallas en la comunicación entre los diferentes proyectos, entre muchas otras.

Es debido a esto que se estableció un equipo de desarrolladores que mediante diversas herramientas de software puedan atender de forma eficiente las problemáticas surgidas en la versión de la aplicación entregada a los usuarios. Pero no solo eso, sino que también puedan agregar nuevas funcionalidades a pedido de los clientes, por ejemplo, interfaces de procesamiento para determinadas marcas de PLC y robots, módulos para escribir lógica en un lenguaje deseado, gemelos digitales de objetos comúnmente usados en los comisionamientos reales o para prototipado, etcétera.

3.1. Contexto de participación profesional

El departamento del cual formo parte es llamado *Technische Büro 25* y está integrado por un equipo de trabajo en Alemania y otro en México. El equipo mexicano está conformado por siete *developers* o desarrolladores, dos *testers* y tres diseñadores de estaciones virtuales. Los desarrolladores somos los encargados de dar mantenimiento al software trabajando sobre el código del mismo, tanto en la corrección de fallas como en la adición de nuevas funcionalidades; los *testers* se encargan de probar que las tareas realizadas por los desarrolladores cumplan los requisitos de funcionamiento y de reportar nuevas fallas; y por último, los diseñadores de estaciones virtuales utilizan el software para crear estaciones a pedido de los clientes, siguiendo los estándares de cada empresa contratante.

Dentro de las responsabilidades que cubro en la empresa se encuentran las siguientes:

- **La implementación de nuevas soluciones para el software de simulación, tanto solicitadas por los usuarios como planeadas por el equipo de trabajo.**
  Las cuales pueden ir desde la creación de nuevas herramientas u objetos para el comisionamiento virtual o prototipado de estaciones, hasta nuevas formas de hacer que el procesamiento de los proyectos sea más eficiente y rápido, reduciendo el espacio en memoria que ocupan o la creación de complementos que ayudan a cumplir determinados estándares en la puesta en marcha de las estaciones.

- **El diseño y desarrollo de Interfaces de Usuario intuitivas, asegurando la consistencia con respecto de la vista general del programa y sus funcionalidades.**
  Estas interfaces las realizo con un *framework* que cuenta con diferentes herramientas para la creación de interfaces de usuario y usa un lenguaje de marcado Lenguaje Extensible de Formato para Aplicaciones (XAML, por sus siglas en inglés *eXtensible Application Markup Language*) para escribirse.

- **La identificación y corrección de fallas encontradas en el código del programa, empleando herramientas de *debugging* o depuración para este fin.**
  El proceso general que sigo consiste en reproducir las condiciones que desencadenaron la falla, busco en la pila de llamadas de subrutinas o *callstack* hasta encontrar el origen del problema y planifico una solución para el problema específico.
  Sin embargo, en ocasiones el error no es identificable de esta forma, pues puede deberse a rutinas ejecutándose en paralelo o en diferentes hilos (*threads*), por lo que es necesario agregar registros o *loggers* con mensajes que den información sobre como se está ejecutando la secuencia de rutinas. De igual forma, se puede utilizar software que permite observar el porcentaje de procesamiento que ocupan las rutinas y de esta forma puedo tener una idea más clara de dónde se encuentra el error.
  Otras veces, la falla se encuentra cuando la memoria reservada no es liberada por alguna rutina o subrutina (*memory leaks*) o cuando el código que contiene el error ha sido codificado para no poder ser comprendido por personas ajenas a la empresa. En estos casos, hago uso de herramientas especiales para decodificar o para analizar el comportamiento de la memoria.

- **Evaluación del funcionamiento de nuevas implementaciones.**
  Esto lo realizo mediante pruebas unitarias (*unit testing*), que es una forma de comprobar que las unidades más pequeñas que conforman el programa funcionen de forma adecuada individualmente, al hacer que cumplan con determinadas condiciones lógicas o matemáticas.

Capítulo 4

Metodología Utilizada

El software desarrollado para esta empresa es bastante flexible en cuanto a funcionalidades y ofrece una gran variedad de herramientas para adecuarse a proyectos en áreas muy diversas. Algunas de las características principales con que cuenta son las siguientes:

- Es posible para diversos usuarios trabajar en un mismo proyecto.
- Creación de lógicas para simular lógica de controladores escrita en FBD (*Function Block Diagram*) y C#;
- La conexión entre elementos, por ejemplo variables y objetos de la simulación, se puede realizar de forma gráfica y automática mediante diagramas.
- Edición de materiales, modelos, físicas y *meshes* para el entorno visual tridimensional.
- Asignación automática de entradas y salidas provenientes del PLC.
- Intercambio bidireccional de información entre archivos CAD y la simulación.
- Cálculos de física integrados, como fricción, velocidad, colisiones, etcétera
- Acoplamiento con diferentes dispositivos de comunicación en un proyecto de simulación (por ejemplo, controladores y robots).

Otra característica importante es su modularidad, esto quiere decir que está integrada por diversos componentes que se ensamblan en un núcleo principal para ofrecer al usuario la capacidad de moldear el programa de acuerdo con sus necesidades. Estos componentes se proporcionan individualmente (algunos de forma gratuita), mediante un sistema de licencias. También, es posible para los usuarios crear sus propios componentes y añadirlos al programa.

El núcleo antes mencionado consiste en un Kit de desarrollo de software (SDK, por sus siglas en inglés *Software Development Kit*) que contiene las funciones principales del programa y de las cuales todos los proyectos de la simulación se nutren. Existen, además, dos aplicaciones principales *core* y *cliente*, una encargada de los procesamientos y cálculos y la otra de presentar el renderizado de los objetos y presentar las diversas herramientas disponibles.

A continuación, se hace una descripción de los diversos proyectos y recursos que forman parte del software de simulación y su función dentro del mismo.

4.1. Kit de desarrollo de software

El SDK es un elemento común en el desarrollo de cualquier tipo de software, y consiste en un conjunto de herramientas para crear aplicaciones en alguna plataforma específica. En él se encuentran proyectos con funciones y datos fundamentales para la construcción de una aplicación, y tienen que ser accesibles para todos los proyectos que componen el sistema pues con ellos se da sustento al software.

En el caso específico del software de simulación se encuentran herramientas para realizar las siguientes funcionalidades: la serialización de paquetes para la transmisión de datos, componentes para la definición de entidades, la cámara utilizada en la escenificación del render, el *clipboard* para las acciones de copiado y pegado, la clase base para todos los objetos existentes, la definición del espacio donde interactúan dichos objetos, definición de *slots* de entradas y salidas de los objetos y la forma en que estos se asignan, los recursos para cambio de idioma en la UI, estructuras de datos especiales hechas a medida, especificación de los métodos de *undo-redo*, la creación de modelos, materiales y texturas para los objetos, los iconos utilizados en la UI, la referencia a la Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés *Application Programming Interface*) y diversas utilidades generales.

4.2. Interfaz de programación de aplicaciones

La API es también un elemento común en el desarrollo de software, esta se encarga de que diversos componentes de software se comuniquen entre sí mediante protocolos y definiciones.

El propósito de la API del software de simulación es establecer una vía de comunicación con aplicaciones externas según las necesidades de los usuarios, exponiendo elementos y funcionalidades primarias del software para así poder extender el núcleo de este a otros programas.

Por lo tanto, si un cliente desea crear sus propios objetos, lógicas o interfaces y modificarlos o eliminarlos desde fuera de la simulación, solo deberá acceder a la instancia de la API y las funciones del software que esta hace accesibles.

4.3. Aplicaciones principales

El software de simulación cuenta con dos aplicaciones principales compuestas, a su vez, de diversos proyectos. Dichas aplicaciones son las que el usuario ve instaladas en su equipo de cómputo y son las que contienen el espacio de trabajo para generar estaciones virtuales, junto con todas las herramientas de diseño, objetos, interfaces, lógicas, etcétera.

Cabe resaltar que estas aplicaciones, al igual que la mayoría de los proyectos que forman parte del sistema toman elementos del SDK para su composición.

4.3.1. Aplicación para procesamiento lógico y matemático (core)

Esta aplicación es la que se encarga de realizar los cálculos y el procesamiento de la información necesaria para que los objetos puedan interactuar entre ellos replicando el comportamiento en el mundo real.

Contiene también, módulos con los protocolos para comunicarse con PLC y robots de diversas marcas, con otros sistemas como MatLab y con bases de datos SQL.

Por otro lado, el core puede considerarse como el punto central de conexión de todas las partes (módulos y componentes) que permiten que los objetos funcionen como gemelos digitales.

4.3.2. Aplicación para visualización 3D (cliente)

En esta aplicación se muestra la vista 3D en la que se pueden observar todos los objetos, además de los controles con los cuales es posible comunicarse con el core. Representa la interfaz de usuario principal en la que el usuario construye estaciones.

En el cliente existen diversos módulos con propósitos diversos, por ejemplo: para procesar archivos CAD, grabar secuencias de movimiento dentro del 3D, crear lógicas en FBD o C#, utilizar Realidad Virtual (VR, por sus siglas en inglés *Virtual Reality*), definir elementos de control vía archivos XML, simular la logística para un depósito, simular monorrieles o *conveyors*, combinar diversos objetos y definirlos como objetos nuevos, entre otros.

De igual manera, es posible contar con más de un cliente conectado a un core, por lo que varias personas pueden trabajar en un mismo proyecto asignando, incluso, diferentes niveles de acceso a cada uno. La API se comunica con el core como si fuera un cliente que no cuenta con interfaz de usuario.

4.4. Módulos

Como ya se mencionó, una de las características más importantes de este software es su estructura modular, esto es, su capacidad de integrar diversas partes y eliminar otras según sea necesario. Estas partes son llamadas módulos y su propósito es muy variado, por ejemplo, el cliente y el core contienen módulos especiales que les otorgan las características antes mencionadas.

Para poder lograr que la aplicación sea extensible a través de los módulos se utiliza una tecnología llamada *Managed Extensibility Framework (MEF)*, que es una librería que permite agregar componentes sin modificar el código fuente de la aplicación. Esto lo logra *componiendo* o construyendo los diversos módulos a partir del código del cual depende, es decir, sus dependencias (en este contexto llamadas *imports*) y sus funcionalidades principales (*exports*), exponiéndolas o haciéndolas accesibles en tiempo de ejecución. Utiliza *metadata*[1] para que los módulos sean *mostrados* sin necesidad de generar instancias de ellos o cargar sus *assemblies*, por lo que el componente no requiere de dependencias fuertes.

MEF utiliza internamente el concepto de reflexión para acceder a los *imports* y *exports* de los componentes. Este concepto implica la habilidad de una aplicación de modificar su comportamiento y

[1] Definida como un grupo de datos que describen el contenido informativo de un objeto al que se denomina recurso.

acceder a su información en tiempo de ejecución.

Al cargar la aplicación se realiza la composición de todos los módulos, que en el caso particular de la simulación, algunos de ellos contienen una instancia pública y estática de sí mismos (patrón *singleton*), por lo que son accesibles desde cualquier parte de la aplicación de la que forman parte (core, cliente, etcétera).

Existen módulos que son propios del cliente, otros propios del core y otros que pertenecen a ambos (incluyendo aquellos que son creados por usuarios). También, existen módulos persistentes, es decir, aquellos que generan archivos para almacenar información y después poder cargarla al iniciar un proyecto.

Por último, existe un tipo especial de módulos llamados complementos o *plugins* que pueden, de igual forma, ser de cliente, core o persistentes.

4.5. Complementos

Los complementos son elementos que permiten agregar funcionalidades específicas al sistema. Por ejemplo, interfaces de comunicación con PLC o robots, proveedores de lógica en diversos lenguajes, nuevos tipos de objetos especiales, manejo de archivos, entre muchos otros.

En el software de simulación existen tres tipos principales de *plugins*, listados a continuación:

4.5.1. Complementos proveedores de interfaces

Este tipo de *plugins* están vinculados al core y son al mismo tiempo persistentes, pues los datos de cada instancia de estas interfaces se guarda en archivos dentro de la carpeta que contiene el proyecto de la estación virtual. Son uno de los elementos principales para poder realizar el comisionamiento virtual, pues son los módulos que permiten vincular la simulación con controladores y robots reales. Dentro de estas interfaces, es posible definir tablas con una gran cantidad de variables de diferente tipo, siguiendo el estándar de cada marca de PLC o robot.

La comunicación entre estos módulos y el exterior se realiza mediante el protocolo TCP, de esta forma se asegura que la información llegué íntegra entre ambos extremos del canal. Es necesario contar con *drivers* específicos de cada PLC o robot con los que se establece conexión mediante el *host* y el puerto del software.

4.5.2. Complementos proveedores de lógica

Estos complementos sirven para crear diferentes tipos de lógicas (C#, FBD, FMU, entre otras) con las cuales se pueden simular controladores para manipular los objetos virtuales. Se encuentran vinculados con el cliente y también son persistentes, lo que quiere decir que se generan archivos de lógicas (con las diferentes versiones creadas) dentro de la carpeta del proyecto.

Otro punto interesante es que las lógicas se pueden vincular a objetos mediante un objeto especial llamado *logic object*, que en términos simples es una representación virtual de un controlador. La conexión se realiza mediante *slots*, que no es otra cosa que puntos de lectura y escritura de tipos de datos diferentes. Este proceso se puede realizar mediante una interfaz gráfica con la que se generan diagramas donde los nodos de conexión son objetos (*slots*), lógicas o compuertas lógicas.

4.5.3. Complementos de aplicación

Estas son aplicaciones que proveen funcionalidades muy específicas y también son persistentes. Algunos ejemplos son: el sistema de diagramas mencionado arriba, el complemento para agregar un sistema de monorrieles eléctricos, el complemento para manipular gabinetes virtuales, el módulo para grabar escenas de la simulación, el complemento para acceder a una librería online con recursos adicionales, el módulo para agregar transportadores *Power and Free* y la librería, uno de los *plugins* principales, que permite guardar y cargar plantillas en archivos especiales de conjuntos de objetos como si fueran nuevos objetos, lo que permite reutilizar estaciones completas o partes de estas en proyectos diferentes.

4.6. Comunicación entre aplicaciones

La forma en que las aplicaciones principales se comunican entre sí es mediante la serialización binaria de paquetes con las propiedades que se desea transmitir. Esta es similar a la realizada con Protobuf, sin embargo, tiene la particularidad de definirse mediante *structs* en vez de clases como los demás elementos de la simulación.

La razón principal reside en una optimización del *garbage collector*[2] en el uso memoria *heap*[3]. El *garbage collector* verifica que no existan instancias de las clases en dicha memoria, para entonces eliminar la referencia a esa clase y liberar espacio. Sin embargo, como la *struct* se almacena en la memoria *stack*[4] entonces para el *garbage collector* es invisible (esto es verdad siempre y cuando la *struct* no sea una propiedad de una clase). De esta manera, se evita que el proceso liberación de memoria produzca que el flujo de la simulación se ralentice, debido a la gran cantidad de paquetes utilizados.

Otro aspecto importante en la comunicación interna del sistema, es el uso de la tecnología RPC, con la cual es posible comunicar a los diversos clientes con el core y esperar una respuesta íntegra de este. Esto asegura la consistencia entre los mensajes del core con cualquier cliente (incluida la API).

Por otro lado, para establecer la comunicación entre el core y los clientes se utiliza el protocolo TCP, donde los clientes contienen una instancia de TCP *client* y el core una instancia de TCP *server* con un host y un puerto definidos, al cual se pueden conectar los clientes interesados.

[2]El cual es un administrador de la memoria RAM utilizada.

[3]Una zona de almacenamiento dinámica en la que se contienen las referencias de clases y variables sin un orden establecido.

[4]Una parte de la memoria en donde se sigue el principio de *Last In First Out*, las variables guardadas aquí son accesibles directamente y no por referencia, y se liberan al final de la ejecución del programa.

4.7. Creación de objetos virtuales

4.7.1. Objetos virtuales

Los objetos que se muestran en la aplicación para simular equipos y diversas herramientas presentes en las estaciones reales son llamados *scene objects*, cuya clase base se encuentra albergada en el core, y contiene las propiedades principales que todo objeto (gemelo digital) debería contener, por ejemplo, un nombre y un id que lo identifican, la lista de componentes que le dan funcionalidad, una serie de funciones que serán actualizadas en cada ciclo de ejecución, la lista de hijos [5], entre otras más específicas según objeto del cual se trata (botón, grúa, banda transportadora, etcétera).

Si bien los objetos se encuentran en el core es necesario poder trasmitir la información que los define a hacia el cliente, pues de esta forma será posible plasmarlos como modelos tridimensionales o proyectarlos como *view models* y mostrar sus propiedades en las *views* de la interfaz de usuario (tablas, custom controls, ribbons managers). Esto se logra mediante la serialización y transmisión de paquetes mencionada en la sección anterior.

Otra particularidad de los *scene objects* es que utilizan el patrón de diseño *flyweight* para su generación. A grandes rasgos, consiste en abstraer las propiedades que un tipo de *scene object* puede tener en común en las diversas instancias que puedan existir de este (*intrinsic data*), por ejemplo, propiedades relacionadas al aspecto del objeto. Después, se construye un diccionario donde el valor es la instancia del objeto y *key* o clave[6] es generada a partir de la *intrinsic data*[7].

Con esta información, cada que se desee una instancia de un *scene object* con una determinada *intrinsic data* se busca si existe ya en el diccionario, si es así se reutiliza dicha instancia, de otro modo, se crea la instancia y se agrega al diccionario para futuras peticiones. Todas las propiedades que pueden variar entre instancias se conocen como *extrinsic data* y se les asigna valor en otro momento, un ejemplo de esto puede ser la información de la posición del objeto en el mundo. De esta manera, se puede manejar una gran cantidad de objetos minimizando el uso de memoria.

4.7.2. Componentes

La creación de objetos o entidades sigue el paradigma ECS, por lo que cada *scene object* se sirve de componentes (en forma de *metadata*) para definir muchas de sus principales características.

Algunos de los componentes principales son: el componente para generar los modelos 3D de los objetos y hacerlos visibles, el que agrega colisión entre objetos para interactuar en el mundo físico virtual, el que permite la generación de tablas a partir de sus propiedades y así poder ser editado desde alguna vista, el que permite seleccionarlos en el mundo, el que añade la serialización - deserialización de ellos al cargar o guardar un proyecto o al hacer una acción de copiado y pegado.

[5] Los objetos se pueden organizar en una estructura jerárquica como padre-hijos.
[6] Un diccionario se define como una colección de pares clave-valor en la que cada clave o *key* se asocia a un único valor.
[7] La *key*, generalmente, es una *string* compuesta por la descripción de las diferentes propiedades de la *intrinsic data*.

4.7.3. Serializador

El serializador es una herramienta muy importante en la manipulación de objetos dentro de la simulación, permite guardar y cargar las definiciones de los objetos, además de sus relaciones entre ellos y con el mundo. Utiliza las propiedades que le brinda la clase *XElement*, que como su nombre indica, representa un objeto en el lenguaje XML. Con esta clase se pueden crear elementos XML, modificar su contenido, agregar, eliminar y cambiar los elementos hijos, agregar atributos y serializar el contenido en forma de texto.

De esta forma el serializador puede guardar las definiciones de los *secene objects* en archivos dentro de la carpeta del proyecto correspondiente. Esto es verdad también para otros elementos del sistema, como las interfaces utilizadas o las lógicas creadas (*proveedores de interfaces y lógicas*). De hecho, por cada guardado del proyecto se generan archivos que representan un estado de la estación en una fecha determinada. Los proyectos usan estos archivos para cargar el último estado del proyecto. El serializador también permite realizar acciones de copiado - pegado y *undo - redo*.

4.8. Interfaz de usuario

Para la creación de vistas en la interfaz de usuario se utiliza el lenguaje XAML y se sigue el patrón de diseño MVVM para vincular los datos y funcionalidades a dichas vistas.

4.8.1. Propiedades de dependencia

Dentro de este paradigma, y como parte del funcionamiento de XAML, existe un tipo de propiedades especiales llamadas propiedades de dependencia o *dependency properties*. Una de sus principales características es que no existen como miembros de una clase particular, sino que se encuentran alojadas en un diccionario proporcionado por la clase *DependencyObject* en donde la *key* es el nombre de la propiedad y el valor es el valor que se desea asignar a la propiedad. Solo en el caso en que el valor por defecto (*default*) es cambiado por otro este se guarda en la instancia del objeto, lo que ayuda a ahorrar mucho espacio en memoria. De esta forma, a pesar de que existan múltiples instancias del objeto que contiene la propiedad su valor no es almacenado en cada instancia, sino que se obtiene dinámicamente a través de una llamada a un método llamado *GetValue()*.

Este método busca jerárquicamente dentro de los elementos de la interfaz de usuario hasta encontrar un valor (si no encuentra ninguno toma el valor *default*).

Otra característica importante es el mecanismo de notificación que permite saber cuando el valor de una propiedad ha sido cambiado, ya sea desde la *view* hacia el *view model* o viceversa.

4.8.2. Gestión de recursos

La aplicación utiliza recursos que establecen una cultura particular del sistema[8]. Esto implica que se puede ajustar el lenguaje en el que se desea trabajar, la configuración de las fechas, el formato de los números, el ordenamiento de las *strings*. Esto se logra con una clase especial llamada *CultureInfo*, que

[8]Donde cultura se refiere las costumbres, lenguaje y construcciones sociales de un determinado país

especifica las reglas de una cultura particular.

Los recursos de cada cultura se guardan en archivos con extensión *resx* basados en XML, cada recurso cuenta con un valor y una *key* de identificación con la cual acceder al valor. Los valores pueden ser cualquier tipo de dato serializable, desde *strings* hasta imágenes svg, por lo que los recursos se pueden usar tanto para traducciones como para asignar iconos en las vistas.

Por último, para acceder a estos recursos en tiempo de ejecución se cuenta con *ResourceManager*, una clase especial con la cual manipular los archivos de recursos para, por ejemplo, encontrar o guardar determinados elementos.

4.9. Sistema de renderizado

El sistema de renderizado del software de simulación está contenido en un proyecto especial, en él se encuentran las herramientas necesarias para poder representar diferentes modelos con texturas y materiales diversos. Como se mencionó anteriormente, los *scene objects* cuentan con componentes que les agregan funcionalidades, por lo que los objetos que deben de ser renderizados contienen un *transform component* y un *model component*. El primero de ellos es el encargado de establecer la posición del objeto en el universo, mientras que el segundo es el encargado de brindar la información sobre el modelo, su textura y el material del que está hecho para que con esta información el motor de renderizado pueda generar representaciones tridimensionales precisas.

El *transform component* contiene una matriz con información sobre la posición, rotación y orientación del objeto, la cual es utilizada por el motor de renderizado para ubicar el objeto en el espacio una vez dibujado el modelo con ayuda del *model component*. Cuando el *scene object* es agregado al render es ubicado en las coordenadas de origen y después reubicado y rotado según sea el caso.

Existe una clase especial, llamada *ContentManager*, encargada de buscar recursos (archivos) en el sistema que contienen la información sobre texturas, materiales y los vértices para generar un modelo. Estos archivos son especiales y sirven para compartir datos sobre animaciones y la geometría 3D del objeto, contienen información para que los *shaders* puedan renderizar los materiales y las texturas. No obstante, la simulación trabaja con un tipo de archivos propio para este fin, por lo que existen herramientas para su conversión durante los procesos de importación y exportación.

4.10. Motor de físicas

Los objetos dentro de la simulación deben de interactuar conforme a las leyes de la física del mundo real, por lo que el software cuenta con un motor de físicas de propósito general llamado *Bepu*. Dispone de diversas formas geométricas y cálculos de colisiones especiales para cada forma, algunas de ellas son: esferas, cápsulas, cajas, triángulos, cilindros o envolventes convexas.

La detección de colisiones, tanto linear como angular, utiliza los métodos de Minkowski para determinar las posibles colisiones. La forma en que se logra esto es generando grupos de objetos cercanos que

podrían colisionar, llamados volúmenes[9]. Una vez obtenidos estos grupos se calcula la figura resultante al superponer y restar geométricamente las figuras unas con otras, esto genera un espacio auxiliar de dimensiones tridimensionales donde si el resultado coincide con el origen significa que existe colisión (la distancia entre ellos en algún punto es igual a cero).

Para conocer el punto exacto de la colisión se determinan las posiciones de los objetos que van a colisionar en un intervalo de tiempo en el futuro (*sweep queries*). A partir de esta información, se deduce cuál es la intersección entre un vector normal a la superficie de contacto de uno de los objetos y el otro objeto. Después, en el instante de tiempo actual (dentro del ciclo de ejecución), se hacen cálculos para obtener los impulsos mecánicos resultantes de la colisión a través del tiempo, de donde luego es posible obtener la fuerza que actúa sobre los cuerpos. Hay que destacar que se toman en cuenta algunas variables como la fricción para obtener el movimiento final de los objetos. Todo esto se realiza dentro de un ciclo de ejecución menor a 1/60 de segundo (relacionado con la frecuencia de refresco de los monitores).

4.11. Opciones del entorno

El sistema de *settings* u opciones del entorno permite que se ajusten diversas características del sistema. Se compone principalmente de tres módulos, uno para *settings* de core, otro para *settings* de cliente y otro para *settings* que afectan a proyectos individuales. Sin embargo, existe un módulo base, del que heredan los anteriores, y que genera dinámicamente las instancias por reflexión. Algunos ejemplos de opciones disponibles son: para definir el aspecto del renderizado, para ajustar la cámara, para definir características del proyecto y para establecer variables del motor de físicas.

Los *settings* se vinculan a una vista en el core y una vista en el cliente mediante una serie de *view models* jerarquizados, los *settings* de core son visibles en el cliente mientras que los de core solo son visibles en este. Los que son propios de un proyecto solo son visibles cuando un proyecto está abierto.

[9]Estos forman en sí mismos figuras tridimensionales, por lo que el algoritmo puede ser aplicado sobre ellos también.

Capítulo 5

Resultados

El software de simulación, es usado por múltiples empresas de diversos sectores industriales como una solución para sus proyectos de automatización debido a su flexibilidad. Algunos de los principales campos de aplicación y los beneficios que el software les brinda, son los siguientes:

5.1. Campos de aplicación

5.1.1. Industria automotriz

Los proyectos dentro de este sector incluyen la producción de componentes automotrices, la construcción de chasis, pintado y ensamblado. Todas estas actividades pueden ser simuladas mediante gemelos digitales dentro del software cumpliendo, además, con los estándares solicitados por la Industria.

Los requerimientos en la industria automotriz son grandes, y cada vez más complejos, pues constantemente se diseñan nuevos modelos y se desea que los ciclos de producción sean menores, por lo que gracias al software de simulación se han podido realizar los proyectos respetando estas características.

Las soluciones disponibles en el software incluyen:

- Ensamblado de sistemas.
- Transportadores eléctricos (SKID, *power & free*, EHB).
- Sistemas de depósitos.
- Sistemas de apilado.
- Construcción de cuerpos de vehículos mediante robots.

5.1.2. Automatización industrial

La industria de la manufactura está adoptando los sistemas de simulación como parte esencial en la gestión de sus proyectos de automatización.

Con el software de simulación se ha logrado que el *hardware* para proyectos de automatización funcione sin errores y que el software de automatización de PLC y robots esté correctamente programado en términos de control y operabilidad. Algunas de las soluciones ofrecidas para este fin son las siguientes:

- Máquinas para procesos de ensamblado y producción.
- Instalaciones robóticas.
- Máquinas para trabajar metal.
- Estaciones de trabajo
- Sistemas de transportadores y depósitos interconectados

5.1.3. Logística

Las compañías necesitan obtener procesos de logística que permitan una excelente planeación y procesos sin contratiempos, por lo que el software de simulación brinda componentes tales como bandas transportadoras, depósitos automáticos y sistemas de transporte automáticos.

Los sistemas de almacenamiento requeridos son cada vez más complejos debido al incremento en la variedad y cantidad de productos. Es por ello que el software de simulación, en conjunto con los sistemas de control, permiten establecer procesos de almacenamiento y relocalización eficientes, considerando la mayor cantidad de posibles variaciones en la configuración. Esto ha permitido simular depósitos con hasta 100 000 espacios de almacenamiento.

El software ofrece una gran variedad de soluciones para cubrir los diferentes requerimientos logísticos:

- Sistemas de almacenamiento automático.
- Transportadores de acumulación, de alta velocidad, de contenedores y de palets.
- Transelevadores.
- Clasificadoras.
- Sistemas de paletización.
- Monorriel eléctrico.
- Sistemas de apilado.

5.1.4. Ingeniería mecánica

El software de simulación permite detectar desde una etapa temprana del desarrollo del proyecto las colisiones de maquinaria, que pueden ser muy costosas y consumir mucho tiempo. Por otro lado, es posible analizar y verificar programas de Control Numérico (NC, por sus siglas en inglés *Numerical Control*) antes de que las máquinas sean terminadas.

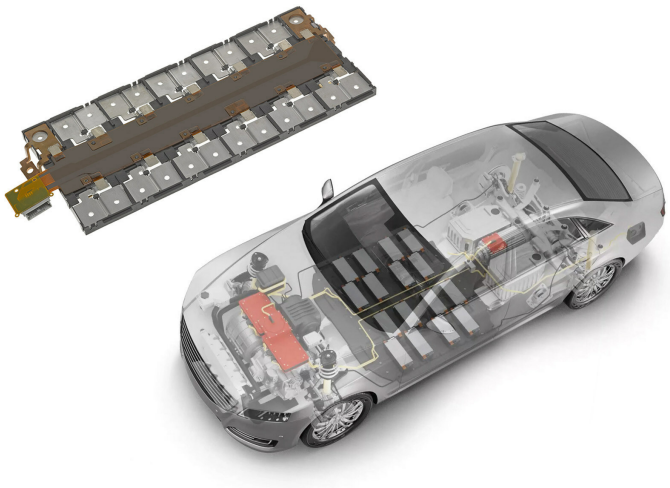

Figura 5.1 Placas con pistas de conexión de baterías eléctricas.[1]

5.2. Caso de uso del software de simulación

El software de simulación se utilizó para realizar pruebas de diseño y ejecución de un proyecto para una empresa automotriz muy importante. Se trata de una estación para el ensamble de baterías para automóviles eléctricos. A continuación, se describirá una sección de la estación para mostrar un caso de uso real del software, explicando que *scene objects* fueron utilizados y cuál fue su función.

El proceso en cuestión, es el transporte y ensamblaje de las tapas superiores de los bancos de baterías que contienen las pistas de conexión para las celdas, la Figura 5.1 muestra un ejemplo de estas.

El primer paso para iniciar la construcción del gemelo digital de la estación es importar archivos CAD y algunos modelos de maquinaria ya ensamblados para posicionarlos dentro del universo con ayuda de un archivo 2D colocado sobre el piso, en él se encuentran las ubicaciones de los equipos y sirve como guía para la generación de la estación virtual. La Figura 5.2 presenta una imagen de una estación de ensamblado de baterías eléctricas real, similar a la que se intenta simular en este proyecto.

Una vez colocados los elementos, la primera etapa del proceso consiste en recibir las pistas de conexión, montadas en un soporte rectangular llamado *skid*, ambos objetos son de tipo *decorator* cuya función principal es mostrar un modelo tridimensional que represente un objeto real. Para ser agregados al mundo, se utiliza un objeto especial llamado *sequence inserter* que serializa los objetos a partir de una plantilla basada en el formato XML. La inserción se realiza conectando el *sequence inserter* con un botón para ser activado mediante diagramas de conexión, en donde, en un entorno gráfico, se unen los *slots* (es decir, las ranuras de entrada y salida de datos) de estos objetos representados por bloques en el diagrama.

Una vez agregado el *skid* y las pistas de conexión en el mundo, son posicionados dentro de un elemento llamado *reader* para detectar el tipo de conexión de la que se trata y así poder asignar

[1]La baterías para coches eléctricos ahora cuentan con conexiones en la tapa superior, evitando así el cableado excesivo.
[2]Los modelos CAD permiten simular una estación de transporte y ensamblado de baterías eléctricas.

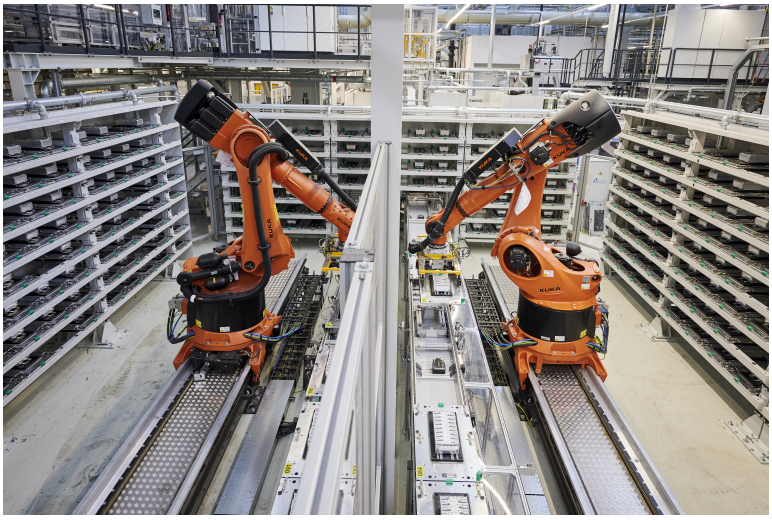

Figura 5.2 Estación de ensamblado de baterías eléctricas.[2]

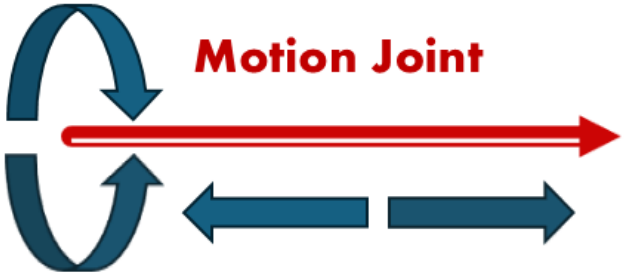

Figura 5.3 El *motion joint* permite simular movimientos lineales y rotatorios.[3]

un número serial acorde y distinto para cada elemento agregado. Cada *skid* contiene una etiqueta o *tag*, un tipo de variable que permite reconocer un objeto específico, el *reader* reconoce el *tag* y con un *logic object* (elemento que simula un controlador y al que se le puede asignar una lógica con entradas y salidas) se lee el tipo de conexión de la que se trata y se asigna un número serial según sea el caso, aumentando en uno dicho número cada que se agrega uno nuevo para que no se repitan.

Para mover los *skid* y las pistas de conexión que están sobre ellos se utilizan robots de seis ejes, que son representados mediante objetos virtuales llamados *CAD assembly*, similares a los *decorators* con la diferencia de que su modelo y su componente de transformación (que define sus coordenadas principales) viene de un modelo CAD. Cada *CAD assembly* describe un eje del robot y el actuador al final de este, que en este caso específico se trata de un *gripper* o pinza de agarre. Los *CAD assembly* están unidos entre sí mediante otro objeto llamado *motion joint*, que realiza la función de una articulación a la que se le pueden determinar las posiciones de rotación máximas y mínimas que puede alcanzar. El *gripper* cuenta con dos elementos de agarre llamados *clap* con un *motion joint* cada una. En la Figura 5.3 se pueden observar los movimientos que puede tener un *motion joint*.

[3]El *motion joint* es un elemento muy importante, pues permite agregar movilidad a los elementos de la estación.

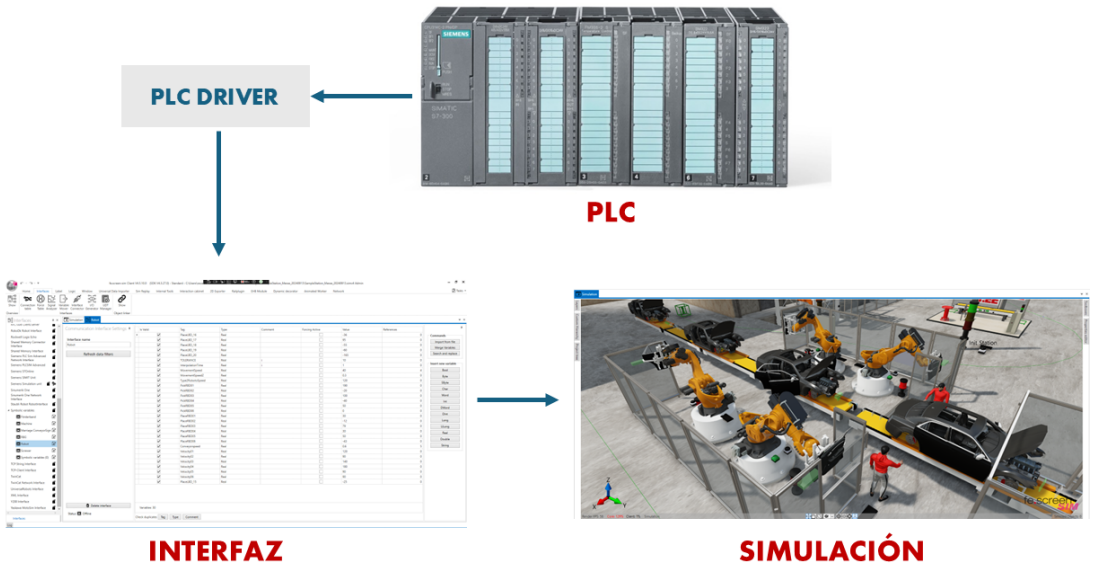

Figura 5.4 Esquema de conexión entre un PLC y el software simulación.[4]

A cada robot se le asignan variables de entrada y salida mediante *plugins* o componentes especiales llamados interfaces. Con dichas interfaces también se definen entradas y salidas para el PLC que hace el control de la estación y para la conexión TCP entre la simulación y el PLC, para este proyecto se optó por un controlador virtual *Plcsim Advanced*. La Figura 5.4, muestra el esquema general de conexión entre un PLC y el software de simulación.

Para representar la trayectoria que debe seguir cada robot se utiliza un objeto llamado *table data* con el cual se puede cargar información de las posiciones deseadas desde un archivo externo con esos datos y un objeto que simula un *HMI* con controles para definir posiciones del robot a partir de la información del *table data*, esto puede ser de forma automática o manual.

El robot simula la recolección y depósito del *skid* y las pistas de conexión mediante un elemento llamado *pick and place*, este utiliza una variable llamada *mark* o marca que le permite identificar un objeto específico. El *gripper* y el *skid* cuentan con una marca respectiva, que mediante lógica (asignada por un *logic object*) hacen que el *pick and place* modifique la jerarquía de estos objetos, el *skid* se vuelve *hijo* del *gripper*. Esto es posible porque los objetos en el universo siguen una estructura jerárquica, donde el Universo es el objeto *padre* de todos, y como los objetos *hijos* siguen la posición del *padre* se crea el efecto de que el *gripper* traslada el *skid* al moverse. De igual manera, las marcas permiten que mediante la lógica se manipule la rotación de los *motion joint* de las *claps* y se simule el movimiento de agarre.

Este proceso se repite, pero en sentido inverso, para depositar el *skid* y las pistas de conexión en una base metálica con varios pisos representada por un *decorator*. Cada piso de esta base cuenta con un

[4]La interfaz permite gestionar las variables de entrada y salida del PLC conectado a la simulación, además de fungir como la conexión entre la lógica del controlador y los gemelos digitales. Para PLC reales es necesario conectar a mediante un *driver*.

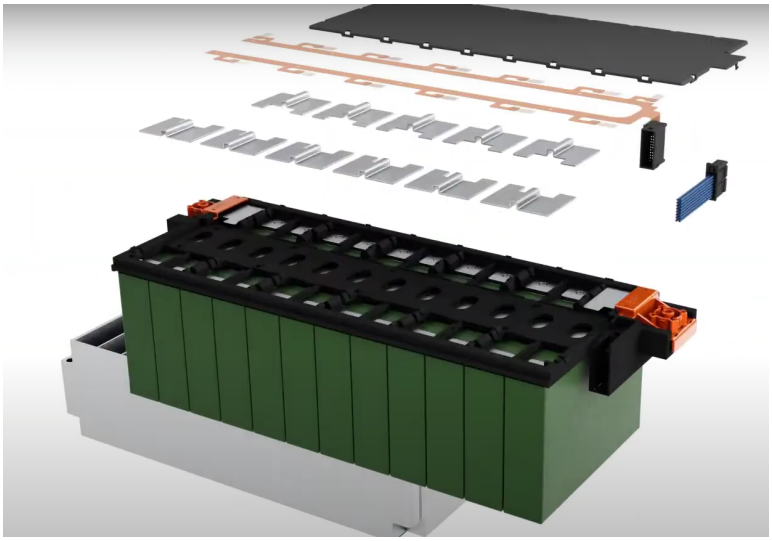

Figura 5.5 Vista explosionada de batería.[6]

objeto genérico llamado *bumper* que, a su vez, contiene una marca para que el *pick and place* haga que el *skid* sea su *hijo* y de esta forma, "liberándolo¨ del *gripper*, para que permanezca en esa posición al moverse el robot.

Otra etapa representada en esta estación es el ensamblaje de las pistas de conexión sobre un banco de baterías. Utilizando los mismos procedimientos descritos anteriormente se mueven con los robots las pistas de conexión, separándolas del *skid* y dejándolo sobre una base metálica. Se simula la adición de pegamento con un par de robots, donde el actuador de uno de ellos es una brocha y se coloca sobre el banco de baterías. Se necesitan cuatro pistas de conexiones para cubrir la superficie superior del banco de baterías. La Figura 5.5 permite visualizar como es el ensamblaje de una batería.

El banco de baterías se representa con un objeto llamado *payload* (cargado con un modelo tridimensional que emula dicho elemento), esto es así porque el *payload* responde de forma especial a la física del mundo, ya que cuenta con un *collider*[5]. Gracias a ello, se le pueden definir parámetros como la cantidad de rebote, el peso o la fricción estática y cinemática, además, si no cuenta con una superficie de soporte tiende a moverse en caída libre.

El banco de baterías se coloca sobre una *surface*, que no es otra cosa que una banda transportadora. Esta *surface* tiene como *hijo* un *motion joint* que actúa linealmente moviéndose de arriba a abajo simulando un elevador. Al llegar a la parte superior un robot coloca las pistas sobre el banco de baterías y, al igual que en el caso de las *clamp* del robot, unas *clamp* estáticas se cierran para ejercer presión entre las pistas y el banco de baterías para terminar de adherirse. Mediante lógica se establece que las *clamp* solo cierran si el *motion joint* del *surface* está en su punto máximo y se detecta un *payload* sobre esta. El elevador baja y el banco de baterías queda suspendido gracias a las *clamp*.

[5]El cual es un elemento que permite al objeto responder a las colisiones de forma realista.

[6]El ensamblaje en la estación aquí descrita consiste en tomar la parte superior metálica de la batería en conjunto con las pistas y colocarlas sobre el banco de baterías adhiriendo pegamento. Esta imagen solamente es explicativa y no representa las dimensiones del modelo usado en el proyecto.

Para simular este estado se usa un objeto llamado *bumper*, similar a un *payload*, pero con la particularidad de que contiene un *slot* para activar o desactivar su *collider*. En este caso, el modelo del *bumper* es invisible y se ubica en la misma posición que la altura máxima que alcanza el *motion joint* con la *surface*. Con ayuda de una lógica, se determina que si se detecta que las *clamp* se han cerrado se active el *slot collider* del *bumper*. Entonces, el *bumper* actúa como superficie de contacto y el banco de baterías queda suspendido sobre este, dando el efecto de que las *clamp* lo están sosteniendo. Cuando las *clamp* se abren y la *surface* vuelve a estar en la posición más alta, se desactiva el *collider* del *bumper* y el banco de baterías cae sobre la *surface* para bajar junto con ella.

Cuando se terminan de colocar las conexiones del banco de baterías, el *surface* elevador regresa a su posición más baja y se conecta con una pista de *surfaces* con sensores que ayudan a trasladar el banco de baterías hasta el lugar donde se encuentra un *reader* que determina (con ayuda de la información contenida en el PLC) si este pasa a la siguiente estación o si es enviado con las baterías defectuosas. Esta bifurcación se realiza con una mesa giratoria, que de forma similar al elevador, cuenta con un *surface* y un *motion joint* como *hijo*, solo que en este caso este último se configura con movimiento rotacional, por lo que sirve de eje para la mesa giratoria. Para este procedimiento también se utiliza un *logic object* y diagramas de conexión para interconectar todos los elementos y la lógica.

Capítulo 6

Conclusiones

El software de simulación ha demostrado ser indispensable en el proceso de comisionamiento de proyectos, permitiendo que estos sean más óptimos y menos costosos en menor tiempo, en contraste con la forma tradicional de puesta en marcha. Dado su carácter modular, que lo hace muy flexible para adaptarse a diferentes proyectos de automatización, existe una enorme complejidad en su desarrollo, por lo que, constantemente pueden surgir contratiempos que deben de ser atendidos. De igual forma, para poder cumplir con los requerimientos solicitados por los usuarios, continuamente se agregan nuevas herramientas para extender las funcionalidades.

No obstante, existen problemas inherentes a la naturaleza de esta nueva tecnología, por ejemplo, los controladores deben de conectarse a las interfaces del software mediante *drives* específicos de cada marca que a veces son difíciles de obtener. Otro problema propio del software es que, a pesar de su carácter flexible, es difícil integrar los estándares de nomenclaturas de diversas empresas, por lo que una homologación de los sistemas de comisionamiento virtual y gemelos digitales es primordial para evitar errores.

De igual forma, hay otros problemas relacionados con el contexto actual en el que se utiliza esta tecnología. Uno de ellos es la falta de formación académica especializada en el área de comisionamiento virtual, pues aún es difícil encontrar en los planes de estudio de las licenciaturas relacionadas con la automatización materias que preparen especialistas en dicha área.

Por otro lado, el costo de este tipo de software es muy elevado por lo que, generalmente, este es utilizado por empresas que cuentan con el suficiente capital para adquirirlo, lo que podría excluir del mercado de la digitalización a empresas de menor tamaño. Así pues, una versión más ligera y a la vez más económica del software que cuente con todas las herramientas, pero con ciertas limitaciones (por ejemplo en el tamaño de las estaciones o el número de objetos usado) podría mitigar este problema.

Por último, se puede mencionar que existen nuevas tecnologías que podrían brindar un mayor alcance a la simulación, por ejemplo, *chatbots* con los cuales el usuario pueda describir una estación y que sea creada automáticamente. Otra integración interesante es la de usar *Universal Scene Description (OpenUSD)*, un ecosistema para generar gráficos 3D compuestos por una gran cantidad de datos, componerlos, simularlos y manipularlos colaborativamente, brindando un lenguaje común para este propósito.

Referencias

[Baur y Wee, 2015] Baur, C. & Wee, D. (2015). Manufacturing's next act. *McKinsey and Company*. (Citado en página 2.)

[Chaudhari, *et al.*, 2021] Chaudhari, P., Utgikar, R., Kelkar, B., & Borse, P. (2021). A novel approach: Bioeconomy and industry 5.0 enhanced version. In *2021 IEEE Pune Section International Conference (PuneCon)* (pp. 1–6). (Citado en página 4.)

[CIMdata, 2020] CIMdata (2020). Speeding your digital transformation journey. private communication. (Citado en página 11.)

[CIMdata, 2021] CIMdata (2021). Digital transformation: Driving competitive advantage. private communication. (Citado en página 1.)

[Dang, 2023] Dang, T. (2023). Revolutionize your code: The magic of data-oriented design (dod) programming https://www.orientsoftware.com/blog/dod-programming/. (Citado en página 21.)

[Dineshchandgr, 2022] Dineshchandgr (2022). What are protocol buffers and why they are widely used? https://medium.com/javarevisited/what-are-protocol-buffers-and-why-they-are-widely-used-cbcb04d378b6. (Citado en página 20.)

[Dollard, 2024] Dollard, K. (2024). Serialización (visual basic) https://learn.microsoft.com/es-es/dotnet/visual-basic/programming-guide/concepts/serialization. (Citado en página 20.)

[Explainers, 2023] Explainers, M. (2023). What is digital transformation? *McKinsey and Company*. (Citado en página 1.)

[Gaiardelli, *et al.*, 2021] Gaiardelli, S., Spellini, S., Lora, M., & Fummi, F. (2021). Modeling in industry 5.0: What is there and what is missing: Special session 1: Languages for industry 5.0. In *2021 Forum on specification and Design Languages (FDL)* (pp. 01–08). (Citado en página 3.)

[Gillis, 2021] Gillis, S. A. (2021). Vwhat is object-oriented programming (oop)? https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP. (Citado en páginas 15 y 16.)

[Gombert, 2020] Gombert, L. H. (2020). How unity dots paves the way for the future of game development https://gombertleonard.medium.com/how-unity-dots-paves-the-way-for-the-future-of-game-development-4b71a3d2532. (Citado en página 21.)

[Grieves, 2015] Grieves, M. (2015). Digital twin: Manufacturing excellence through virtual factory replication. (Citado en página 9.)

[Hoffman, 2012] Hoffman, N. (2012). Background: Physics and math of shading https://api.semanticscholar.org/CorpusID:1311637. (Citado en páginas 23 y 24.)

[Ipacs, 2023] Ipacs, D. (2023). Physics engine: A key component of game engines https://bluebirdinternational.com/physics-engine. (Citado en página 21.)

[Isaac, 2020] Isaac, A. (2020). Deserialization vulnerability protection in java https://medium.com/tech-learnings/serialization-filtering-deserialization-vulnerability-protection-in-java-349c37f6f416. (Citado en página 20.)

[Kouraklis, 2016] Kouraklis, J. (2016). *MVVM as Design Pattern*, (pp. 1–12). Apress: Berkeley, CA https://doi.org/10.1007/978-1-4842-2214-0_1. (Citado en páginas 18 y 19.)

[Kuc, 2021] Kuc, A. (2021). Virtual commissioning - software testing not just in it https://www.nearshore-it.eu/articles/technologies/virtual-commissioning-software-testing/. (Citado en páginas 13 y 25.)

[Lee y Park, 2014] Lee, C. G. & Park, S. C. (2014). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering*, *1*(3), 213–222, https://doi.org/https://doi.org/10.7315/JCDE.2014.021 https://www.sciencedirect.com/science/article/pii/S2288430014500292. (Citado en páginas 12, 13, 14 y 15.)

[Mauricio y Alberto, 2022] Mauricio, S. & Alberto, M.-R. R. (2022). Tecnologías clave para la transformación digital en las organizaciones. In *Transformación digital en las organizaciones* chapter 2, (pp. 31 – 75). Universidad del Rosario, 1 edition. (Citado en páginas 1 y 2.)

[Möller*, et al.*, 2022] Möller, D. P. F., Vakilzadian, H., & Haas, R. E. (2022). From industry 4.0 towards industry 5.0. In *2022 IEEE International Conference on Electro Information Technology (eIT)* (pp. 61–68). (Citado en páginas 2, 4 y 5.)

[Muratet y Garbarini, 2020] Muratet, M. & Garbarini, D. (2020). Accessibility and serious games: What about Entity- Component-System software architecture? In *GALA 2020* Laval, France https://hal.science/hal-02987484. (Citado en página 21.)

[Qi*, et al.*, 2021] Qi, Q., *, et al.* (2021). Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*, *58*, 3–21, https://doi.org/10.1016/j.jmsy.2019.10.001. (Citado en páginas 6 y 11.)

[Rockwell-Automation, 2024] Rockwell-Automation (2024). Innovación, productividad y sostenibilidad para la fabricación del futuro. private communication. (Citado en páginas 3 y 5.)

[Serrano, 2019] Serrano, H. (2019). How does the game engine loop make a game possible? https://www.haroldserrano.com/blog/the-heart-of-a-game-engine-the-game-engine-loop. (Citado en páginas 22 y 23.)

[Stonis, 2022] Stonis, M. (2022). *Enterprise Application Patterns Using .NET MAUI*. Microsoft Developer Division, .NET, and Visual Studio product teams. (Citado en páginas 18 y 19.)

[Striffler y Voigt, 2023] Striffler, N. & Voigt, T. (2023). Concepts and trends of virtual commissioning – a comprehensive review. *Journal of Manufacturing Systems*, *71*, 664–680, https://doi.org/https://doi.org/10.1016/j.jmsy.2023.10.013 https://www.sciencedirect.com/science/article/pii/S0278612523002145. (Citado en páginas 13 y 14.)

[Tao y Qi, 2019] Tao, F. & Qi, Q. (2019). Make more digital twins. *Nature*, *573*, 490–491, https://doi.org/10.1038/d41586-019-02849-1. (Citado en página 9.)

[Ullah, 2022] Ullah, N. (2022). ¿qué es protobuf? https://appmaster.io/es/blog/que-es-protobuf. (Citado en páginas 20 y 21.)

[Walker, 2023] Walker, A. (2023). Singapore's digital twin - from science fiction to hi-tech reality https://infra.global/singapores-digital-twin-from-science-fiction-to-hi-tech-reality. (Citado en página 9.)

[Weisfeld, 2009] Weisfeld, M. (2009). *The Object-Oriented Thought Process*. Addison-Wesley. (Citado en páginas 15, 16 y 17.)

[Yuchen Jiang y Kaynak, 2021] Yuchen Jiang, Shen Yin, K. L. H. L. & Kaynak, O. (2021). Industrial applications of digital twins. *Royal Society*, https://doi.org/10.1098/rsta.2020.0360. (Citado en páginas 9 y 10.)