PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

SYSML COMO HERRAMIENTA PARA GARANTIZAR LA TRAZABILIDAD DE REQUERIMIENTOS EN EL DISEÑO MECATRÓNICO

Tesis para optar el Título de **Ingeniero Mecatrónico,** que presenta el bachiller

JESUS ENRIQUE VIDAL SANDOVAL

ASESOR: ELIZABETH R. VILLOTA CERNA

Lima, Enero del 2018

RESUMEN

La trazabilidad de requerimientos durante el proceso de diseño mecatrónico es de suma importancia ya que permite rastrear, ubicar y verificar eficazmente el cumplimiento de los requisitos establecidos por el cliente, o interesados. Si se introduce o aplica durante las primeras etapas de diseño es posible garantizarla, permitiendo resolver a priori problemas de diseño que de otra forma no se hacen evidentes hasta etapas posteriores. En ese sentido, la presente tesis muestra cómo los requisitos de trazabilidad para el diseño mecatrónico se pueden lograr mediante el empleo de una metodología de ingeniería de sistemas basada en modelos (MBSE) y la herramienta asociada, *Systems Modeling Language* (SysML). SysML es un lenguaje de multivista de propósito general para el modelado de sistemas capaz de vincular los requisitos a los elementos del sistema al capturar los requisitos textuales y colocarlos en los modelos de diseño. Además, SysML se puede acoplar a otras herramientas, incluidas las hojas de cálculo y el software de diseño y simulación, como *Matlab* o *Modelica*, lo que permite la verificación de los requisitos. Un actuador electromecánico (EMA), actuador de superficie del avión, se elige como caso de estudio de sistema mecatrónico. Al unirse a SysML y *Matlab/Simulink*, fue posible rastrear los requisitos para el diseño mecatrónico de EMA y, por lo tanto, verificar el cumplimiento de los principales requisitos asociados al diseño de control. Finalmente, esta tesis cierra proponiendo un trabajo futuro que contempla la necesidad de trabajar una lista más completa de requerimientos, los cuales llaman al uso de herramientas computacionales de ingeniería del tipo CAD como *Autodesk Inventor* o *CadSoft Eagle*, entre otros.

ÍNDICE DE CONTENIDO

Pág.

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Figuras de los Anexos

GLOSARIO Y NOMENCLATURA

Glosario	
Acción	Componente básico de las actividades y describe como debe ejecutarse
Actor	Es un rol (entidad, agente, elemento, etc.) que interactúa con el sistema. Un actor es externo al sistema.
Asociación	Describe una relación estructural entre dos 2 clases.
Asociación compuesta	Relación entre dos bloques del tipo todo-parte. Se simboliza con un rombo negro
Asociación de referencia	Relación general de un bloque. Se puede simbolizar con un rombo blanco o una línea
Atributo	Define una propiedad estructural de una clase. Consiste de visibilidad, nombre, tipo y multiplicidad
Bloque o *Block*	Es una unidad modular que describe la estructura de un sistema o elemento
Bloque de restricción	Permite la construcción de modelos paramétricos. Posee un conjunto de parámetros y una expresión que los relaciona.
Contenedor	Almacena uno o más elementos modelo.
Diagrama de actividades o **act**	Permite modelar un comportamiento en particular mediante una secuencia de acciones
Diagrama de definición de bloques o **bdd**	Es un diagrama que muestra la definición de un bloque y sus componentes
Diagrama de casos de uso o **uc**	Representa la funcionalidad en términos de como el sistema es usado por actores para cumplir una serie de metas
Diagrama de definición de bloques o **ibd**	Describe la estructura interna de un bloque en términos de cómo sus partes están interconectadas

Diagrama de máquinas de estado o **stm**	Representa el comportamiento de un bloque que transita a través de diferentes estados
Diagrama de paquetes o **pkg**	Es un contenedor que puede tener una estructura jerárquica para representar la organización de un modelo.
Diagrama paramétrico o **par**	Permite crear sistemas de ecuaciones que limite las propiedades de los bloques
Diagrama de requerimiento o **req**	Representa los requerimientos y sus relaciones con otros requerimientos, elementos de diseño y casos de uso que ayudan a la trazabilidad
Diagrama de secuencia o **sd**	Representa el comportamiento en términos de una secuencia de mensajes intercambiados entre el sistema o sus partes
Elemento modelo	Es cualquier elemento o mecanismo dentro de *SysML* que puede modelarse o representarse.
EMA	Actuador Electro-mecánico
Flujo de Objetos	Describe como los ítems de entrada y salida fluyen entre las acciones
FWB	*Fly-by-wire* o conexión por cables
Herramienta para modelar	Es una aplicación de software usada para crear y manejar modelos de *SysML*
Id	Identificación
Idle	Estado en el cual un procesador/microcontrolador no realiza ninguna acción porque no es usado por ningún otro recurso/tarea/computador
Instancia	Representa un elemento o característica para una situación en específico
MagicDraw	*Software* que permite modelar el sistema en *SysML*
Mensaje	Representa tanto la invocación de un servicio o un envío de una señal en un componente del sistema

Modelo	Un modelo describe un sistema para un propósito o dominio especifico.
Multiplicidad	Es un intervalo de *integers* positivos que describen cuántos objetos un atributo puede aceptar.
Namespace	Es aquel que contiene todos los elementos que son identificados únicamente por sus nombres (ejemplos de *namespace* son un paquete o un *block*)
Operando de Interacción	Pueden ser **Seq**, **Par**, **Alt/Else**, **Opt**, **Loop**, **Strict**, **Break**, **Critical**, **Neg**, **Consider** o **Ignore.** Ver Sección 2.5
Paquete o *Package*	Agrupa elementos modelo y forma un *namespace*
ParaMagic	*Plug in* que se acopla a *MagicDraw*. Permite vincular *Matlab/Simulink* y *Excel* a *MagicDraw*
Pin	Es un link entre parámetros de una acción y objeto que fluye. Hay 2 tipos *pin* de entrada o de salida.
Puerto	Describe un punto de interacción que es usado por una clase o bloque del sistema.
PWB	*Power-by-wire*
Relación de dependencia	Puede ser uso, refinamiento, realización, trazo o asignación
Requerimiento	Describe propiedades o comportamientos del sistema o de partes del sistema o de agentes externos al sistema que deben cumplirse para su desarrollo y desempeño
SysML	*System Modeling Language*. Es un *plug in* que debe acoplarse a *Magic Draw* para permitir modelar al sistema desde varios dominios.
Trazabilidad	Es la relación entre 2 o más elementos que permiten ubicar, rastrear y modificar eficaz y eficientemente algo.
Tipo de cantidad	Identifica un tipo de cantidad física como p. ej. altura cuyo valor puede estar expresado en metros (unidad).

Unidad	Relacionado siempre con un tipo de cantidad (h, m). Por ejemplo: metros, kilogramos, etc.

Nomenclatura		
Descripción	Símbolo	Unidad
Factor de amortiguamiento	cb	lb s/in
Constante resorte de la transmisión mecánica	kb	lb/in
Velocidad máxima del motor	w_{max}	rpm
Desplazamiento máximo de la transmisión mecánica	x_{lim}	in
Posición deseada del alerón	xd ó pd	in
Fricción viscosa de la carga	B_l	lb s/in
Ganancia efectiva después de compensación	Gb_{comp}	
Ganancia por zona muerta	Gb_1	
Ganancia de control proporcional	Gc	
Función de transferencia del lazo interno	G_2	
Ganancia de lazo interno	K_{inner}	
Constante de resorte de la carga	K_l	lb/in
Ganancia por realimentación de velocidad del motor	K_v	
Peso de la carga	M	lbm
Relación de Transmisión	N	rad/in
Voltaje de saturación debido al driver	V_{sat}	V
Caracterización de la zona muerta, ancho	α	in
Caracterización de la zona muerta, pendiente	m	in

INTRODUCCIÓN

Hoy en día nos encontramos en la cúspide de una Cuarta Revolución Industrial conducida por el uso de dispositivos inteligentes en la industria manufacturera. Esto se traduce en una exigencia de productos que deben realizar actividades complejas, lo cual implica una mayor capacidad en términos de datos, información y materia prima a procesar. El resultado es un elevado número de requerimientos que garanticen funcionalidad, interoperabilidad, rendimiento, fiabilidad y la consideración de lograr los objetivos a costa del tamaño más eficiente. La interconectividad entre los subsistemas mecánico, electrónico, de control y software también añade requisitos debido a que se espera que estos subsistemas trabajen en armonía. Lo antes descrito, hace énfasis en que actualmente los sistemas poseen una lista con mayor número de requerimientos, de ahí que la trazabilidad de estos sea de gran interés. Dichos requerimientos deben de tener un nexo que pueda ser trazable a lo largo de todas las etapas del diseño; en caso no se cumplan, debe informarse y actualizarse a conveniencia en el menor tiempo posible. A continuación, se presenta la situación problemática que motiva el desarrollo de este trabajo; para poder establecer luego los objetivos, justificación y alcance de la tesis. Finalmente, se presenta un breve resumen de los elementos desarrollados en esta tesis.

Situación problemática

El escenario de mercado está caracterizado fuertemente por las expectativas de los clientes, las cuales generan una competencia entre las compañías manufactureras que producen productos que intentan destacar entre ellos en cuanto a su durabilidad y rendimiento [1]. El tiempo y el costo son significativos para lograr esta tarea, se necesitan tiempos cortos y costos bajos para el proceso de diseño de un sistema [2]. El reto para los ingenieros es diseñar máquinas que sean capaces de funcionar y realizar los procesos en una manera determinística, en donde el diseño y la tecnología están

involucrados [3]. El reto eleva la expectativa de conocimientos, motivando el campo de la ingeniería mecatrónica. El término mecatrónica hace referencia a la cooperación simbiótica de la mecánica, la electrónica, el control y la ingeniería de software para mejorar el comportamiento de un sistema técnico [4], esto se muestra en la Figura 1.1. Así, un sistema mecatrónico está especificado en términos de, por ejemplo, software, elementos mecánicos, procesos termodinámicos, circuitos eléctricos y electrónicos, entre otros. Estas especificaciones resultan en un mayor número de requisitos del sistema y se formula con ellos la lista de requerimientos. Dicha lista debe poder rastrearse, es decir se debe conocer, su ubicación y trayectoria, a esto se le conoce con el nombre de trazabilidad[1]. La trazabilidad concierne a todos los requisitos con los cuales se define el sistema [5].

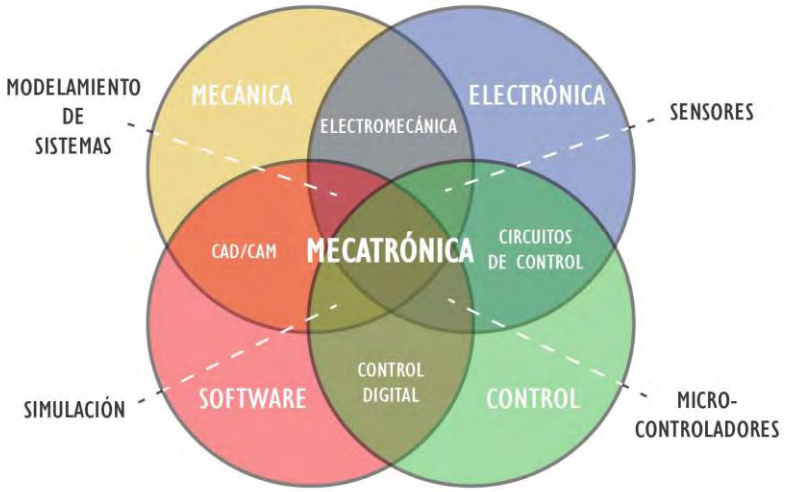


Figura 1.1 Mecatrónica; Fuente:
http://todanovidade.com.br/ciencia/engenharia-mecatronica/

Existen algunas normas, estándares y guías que se han desarrollado para poder realizar el diseño de un sistema mecatrónico. Por ejemplo, algunas de estas han sido desarrolladas por la Asociación de Ingenieros Alemanes (*VDI* por sus siglas en alemán) como la *VDI 2206* o la *VDI 2221*. En general puede describirse el diseño de un producto como un conjunto de fases (ver Figura 1.2). que comprenden *Diseño y Especificación (SD)*, *Diseño Conceptual (CD)*, *Diseño de Producto (PD)*, *Soporte y Producción de Producto (PS)* [6]. Sin embargo, en las guías de diseño VDI no se

[1] La Organización Internacional para la Estandarización (ISO 9001:2008) [26] define la trazabilidad como "la propiedad del resultado de una medida o del valor de un estándar, que éste pueda relacionar con todas las demandas del cliente, o estándares (normas) que el gobierno regula a través de una cadena continua de comparaciones,".

especifica una técnica que garantice la trazabilidad; y es que, seguir una secuencia de estas etapas no implica una garantía de la trazabilidad de los requerimientos.

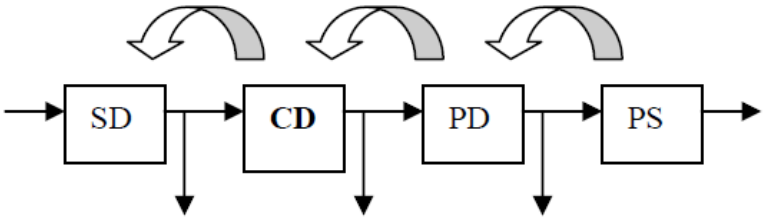


Figura 1.2 Fases Generales del Diseño de un Producto;
Fuente: [6]

Si el cliente cambia de opinión en los requisitos de la máquina, por ejemplo, por factores de costo o rentabilidad, y no existe una trazabilidad de los requerimientos, las modificaciones implicarían un análisis adicional del sistema que se traduce en tiempo de espera, esfuerzo y evidentemente un costo significativo. En el peor de lo casos, un producto puede retornar debido a fallas, lo que significa que uno de los requerimientos no se está cumpliendo; y si los ingenieros no pueden localizar rápidamente a qué sistema está vinculado este requisito y por tanto resolver la falla, la imagen de la empresa puede verse afectada.

El contexto anteriormente descrito implica que una ingeniería de requerimientos que garantice la trazabilidad debe estar presente no solamente en la perspectiva técnica, sino que también debe considerarse en la situación contractual y durante todo el desarrollo del diseño puesto que la especificación de requerimientos es materia de interés entre el cliente y el contratista.

El lenguaje unificado de modelado – *UML* por sus siglas en inglés – presenta características para especificar o describir métodos, procesos o modelos. *UML* es un lenguaje de visualización gráfica, capaz de especificar, construir y documentar un sistema, lo cual lo vuelve un lenguaje bastante útil para la ingeniería de requerimientos.

El lenguaje de modelado de sistemas – *SysML* por sus siglas en inglés – es un lenguaje ampliado de *UML*; puesto que *UML* es usado para representar software, *SysM*L se crea para poder representar no solo sistemas de software, sino otros dominios útiles para otros campos de la ingeniería.

Sin embargo, lidiar con diferentes modelos y documentos para validar que los requerimientos se están cumpliendo es un reto común a lo largo del diseño. Un mismo sistema puede modelarse con diferentes herramientas (*Inventor Autodesk*, *Simulink*,

Eagle, etc.) en donde cada modelo representa una perspectiva importante y diferente del sistema que al mismo tiempo presenta una interacción entre sus elementos, los cuales en conjunto buscan cumplir los objetivos a un determinado comportamiento.

La trazabilidad de requerimientos con un enfoque de Ingeniería de Sistemas Basada en Modelos (MBSE por sus siglas en inglés), y con énfasis en las dependencias y asociaciones que existen entre los diferentes modelos, permiten rastrear y encontrar rápidamente un requerimiento, ayudando a los ingenieros a mitigar los efectos negativos que pueda reflejar el cambio de los requerimientos del diseño o el modelo.

En ese sentido, es importante mencionar el objetivo general de esta tesis, así como también los objetivos específicos y sus alcances.

Objetivo general

Mostrar el uso de *SysML* como una herramienta que permite garantizar la trazabilidad de requerimientos en el diseño de sistemas mecatrónicos.

Objetivos específicos

a. Contextualizar brevemente el estado del arte del diseño mecatrónico y hacer énfasis en la trazabilidad de requerimientos.
b. Presentar *SysML* como un lenguaje que permite modelar a un sistema mecatrónico en diferentes dominios o perspectivas.
c. Presentar *SysML* como un lenguaje que permite la trazabilidad de requerimientos a lo largo del diseño meactrónico.
d. Mostrar mediante un caso de estudio la trazabalidad de requerimientos para un diseño mecatrónico: actuador electro-mecánico.

Alcances

La presente tesis analizará un actuador electro-mecánico como sistema mecatrónico desde un enfoque de ingeniería de sistemas basada en modelos. Se utiliza como herramienta y software principal *MagicDraw* de la empresa ***No Magic*** [7] junto con el *plug in* que permite usar *SysML*. En adición, se añade al software el *plug in ParaMagic* el cual permite vincular *MagicDraw* con *Matlab*/*Simulink* y *Excel*. No concierne a este trabajo explicar a detalle cómo se logra el intercambio de información entre las diferentes herramientas de ingeniería y el software principal. Asimismo, los modelos que se realizan están simplificados para no perder de vista el objetivo principal de la tesis, trazabilidad de requerimientos.

Resumen de la tesis

La presente tesis comienza describiendo la problemática y consiguientes objetivos del trabajo, además de los alcances, los cuales se detallan en esta introducción. El capítulo uno comprende la pertinente descripción del estado del arte, donde muestra la situación actual de la trazabilidad de requerimientos usando como base estándares y guías de diseño en el ámbito de la ingeniería afines a la mecatrónica. El capítulo dos presenta de manera breve y sucinta el software *Magic Draw* acoplado con *SysML* (*System Modeling Language*) como una herramienta que permite modelar sistemas mecatrónicos complejos desde diferentes perspectivas y con un enfoque basado en modelos (modelo organizado por paquetes, modelo estructurado por bloques, modelo paramétrico, modelo de comportamiento usando diagrama de actividades, modelo de interacciones, modelo de máquinas de estado y modelo de funcionalidad). Seguidamente, el capítulo tres discute la trazabilidad de requerimientos de diseño usando *SysML* haciendo énfasis en que los elementos en los diferentes modelos antes descritos pueden tener la característica de trazables. Un caso de estudio valida la versatilidad del software *MagicDraw* acoplándolo con *Simulink* y *Excel* en el capítulo cuatro, resaltando que ante el uso de otras herramientas la trazabilidad no se ve afectada sino al contrario se hace evidente. Finalmente, se discuten los resultados, se presenta las conclusiones y se enuncian las respectivas recomendaciones.

CAPÍTULO 1
ESTADO DEL ARTE

1.1 Requerimientos en el diseño mecatrónico y afines

Los departamentos de diseño enfrentan problemas con respecto a la identificación de los requerimientos de diseño [8]. Una lista de requerimientos es necesaria para comprender y aclarar la solicitud del cliente. Sin embargo, realizarla suele ser una tarea dificultosa ya que se deben aterrizar las ideas del cliente o de los interesados en el proyecto. Por suerte, expertos en el tema como Pahl y Beitz [8] ya han determinado las preguntas básicas que se deben realizar para enunciar dicha lista. Algunas de las preguntas se listan a continuación:

- ¿Cuál es verdadero problema?
- ¿Cuáles son los objetivos que la solución espera satisfacer?
- ¿Cuáles son las propiedades que la solución debe poseer?
- ¿Cuáles propiedades no debe de tener la solución?
- ¿Cuáles son deseos implícitos y qué se espera del producto?
- ¿Qué caminos están abiertos para el desarrollo del producto?

Estas preguntas evidencian una delimitación de los requerimientos. Sin embargo, es importante seguir pasos para desarrollar la lista de requerimientos, los cuales se muestran en la Figura 1.1a. El procedimiento comprende dos etapas principales. En la primera, los requerimientos evidentes son definidos y guardados. En la segunda etapa, estos requerimientos son detallados y extendidos usando métodos especiales [8].

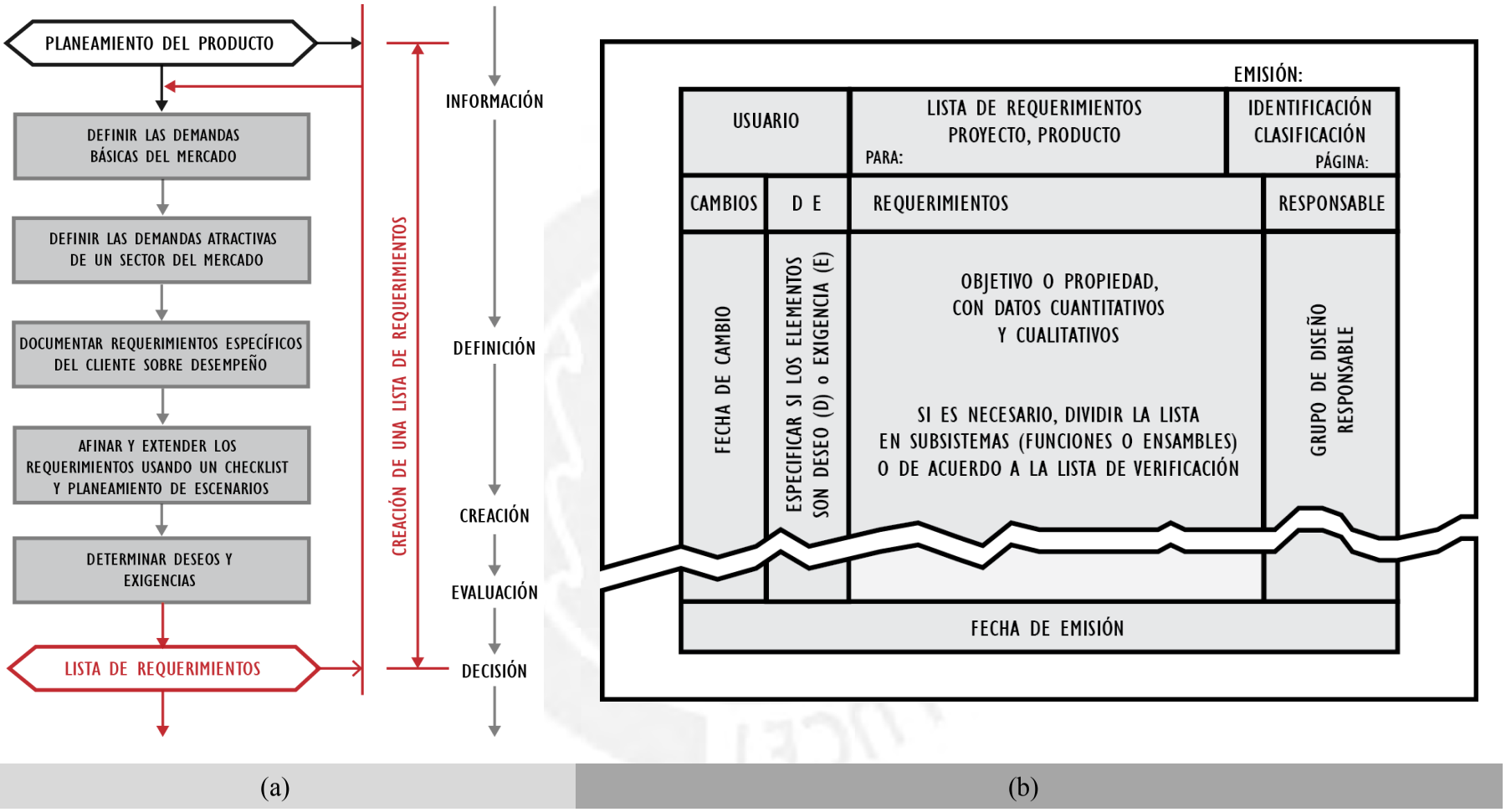


Figura 1.1 (a) Pasos para hacer una lista de requerimientos, (b) Lista de Requerimientos;
Fuente [8]

Como se muestra en la Figura 1.1b se debe hacer una lista de requerimientos, los cuales incluyen exigencias (E) y deseos (D), en donde cumplir las exigencias es un requisito del cliente y un compromiso de la empresa que brinda el servicio [8]. La información que se tiene de esta lista es en su mayoría cuantitativa o referente a lo cualitativo, en donde:

- Cuantitativa: Es toda la información en números y magnitudes, como número de ítems requeridos, peso máximo, salida de voltaje, tasa de flujo, etc.
- Referente a la cualitativo: Es toda la información que especifica variaciones permisibles o requerimientos especiales, como: a prueba de agua, a prueba de corrosión, a prueba de golpes, etc.

Es importante mencionar que, en principio, la lista de requerimientos debe ser coherente y bien definida. Al comienzo la lista es provisional, puesto que en el trayecto del diseño esta crece y se modifica; Pahl y Weitz advierten que todo intento de formular todos los posibles requerimientos, en primera instancia, fallará y causará retrasos considerables al proyecto [8].

La lista de requerimientos depende del estado del diseño del producto y la fase en que se encuentra el proceso del diseño. Incluso, la lista cambia después de ser entregada al usuario final, este es el caso de una versión 2.0 del producto en donde se encuentran características adicionales o mejoras.

Por otro lado, es común tener una lista de requerimientos parciales (ver Figura 1.2.), que se derivan de otras áreas de la compañía o de los integrantes del proyecto y que se suman a la lista principal de requerimientos [8].

Una última característica de la lista de requerimientos, es que no refleja solamente la postura inicial del cliente, ya que, al ser revisada constantemente, tiene otras funciones como:

- Actualizar la documentación que se realiza.
- Proveer información si se examina en colaboración con el cliente o se examina un nuevo producto.
- Dar conocimiento básico para la gestión de sistemas.

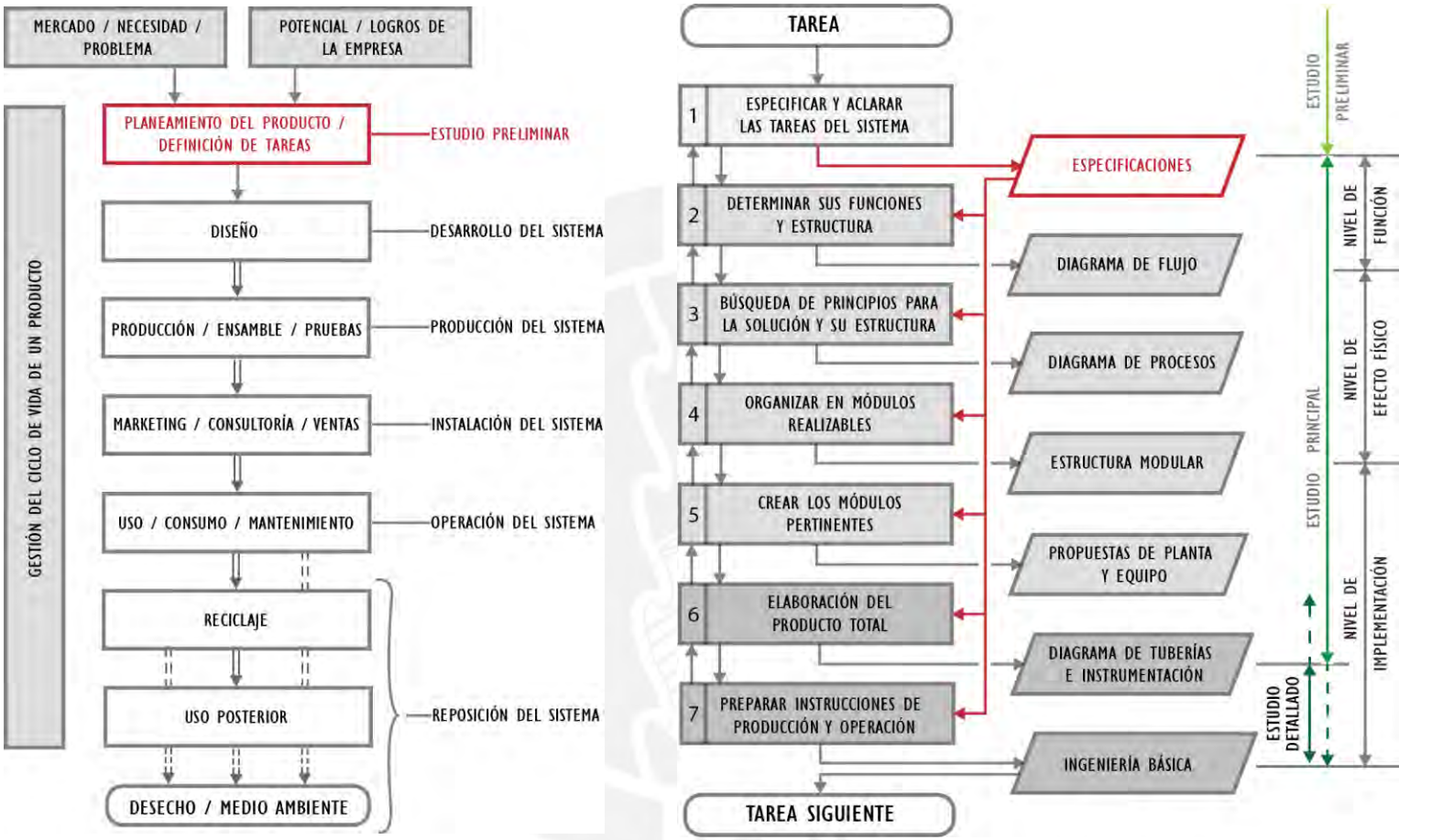
Figura 1.2 Requerimientos Parciales;
Fuente[8]

Entendiendo la lista de requerimientos como paso principal en el diseño, a continuación, se presentan las guías y o estándares internacionales para los sistemas mecatrónicos y afines con el objetivo de identificar el uso del término trazabilidad en estos. Brevemente se describe en las secciones 1.1.1 - 1.1.3 los enfoques de las casas alemanas y en las secciones 1.2.1 - 1.2.3 los enfoques para el caso de la ingeniería de sistemas.

1.1.1 VDI 2221

La VDI 2221 presenta un enfoque sistémico en el diseño y desarrollo de productos y sistemas técnicos [9]. Fue propuesta por la Asociación de Ingenieros Alemanes – VDI - por sus siglas en alemán.

Esta guía contiene 6 capítulos: objetivos, fundamentos del enfoque sistémico, plan de enfoque, ejemplos del enfoque sistémico, métodos y terminología. La descripción del enfoque sistémico, se ayuda de la ingeniería de sistemas para definir los conceptos básicos, en esta parte se mencionan los requerimientos haciendo referencia a la definición del problema e inclusive se muestra el ciclo de vida del producto y el enfoque general de diseño (ver Figura 1.3 a y b). En ambas figuras se mencionan los requerimientos; sin embargo, en el ciclo de vida, los requerimientos no se relacionan con sus siguientes etapas. El aporte más significativo es el enfoque general de diseño, en donde se observa que los requerimientos deben compararse y verificarse en cada etapa del diseño. Lamentablemente, en esta guía no aparece la palabra trazabilidad, ni se explica como los requerimientos pueden mantenerse firmes a lo largo del diseño.



(a) (b)

Figura 1.3 (a) Ciclo de vida del producto y (b) Enfoque General de Diseño de Ingeniería según VDI 2221;
Fuente [9]

Al ser secuencial no se puede apreciar si el cambio en una etapa tardía afecta una temprana.

1.1.2 VDI 2422

VDI 2422 presenta un desarrollo sistémico para dispositivos controlados por microcontroladores. Esta guía no posee mucho detalle dado que se enfoca en el desarrollo de un producto en general [10]. Por otro lado, hay un desarrollo en paralelo o concurrencia como se muestra en la Figura 1.4, siendo el último paso la integración de estos. No hay un concepto general de trazabilidad.

1.1.3 VDI 2206

VDI 2206 es una metodología de diseño para sistemas mecatrónicos. Esta metodología fue realizada para complementar la VDI 2221 y la VDI 2422. Consiste en una serie de pasos secuenciales que transmiten disciplina al diseñar en el campo de la ingeniería, lo que resulta en un diseño con mayor sinergia [10].

Figura 1.4 Modelo General de Desarrollo según VDI 2422;
Fuente [10]

Los conceptos clave que se mencionan en esta VDI son:

- El problema general al diseñar o resolver a nivel micro
- La forma de V en el diseño en un modelo macro
- Procesos predefinidos para repetir pasos durante el diseño de sistemas mecatrónicos

Se observa en la Figura 1.5 el modelo en forma de V. Las partes que la componen son requerimientos, el punto de partida que describe el producto en forma de requisitos; diseño del sistema, que describe el concepto de solución en dominios cruzados, para esto se descompone la función del sistema en sub-funciones que luego pueden probarse para verificar el producto; diseño de dominio específico, se desarrolla el concepto de solución en paralelo, la ingeniería mecánica, eléctrica y tecnológica; integración del sistema, es el resultado de unir los dominios desarrollados anteriormente; comprobación de propiedades, es el paso que verifica el concepto de solución con los requerimientos; las últimas 2 partes son el de modelado y análisis del modelo que finalmente lleva al producto final. Está guía tiene un enfoque en el dominio de la mecánica y está orientada a un punto de vista con referente a la forma [10]. Si bien

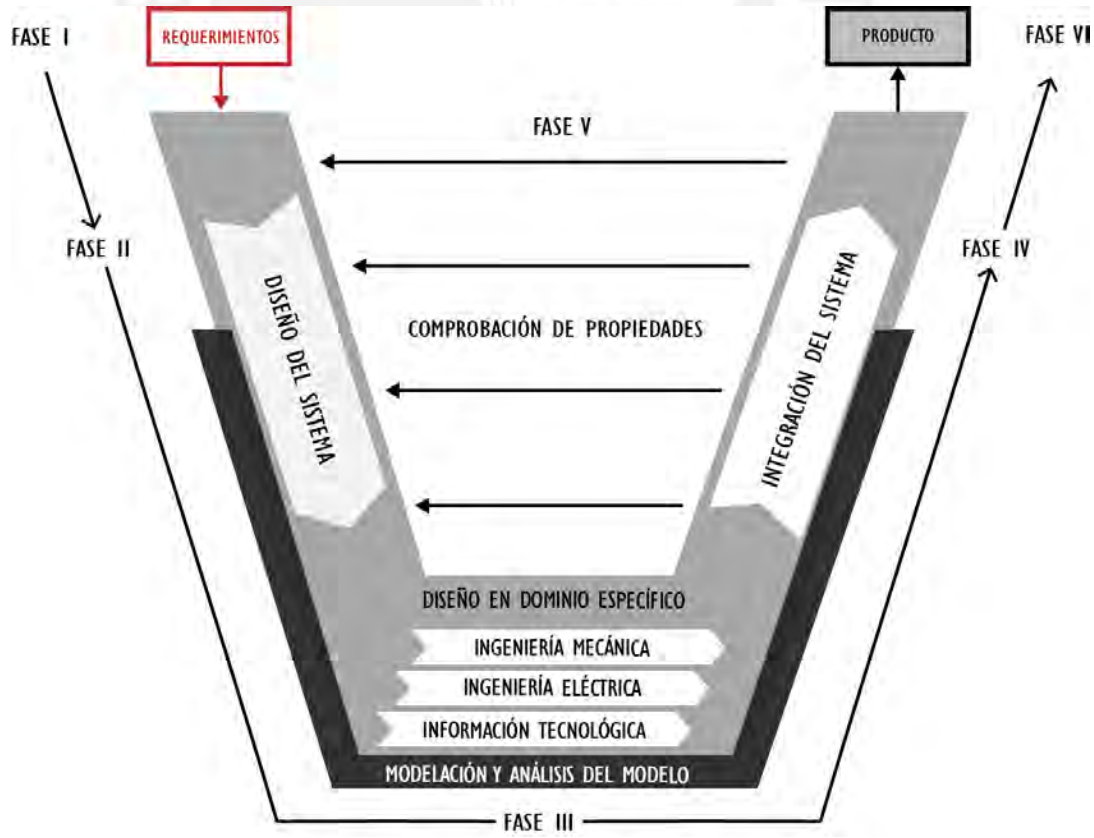


Figura 1.5 Modelo en Forma de V según VDI2206;
Fuente [10]

esta metodología es una de las más completas, no otorga una técnica, ni una forma rápida de vincular los requerimientos en las etapas del diseño mas sí enfatiza en qué partes deben vincularse.

1.2 Ingeniería de Sistemas / Ingeniería de Requerimientos

A continuación se mencionan algunos estándares que se encuentran dentro de la ingeniería de sistemas y la ingeniería de requerimientos, definidos por el Instituto de Ingenieros Eléctricos y Electrónicos – IEEE de sus siglas en inglés, la Alianza de Industrias Electrónicas – EIA de sus siglas en inglés, la Organización Internacional de Estandarización – ISO de sus siglas en inglés, y la Comisión Electrotécnica Internacional – IEC de sus siglas en inglés.

1.2.1 Estándar IEEE 1220

Es un estándar para la gerencia y desarrollo de sistemas en el proceso de ingeniería. El mismo define las tareas interdisciplinarias que son requeridas en el ciclo de vida del producto para transformar las necesidades y limitaciones en una solución [11]. La cláusula 4 hace mención especial a los requerimientos y la Figura 1.6 hace referencia al proceso en la ingeniería de sistemas descomponiéndolo en 8 subprocesos con sus asociadas tareas, flujo general del proceso y actividades. Hay que resaltar, que el estándar hace mención a la trazabilidad dirigiéndose a la verificación del diseño. Sin embargo, es una trazabilidad con respecto a los componentes iniciales y no hacia los requerimientos del sistema.

1.2.2 Estándar EIA 632

Es un estándar que define un enfoque sistémico de ingeniería o reingeniería como se muestra en la Figura 1.7, además posee 3 premisas fundamentales [12]:

- Un sistema es uno o más productos finales que trabajan en conjunto con otros productos a lo largo de su ciclo de vida para satisfacer las necesidades y expectativas del cliente o interesados y permiten alcanzar un producto final;
- Los productos son una composición de elementos en estructura jerárquica que se integran para satisfacer las necesidades y expectativas del cliente o interesados;
- La Ingeniería de Sistemas y sus productos relacionados alcanzan sus objetivos al aplicar un conjunto de procesos a cada elemento del sistema jerárquico en

un ambiente y con un equipo multidisciplinario que tiene como requisito los conocimientos y habilidades.

EIA 632 contiene 33 requerimientos guías y se define la palabra trazabilidad de requerimientos en la página 19 como el seguimiento de los requisitos que abarcan la identificación de la adquisición de los requerimientos de los clientes o interesados hasta los requerimientos del sistema técnico, representación de la solución lógica, representación de la solución física, requerimientos técnicos derivados, y requerimientos específicos [12].

1.2.3 Estándar ISO/IEC 15288

El estándar ISO/IEC 15288 establece un marco de trabajo para el proceso de ciclo de vida [13]. Este estándar desglosa el sistema de interés en partes que se pueden implementar en forma discreta, que permiten contener uno o más subsistemas [11]. La Figura 1.8 muestra el proceso técnico según ISO/IEC 15288 para un sistema de ingeniería. Es importante mencionar que posee un nivel de abstracción mayor comparado con los estándares EIA632 e IEEE 1220 [14]. Además, puede trabajar en colaboración con IEEE 1220, y contiene una buena referencia para definiciones de procesos suplementarios [11]. La palabra trazabilidad se menciona en este estándar y también hace referencia al modelo en V de la VDI 2206.

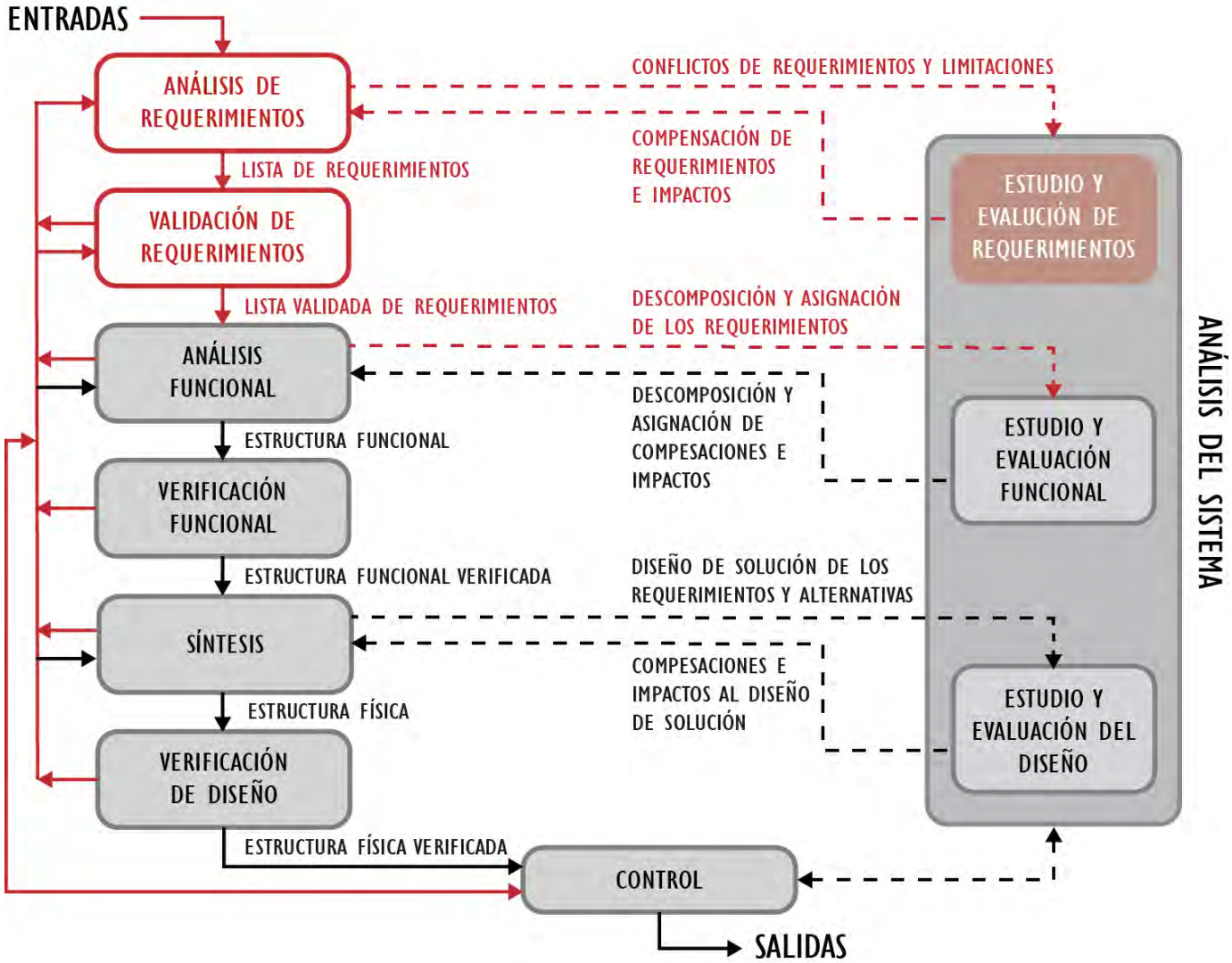


Figura 1.6 Proceso de Ingeniería de sistemas subdividido en 8 subprocesos según IEEE1220; Fuente [11]

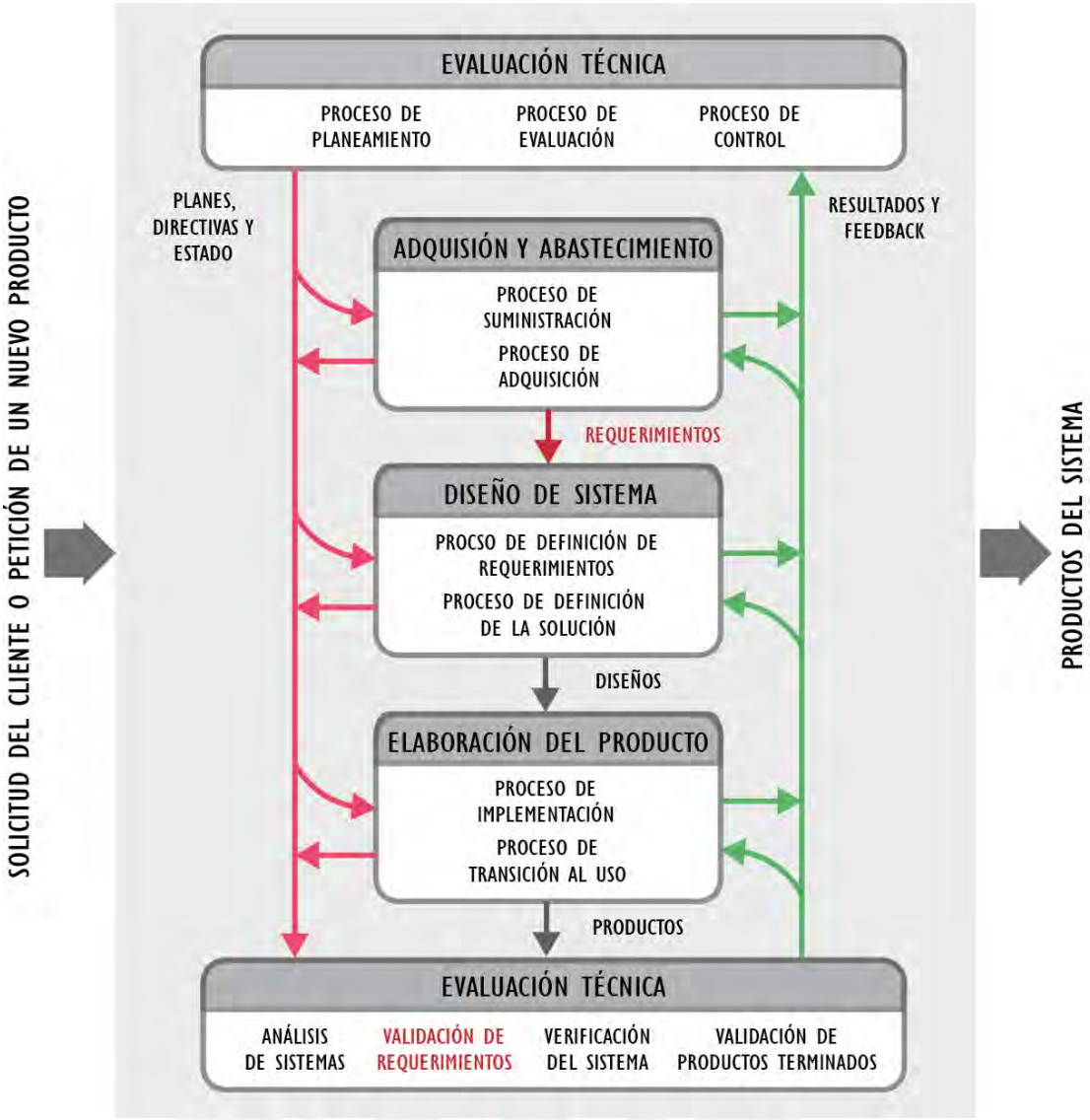


Figura 1.7 Relación de Procesos en la Ingeniería de Sistemas según EIA 632; Fuente [12]

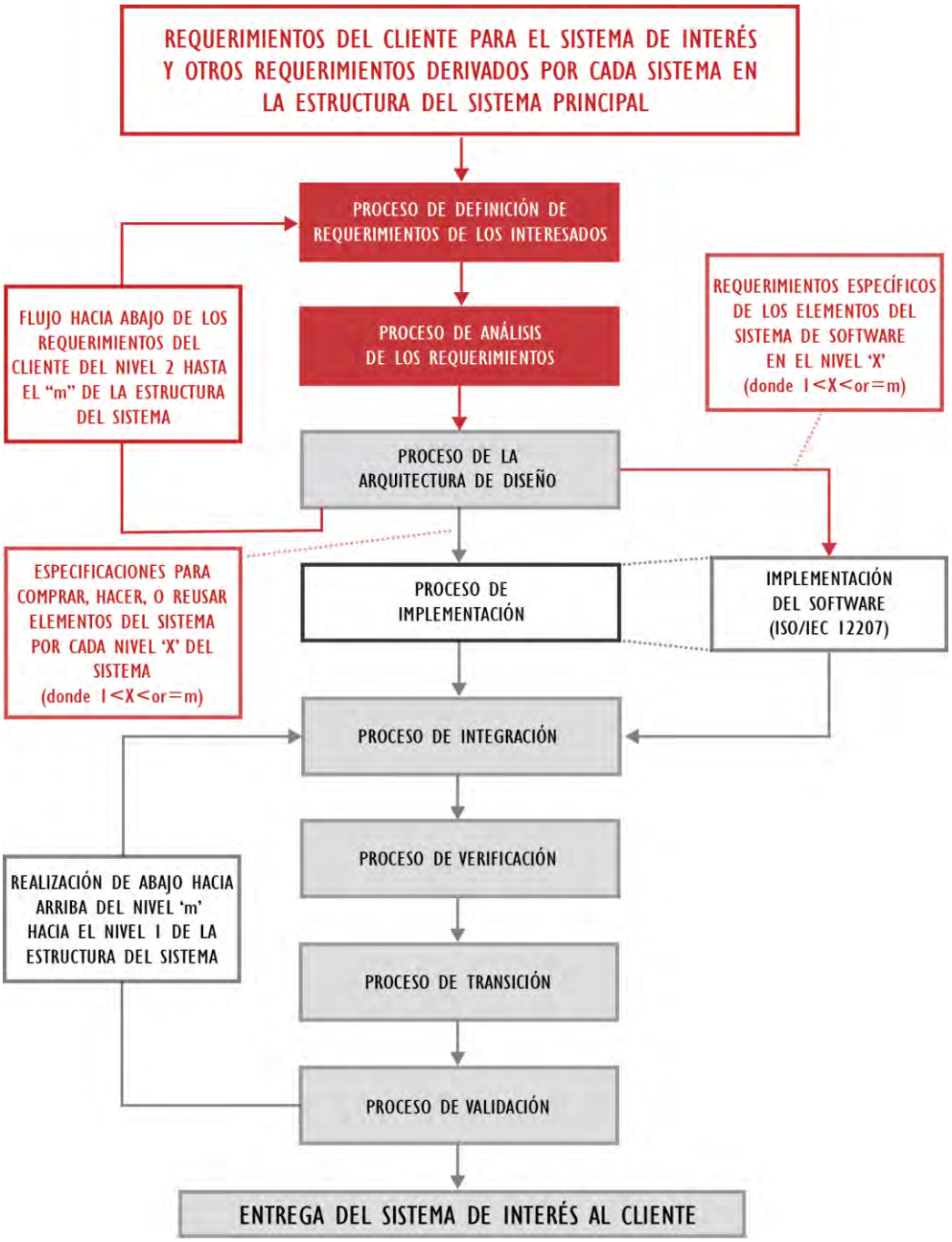


Figura 1.8 Proceso Técnico según ISO/IEC 15288 a un sistema de ingeniería; Fuente [13]

1.3 Integración de la trazabilidad de requerimientos al proceso de diseño mecatrónico

1.3.1 Diseño mecatrónico basada en modelos

Según investigaciones en los últimos años [3, 4, 7, 10-19], una alternativa para lograr la trazabilidad de requerimientos de diseños mecatrónicos complejos es utilizar un enfoque basado en modelos, a diferencia de un diseño secuencial como las explicadas

en las secciones 1.1 y 1.2. Esto resulta en una nueva manera de pensar para los ingenieros de diseño. Según [17] una mecatrónica integrada usando la Ingeniería de Sistemas Basada en Modelos – MBSE de sus siglas en inglés- permite una serie de beneficios potenciales como son una terminología uniforme, una representación gráfica del producto con una arquitectura estructurada, una documentación uniforme y estándar del producto, una reducción significativa de confusión del comportamiento del sistema, identificación rápida de conexiones e interdependencias en soluciones similares, mejoramiento en desempeño y comunicación del equipo, e identificación y resolución de problemas de diseño antes de la especificación detallada. Por otro lado, el mismo autor resalta que los diseños de sistemas mecatrónicos deben realizarse usando un enfoque paralelo, como se muestra en la Figura 1.10.

Figura 1.9 Mecatrónica integrada a un sistema basado en modelos;
Fuente [17]

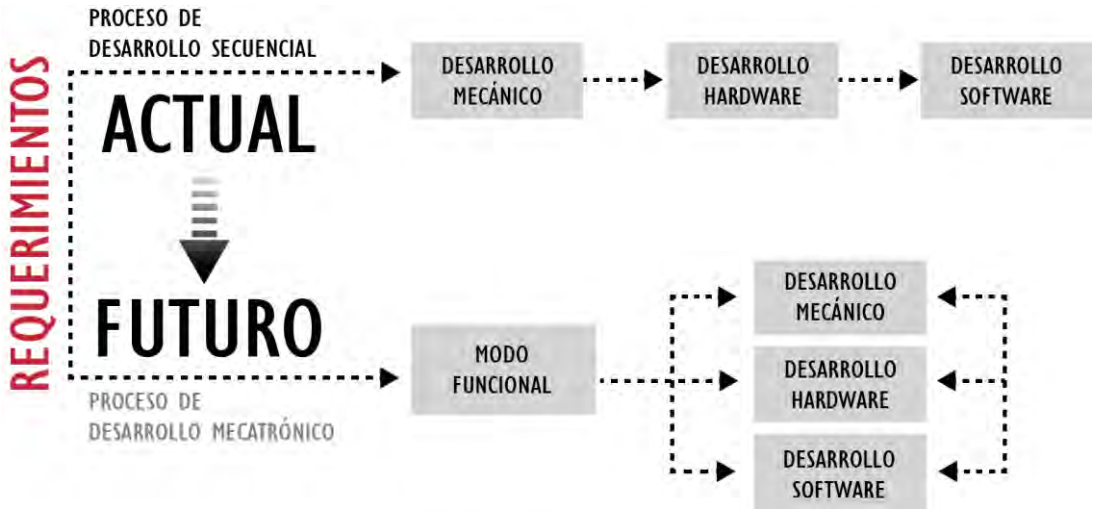


Figura 1.10 Enfoque Secuencial y Enfoque en Modelos;
Fuente [17]

La MBSE necesita un buen planeamiento en conjunto de una buena coordinación para definir el grupo de subsistemas que comprenderá el sistema principal. Naturalmente, definir qué es un modelo es necesario y en esencia, un modelo es la abstracción de un sistema real y solo representa una parte de las características del sistema. Esta abstracción permite construir modelos ligeros que son fáciles de entender y que consumen menos tiempo a la hora de hacer una simulación [18]. MBSE busca enfocar los modelos de un sistema utilizando cualidades de la ingeniería de sistemas, para brindar apoyo en el análisis, especificación, diseño y verificación del sistema que se desea desarrollar [19], ver Figura 1.11.

1.3.2 Trazabilidad de requerimientos basado en modelos

Tradicionalmente se utilizan documentos basados en ingeniería de sistemas, los cuales pueden tener el plan de manejo del sistema que contienen diagramas funcionales, diagramas de bloques, diagramas de flujo, bocetos, entre otros. A todos estos documentos, a uno individual, o a algunos agrupados, se les puede hacer un análisis, como por ejemplo de confiabilidad, de desempeño, de riesgo, de seguridad, de propiedades, etc. Estos documentos son rigurosos, y pueden alcanzar un buen resultado; sin embargo, una limitación es el acceso a la información que estos documentos comparten con los requerimientos de diseño, dificultando el análisis de ingeniería e información de testeo [19].

Un enfoque basado en documentos ocasiona un esfuerzo grande, dado que debe existir un manejo riguroso de estos; además de ser laborioso, consume mucho tiempo clasificar, actualizar y encontrar los documentos que se necesitan [18]. Incluso existe

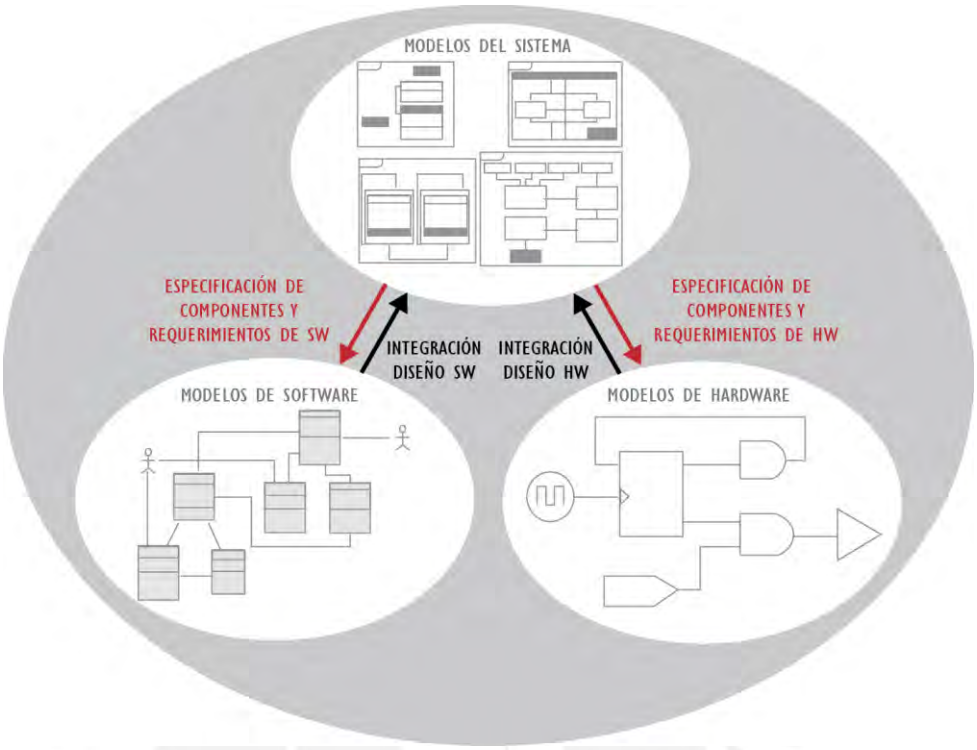


Figura 1.11 Ingeniería de Sistemas Basada en Modelos o MBSE;
Fuente [19]

un reto adicional, el cual es manejar las diferentes versiones del documento y asegurarse que se esté trabajando con la última versión [18].

Utilizar la trazabilidad en el diseño se convierte es una necesidad para reducir estos problemas. Lo que se necesita es vincular la etapa de diseño con los requerimientos sabiendo que través de diferentes perspectivas de ingeniería se pueden lograr diferentes modelos del sistema (p. ej. modelo mecánico o el eléctrico) y que las interacciones de estas perspectivas con sus requerimientos (la solicitud del cliente o las exigencias que las demandan), otorguen su estado (si se logró cumplir el requerimiento o no) y ubicación (a qué parte del sistema está vinculado), permitiendo obtener el adjetivo de trazables. En la Figura 1.12 se puede observar como los requerimientos están vinculados a los dominios o modelos del sistema, el cambio de un modelo o un requerimiento en el trayecto del diseño puede afectar a otros; sin embargo, una solución rápida puede obtenerse si los requerimientos poseen trazabilidad.

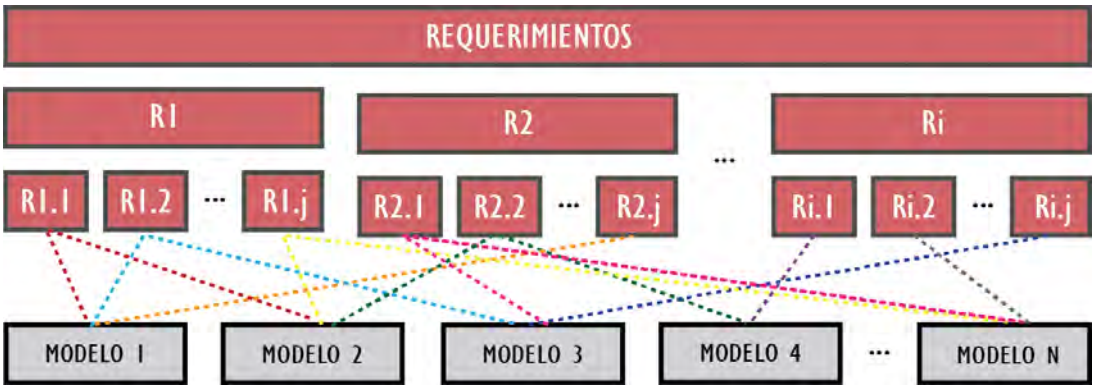


Figura 1.12 Trazabilidad en un Sistema de Ingeniería Basado en Modelos;
Fuente propia

1.3.3 Integración del diseño mecatrónico y trazabilidad basada en modelos

MBSE puede usar diferentes herramientas y lenguajes dependiendo de: los diferentes dominios que involucran al sistema, el detalle de nivel, los aspectos del sistema a ser modelados, etc. Se puede utilizar, *CATIA* o *ProEnginer*, para diseño asistido por computadora (*CAD*); *Modelica* para sistemas físicos complejos; *VHDL-AMS* para señales del sistema y circuitos integrados; *Matlab Simulink* para el análisis de la dinámica del sistema, etc. [18].

Las etapas tempranas del diseño exigen que los requerimientos del cliente y de los interesados sean definidos y progresivamente refinados. Además, los modelos funcionales abstractos del sistema, deben ser trazados por sus requerimientos respectivos, para vincular sus funciones, dependencias y comportamiento. *SysML*, el Lenguaje para Modelar Sistemas calza en las solicitudes demandadas para realizar los modelos del sistema.

SysML - System Modeling Language - [20] ha sido propuesta por el *Object Management Group* (*OMG*) [21] como un lenguaje que permite la comunicación entre diferentes disciplinas como también la construcción de modelos que muestren que se cumplan los requisitos del diseño, es decir modelos que permitan trazabilidad en el dominio de un diseño específico. Por otro lado, *SysML* es capaz de capturar los requerimientos textuales y situarlos en los diferentes modelos del diseño [3].

Un ejemplo a grandes rasgos de como *SysML* puede relacionarse con el sistema en general se muestra en la Figura 1.13, en donde existen varios modelos contenidos en paquetes [19]. El modelo de requerimientos contiene el requisito R1 que a su vez contiene los requisitos R1.1 y R1.2. El modelo de comportamiento contiene a la actividad A1 que realiza el sistema *System 1* y que satisface el requerimiento R1.2.

Esta vinculación permite que R1.2 sea trazable. Por otro lado, hay un modelo de estructura que contiene las partes que conforman el sistema, subdividiéndolo en los sistemas *System1* y *System2,* y granulándolo aún más en los componentes que *System1* posee. Se observa, también, las vinculaciones entre las actividades y los componentes. Finalmente, hay un modelo paramétrico, que en esencia es el modelo matemático de una parte del sistema, conteniendo propiedades que están vinculadas con la estructura del sistema.

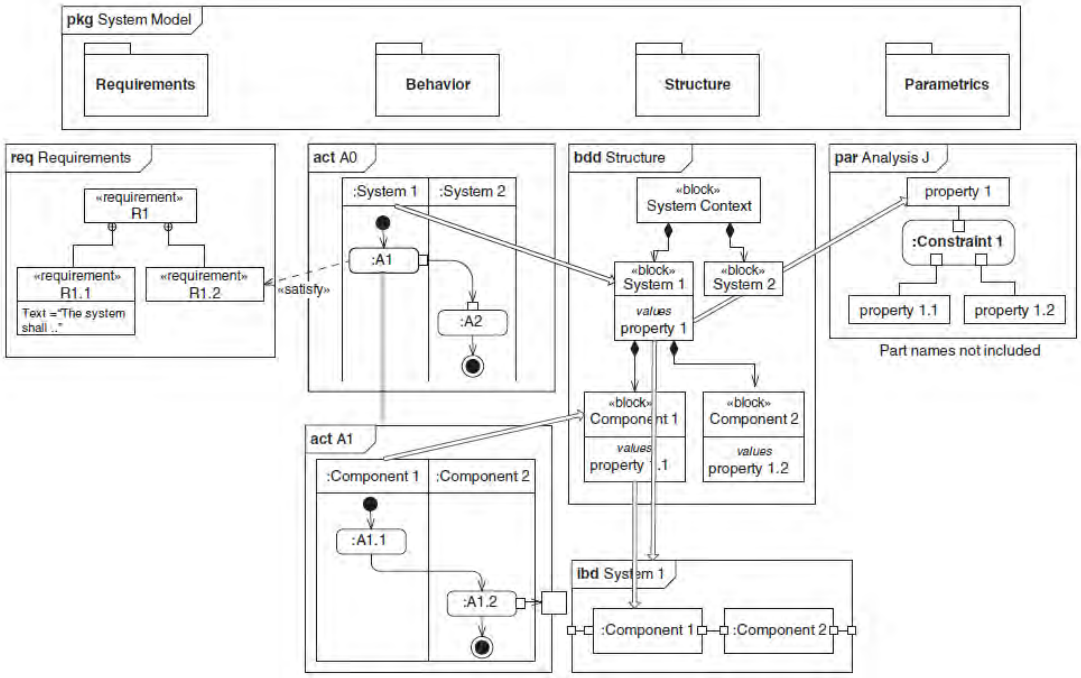


Figura 1.13 Trazabilidad en SysML;
Fuente [19]

CAPÍTULO 2
MODELOS EN SYSML

La siguiente información es una recopilación [2,19,20,23] de los diferentes modelos que *SysML* ofrece para poder estructurar y modelar un sistema de ingeniería complejo.

Se debe recordar que *SysML* es un software de modelado gráfico de propósito general que ayuda al análisis, especificación, verificación, y validación de sistemas complejos [19]. En general el diagrama *SysML* puede contener hasta 9 diagramas que permiten representar varios aspectos del sistema como: composición estructural, interconexión de estados, restricciones en el desempeño físico, requerimientos y su interacción y relación con otros requerimientos, elementos de diseño y casos de prueba. Dichos diagramas son:

i. Diagrama de Paquetes o *Package*: Representa la organización de un modelo en términos de paquetes que contienen elementos del modelo.
ii. Diagrama de Requerimientos o *Requirement*: Representa los requerimientos y sus relaciones con otros requerimientos, elementos de diseño y casos de prueba que ayuda la trazabilidad de los requerimientos.
iii. Diagrama de Actividades o *Activity*: Representa el comportamiento en términos del orden en que las acciones se ejecutan basadas en la disponibilidad de sus entradas, salidas, del control y como las acciones transforman las entradas en salidas.
iv. Diagrama de Secuencias o *Sequence*: Representa el comportamiento en términos de una secuencia de mensajes intercambiados entre el sistema o entre partes del sistema.
v. Diagrama de Máquina de estados o *State Machine*: Representa el comportamiento de una entidad en términos de sus transiciones entre cambios de estados provocados por eventos.

vi. Diagrama de Casos de Uso o *Use Case*: Representa la funcionalidad en términos de como el sistema es usado por agentes externos (actores) para cumplir una serie de metas.

vii. Diagrama de Definición de Bloque o *Block Definition*: Representa elementos estructurados llamados bloques o *blocks* en su composición y clasificación.

viii. Diagrama Interno de Bloque o *Internal Block*: Representa las interconexiones e interfaces entre las partes de un bloque.

ix. Diagrama Paramétrico o *Parametric*: Representa las limitaciones en valores de sus propiedades, como ecuaciones por ejemplo $F = m \cdot a$.

Los nueve diagramas se muestran en la Figura 2.1

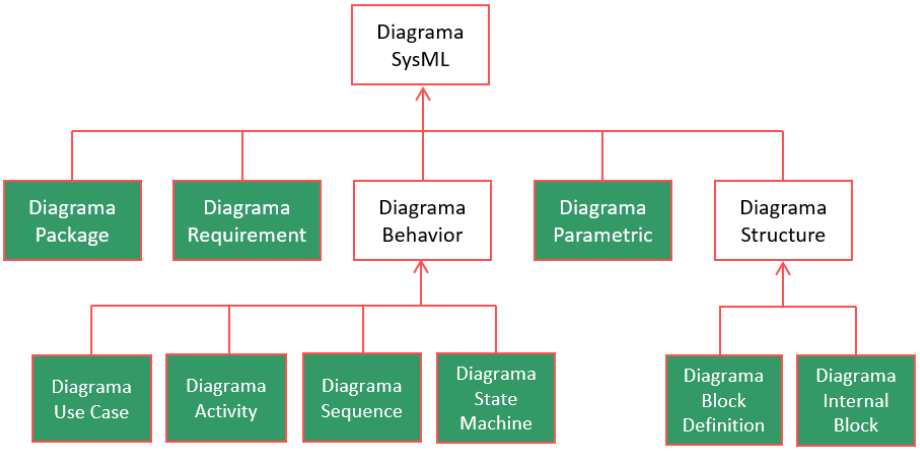


Figura 2.1 Los 9 Diagramas en SysML; Fuente [19]

2.1 Modelo organizado por paquetes

Un modelo complejo en *SysML* puede contener miles de **elementos del modelo**. En *SysML* cada *elemento del modelo* es almacenado en un **contenedor** simple que es llamado el *propietario* o *pariente*. Los elementos contenidos son llamados *hijos*. Cuando un contenedor es borrado o copiado, sus hijos también son borrados. Este patrón de almacenamiento, significa que cualquier modelo en *SysML* tiene forma de una jerarquía tipo árbol. También, es posible que algunos elementos hijos puedan ser contenedores, lo que significa que representan a un contenedor anidado.

Los paquetes o *packages* o ***pkg*** son un ejemplo de un contenedor. Los elementos modelo contenidos en un paquete pueden mostrarse en un diagrama tipo paquete y la sintaxis del título del diagrama es el siguiente:

pkg *[tipo de element modelo] nombre del paquete [nombre del diagrama]*

En donde ***pkg*** indica el tipo de diagrama (paquete) y el tipo de elemento modelo puede ser modelo o *model*, paquete o *package*, librería del modelo o *model library,* o vista o *view*.

Cualquier elemento del modelo es contenido en exactamente un contenedor. Los elementos contenidos dentro de un paquete son llamados elementos empaquetables (ver Figura 2.2). Algunos ejemplos son bloques, diagramas tipo casos de uso y diagrama de actividades. Como los paquetes son elementos empaquetables, entonces pueden soportar una ***jerarquía de paquetes***.

Un modelo en *SysML* es un paquete de nivel alto en una jerarquía de paquetes anidados. En una jerarquía de paquetes, los modelos pueden contener otros modelos, paquetes o vistas. En adición a tener un lugar donde contenerse jerárquicamente, cada elemento modelo con nombre, debe ser parte de un espacio de nombres – *namespace* – lo que permite a sus elementos ser identificados con un nombre en particular y que el código fuente que sigue a la declaración se compile dentro de ese espacio de nombres. Dicho lo anterior, debe quedar claro que un paquete es un *namespace* para los elementos empaquetables contenidos en él.

SysML también permite una relación de dependencia entre los nombres de los elementos, que puede ser especificado a conveniencia para reflejar una semántica en particular. Estas dependencias son representadas mediante una flecha formada por guiones apuntando desde el elemento original hacia el elemento designado. Se explica las dependencias a continuación, y en la Figura 2.3 se observa un ejemplo práctico de su uso.

- Uso o *Use* – indica que el elemento original usa un tipo especial de unidad, valor o parte específica como parte de su definición y se especifica en el elemento designado.
- Refinamiento o *Refine* – indica que el elemento designado representa un incremento de detalle comparado a la especificación del elemento original.
- Realización o *Realization* – indica que el elemento original depende de un requerimiento que concierne al elemento designado.
- Trazo o *Trace* – indica que existe un vínculo entre el elemento original y el elemento designado sin imponer más restricciones semánticas.

- Asignación o *Allocate* – indica que el elemento designado es asignado por el elemento original.

Una organización efectiva facilita el re uso de los elementos modelo, además de acceso y navegabilidad entre ellos. Por eso, *SysML* incluye el concepto de **librería del modelo** o *model library*, la cual especifica y contiene los elementos modelo que pueden ser compartidos entre y dentro de modelos.

El concepto de **punto de vista** o *viewpoint* y vista o *view* también es importante. Un punro de vista describe una perspectiva de interés del cliente que usa para especificar una *vista* de un modelo. Una **vista** es un tipo de paquete y debe de estar conformado y referenciado por su **punto de vista**. Una relación conforme entre una vista y su punto de vista debe ser mostrada mediante una flecha de guiones y con la palabra `<<conform>>` en el medio. Una vista generalmente no contiene elementos, si no que importa elementos modelo para guardarlos en un *namespace* común para visualizarse vía paquete o en otro diagrama.

2.2 Modelo estructurado por bloques

Un **bloque** es la unidad modular de una estructura en *SysML*, es usada para definir un tipo de entidad lógica o conceptual. También, puede ser una entidad física (sistema),

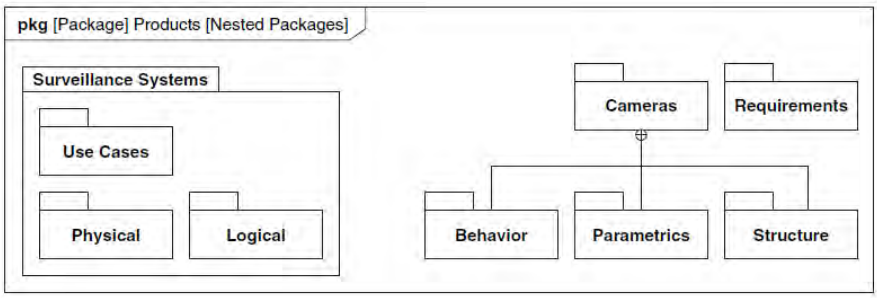


Figura 2.2 Paquetes en *SysML* Fuente [19]

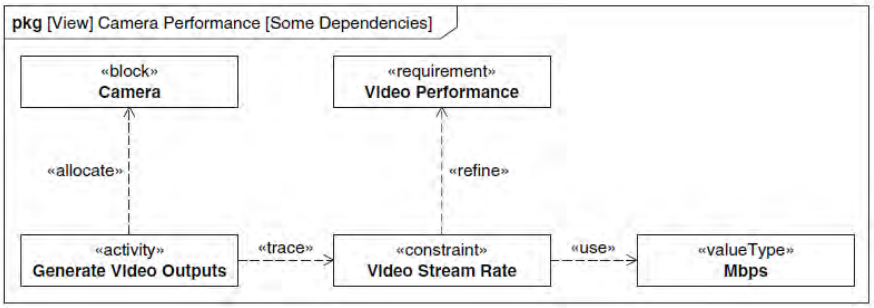


Figura 2.3 Dependencias entre bloques; Fuente [19]

hardware, software, persona, componente tipo dato, un ítem que fluye a través del sistema (como el agua) o una entidad en el medio ambiente (como la atmósfera).

Los bloques son un tipo o ***type***, quiere decir que son una descripción de un conjunto de *instancias o* ***instances****, objetos u* ***objects*** todos los cuales presentan características similares. Un bloque posee cualidades que sirven para describir las características de sus instancias. **Cualidades estructurales** para definir su estructura interna y propiedades. **Cualidades de comportamiento** para definir como interactúa con el entorno y como sus estados son modificados debido a esta interacción.

Una cualidad estructural son las *propiedades*, las cuales tienen diferentes formas de representarse como propiedades de **valores** o ***values***, que describen características cuantificables de un bloque (peso, velocidad); propiedades de **partes** o *parts*, que describen la descomposición de un bloque en sus elementos que lo constituyen y propiedades de **referencias** o *references*, cuyos valores hacen referencia a partes de otro bloque.

Los **valores** pueden ser de 3 tipos diferentes (ver Figura 2.4) **primitivos** o *primitives,* ayudan a definir valores escalares como *integer, boolean, string* y *real*; **enumeración** o *enumeration,* definen un conjunto de nombres con valores literales por ejemplo colores o días; tipo **estructural** o *structural type* representan una especificación de un dato estructural que incluye más de un elemento, cada uno pudiendo ser representado por un *value property.* Por ejemplo, *complex* es un tipo estructural pre definido por *SysML*, también puede ser 'posición' con *value properties* para *x,y* y *z*.

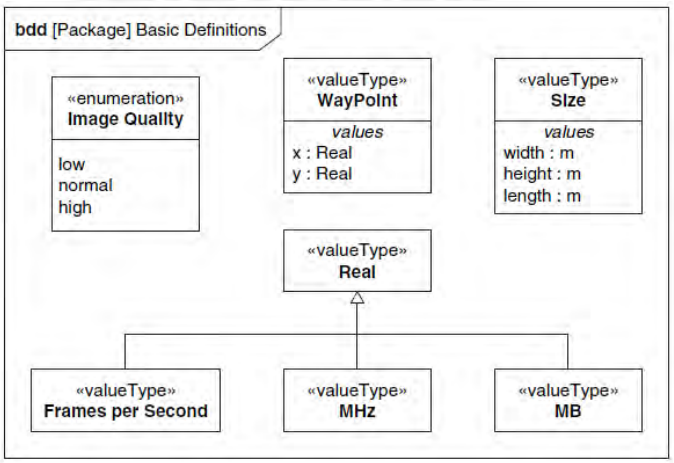


Figura 2.4 Definiciones básicas de valores en SysMl;
Fuente [19]

Además, *SysML* posee los conceptos de **tipo de cantidad** o *quantity kind* y **unidad** o *unit* (Figura 2.5). Un **tipo de cantidad** identifica un tipo de cantidad física como altura, cuyo valor puede estar definido en términos de metros. Una **unidad** debe estar relacionada siempre con un tipo de cantidad; sin embargo, no necesariamente un tipo de cantidad debe tener asociada una unidad y generalmente las ecuaciones pueden ser expresadas en términos de cantidades que incluyan *quantity kinds* sin especificar sus unidades.

Una vez definido los *value types,* estos pueden usarse en bloques usando la etiqueta *values.* Por otro lado, la diferencia principal entre un *part* y una instancia de un bloque es que un *part* describe una instancia o instancias de un bloque en su contexto; mientras que una instancia no requiere un contexto. Una parte puede estar definida de la siguiente manera:

nombre de la parte: nombre del bloque [multiplicidad]

Una instancia que compone un bloque puede incluir múltiples instancias que corresponden a las propiedades de partes. El número potencial de instancias es especificado por la multiplicidad de la propiedad de la parte, se define como:

- **Cota inferior** (mínimo número de instancias): puede ser 0 o un *integer* positivo. Este término es opcional cuando la cota inferior es cero porque una instancia del conjunto no está obligada a incluir todas las instancias de la parte.
- **Cota superior** (máximo número de instancias) puede ser 1, varios (denotado por "*", o cualquier *integer* positivo igual o mayor a la cota inferior.

Generalmente las cotas inferiores y superiores son expresadas de la siguiente manera: ***cota inferior. .cota superior***, excepto cuando tienen el mismo valor, en cuyo caso solo aparece la cota superior. Si no existe multiplicidad, se asume el valor de 1..1.

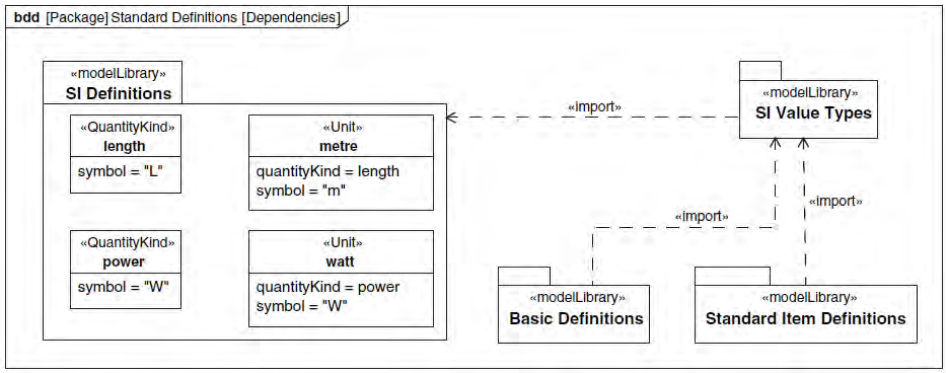


Figura 2.5 Tipo de cantidad y unidad; Fuente [19]

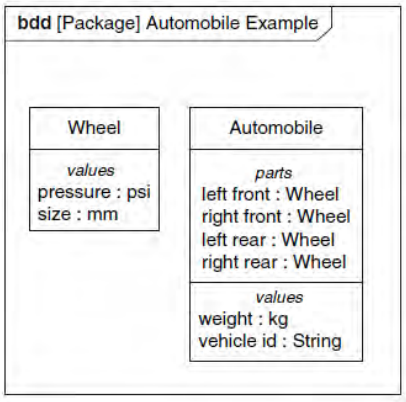


Figura 2.6 Automóvil descrito con 4 llantas (partes);
Fuente [19]

Un **diagrama de definición de bloques** o *block definition diagram*, se usa para definir bloques y su relación entre ellas, por ejemplo, una relación jerárquica. Por otra parte, puede especificar instancias entre bloques, incluyendo su configuración y valores de datos. Su sintaxis es la siguiente:

bdd *[tipo de elemento modelo] nombre del elemento modelo [nombre del diagrama]*

El tipo de diagrama es ***bdd***, el tipo de elemento modelo puede ser paquete, bloque o restricción de bloque.

SysML posee el concepto de **asociación compuesta,** la cual relaciona dos bloques en una relación tipo todo-parte (Ver Figura 2.7). Tiene 2 terminales, una describiendo el todo y la otra la parte. Esta asociación se representa mediante una línea entre 2 bloques con varios símbolos en sus terminales, la terminal que describe el todo se representa mediante un **rombo negro** mientras que la terminal que describe la parte mediante **una flecha**.

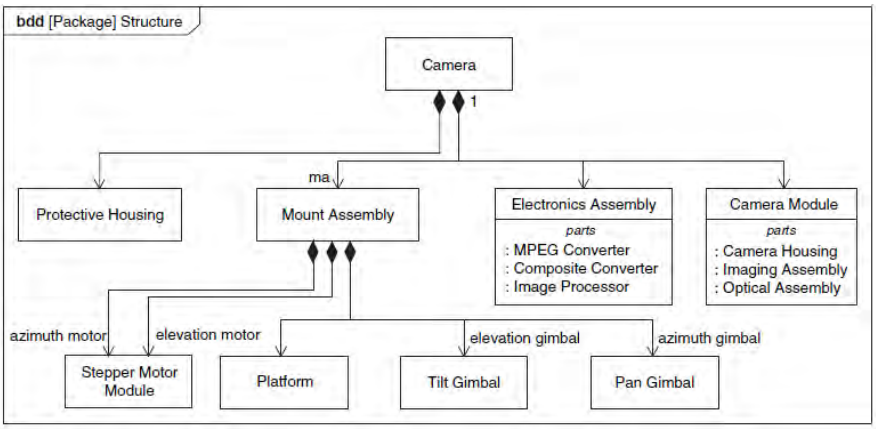


Figura 2.7 Asociación compuesta;
Fuente [19]

La terminal que describe la parte (flecha) es una vista de una propiedad de la parte que pertenece al bloque que describe el todo (rombo negro). Además, la terminal que describe el todo, provee información adicional como la multiplicidad o la cota límite superior (en esta terminal siempre es 1 porque una instancia de una parte solo puede existir en un todo en un tiempo determinado). La cota inferior puede ser 0 ó 1. En realidad, cada terminal puede mostrar multiplicidad y un nombre; sin embargo, si no se muestra esta característica significa que posee una multiplicidad de 0..1.

Una **asociación de referencia** se representa como una línea entre dos bloques (Ver Figura 2.8). Cuando hay una propiedad de referencia en solo una terminal, la línea tiene una flecha en la terminal que señala la asociación, es decir del dueño hacia el que es referenciado. Si la asociación es bidireccional, no hay flechas en ninguna de las terminales. Además, una terminal puede representar por un **rombo blanco**. *SysML* asigna el mismo significado a pesar de que no esté presente el rombo. La multiplicidad es igual que la asociación compuesta.

Un **diagrama de bloque interno** o *internal block diagram* se usa para describir la estructura de un bloque en términos de cómo sus partes estás interconectadas.

***ibd** [Block] nombre de bloque [nombre del diagrama]*

El marco de un diagrama de bloque interno siempre representa un *block.* El nombre del bloque es aquel designado por el marco del **bdd**. Por ejemplo, si se quiere hacer un **ibd** del bloque cámara, el *nombre de bloque* será cámara.

Las **propiedades de parte** pueden mostrarse como un bloque en donde su nombre en *string* está compuesto de el nombre de parte **seguido de dos puntos** y el tipo de parte.

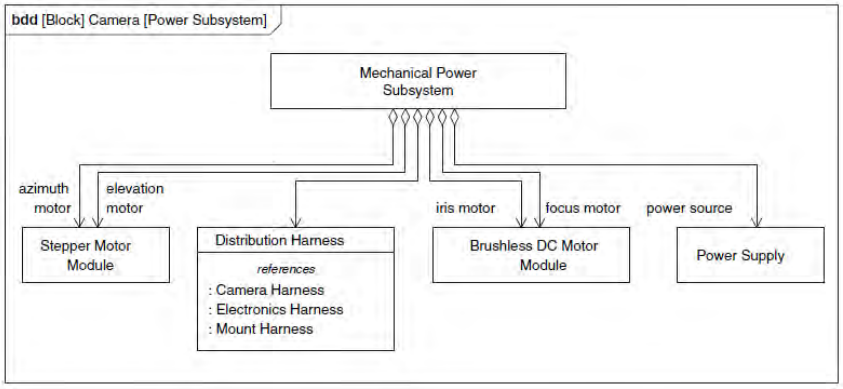


Figura 2.8 Asociación de referencia;
Fuente [19]

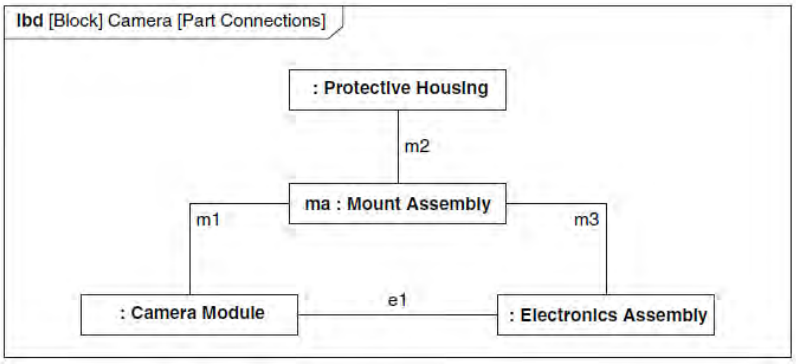
Las partes pueden estar vinculadas en un diagrama de bloque interno usando **conectores** (Ver Figura 2.9)que permiten la interacción de uno a otro, que incluyen la retransmisión de los ítems que fluyen tanto como entrado o como de salida de estos e invocando su comportamiento. La sintaxis de los conectores es:

nombre de conector: nombre de asociación

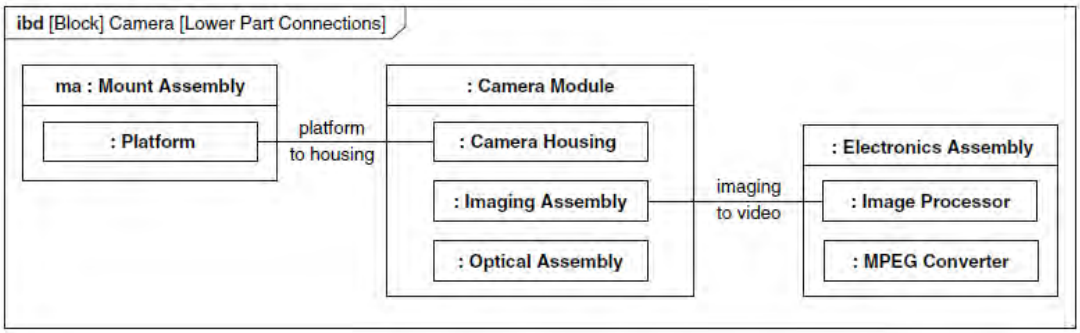
El terminal de un conector puede incluir una flecha, lo que significa que el tipo de asociación se muestra en un símbolo; normalmente no se muestra, y no debe confundirse con los flujos. También puede incluir nombre y multiplicidad. Cuando un conector cruza a otro **la intersección se designa por un semicírculo** para indicar que ambos no están conectados.

Los conectores también pueden conectar **puertos** o *ports* (Ver Figura 2.10 y Figura 2.11) que son otra cualidad estructural de un bloque, que especifica los puntos de acceso a los cuales el bloque puede interactuar con otro. Existen 2 tipos de puertos *full* y *proxy*. El *full port* es equivalente a una parte en el contorno de un bloque pariente como punto de acceso. Su sintaxis es:

*<<**full**>> nombre del puerto: nombre del bloque [multiplicidad]*

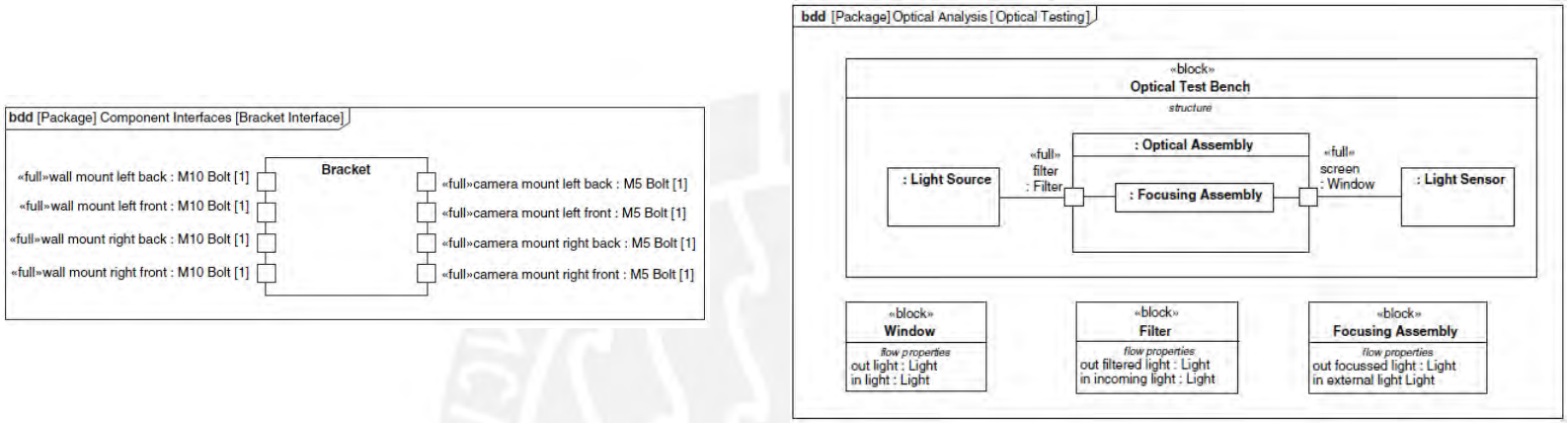


(a)



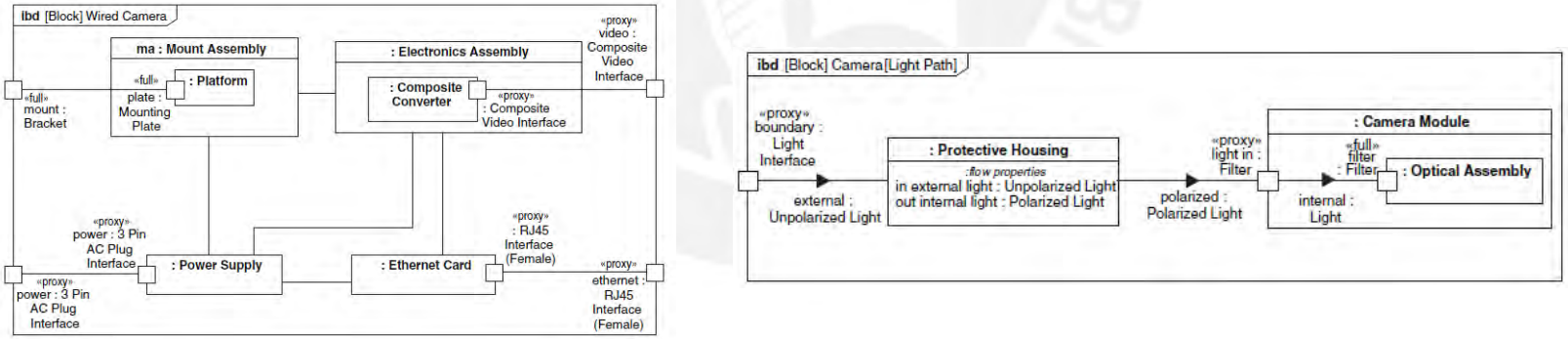
(b)

Figura 2.9 (a) y (b) Conexiones entre partes en **ibd**;
Fuente [19]



(a) (b)

Figura 2.10 (a) y (b) Puertos en **bdd**;
Fuente [19]



(a) (b)

Figura 2.11 (a) y (b) Conexiones entre puertos en **ibd**;
Fuente [19]

Un *proxy port* provee acceso externo a las características de su bloque pariente o la parte del bloque sin modificar sus entradas o salidas. Su sintaxis es:

<<***proxy***>> *nombre del puerto:nombre del bloque interfaz [multiplicidad]*

También el *port string* puede estar incluido en un compartimiento usando:

Dirección de puerto: bloque interfaz [multiplicidad]

Existen bloques que están relacionados con el comportamiento del sistema, es decir como el bloque responde ante un estímulo; sin embargo, se hablará de esto en las secciones 2.5 y 2.6.

Existen otras propiedades (flujo) las cuales el presente trabajo no abarcará debido al límite de hojas; sin embargo, se puede acceder a la bibliografía para mayor información.

2.3 Modelo paramétrico

Los modelos paramétricos contienen las limitaciones que el sistema puede tener. Generalmente estas limitaciones son propiedades que pueden modelarse mediante ecuaciones, que luego se pueden evaluar mediante una herramienta análisis (p. ej. *Matlab/Simulink*). Cabe resaltar que cada modelo paramétrico refleja un análisis de ingeniería en particular en un diseño.

SysML incluye **bloques de restricción** o *constraint blocks* que permiten la construcción de modelos paramétricos. Este bloque sigue un patrón de definición similar a los mencionados en la sección 2.2. Un bloque de restricción posee 2 elementos importantes: un conjunto de parámetros y una expresión que limita estos parámetros. Además, los usos de estos bloques se traducen en **propiedades de restricción**. La definición y uso de estos son representados en un diagrama de definición de bloques (**bdd**) como se muestra en la Figura 2.12.

La expresión dentro de un bloque de restricción puede ser una ecuación matemática y puede tener dependencia explícita del tiempo, como una derivada en una ecuación diferencial. Los parámetros de restricción no tienen una dirección para designarse como variables dependientes o independientes con respecto a la expresión principal.

En lugar de eso, la interpretación de dependencias entre los parámetros está basada en la semántica del lenguaje que se usa para especificar la expresión. Por ejemplo, en lenguaje C, la expresión $a = b + c$ indica que la variable a es dependiente del valor

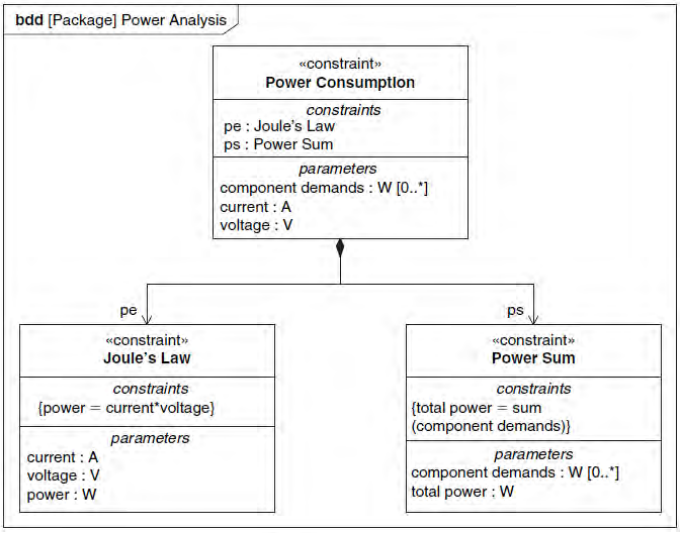


Figura 2.12 Modelo paramétrico en **bdd**; Fuente [19]

b y c, mientras que la expresión $a == b + c$, al ser una sentencia declarativa, no identifica la dependencia versus la independencia de las variables. Los parámetros también pueden ser restringidos mediante la especificación de **unidad específica** y **tipo de cantidad** (ver sección 2.2). Para crear sistemas de ecuaciones que puedan limitar las propiedades de los bloques se usan **diagramas paramétricos** o *parametric diagrams* (ver Figura 2.13). El cabezal de este diagrama es el siguiente:

par *[tipo de elemento modelo]nombre del elemento modelo [nombre del diagrama]*

El tipo de diagrama es ***par***, el tipo de elemento modelo puede ser bloque o un bloque de restricción.

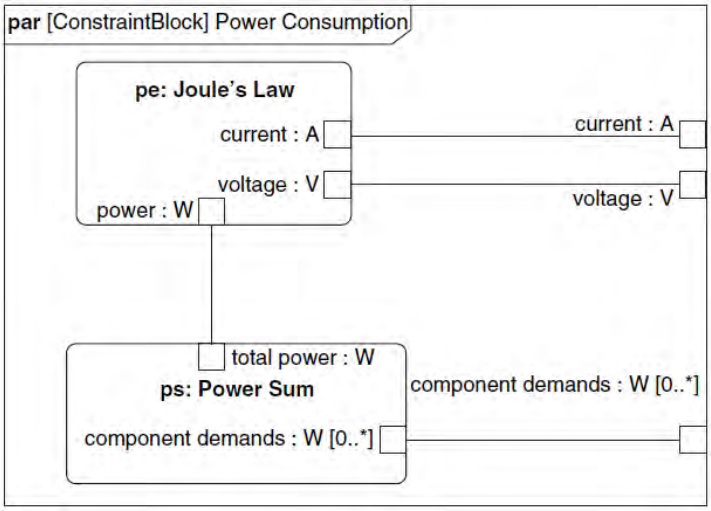


Figura 2.13 Diagrama paramétrico; Fuente [19]

El diagrama paramétrico muestra como las propiedades de restricción son conectadas mediante la unión de sus parámetros hacia otros y también de las propiedades de los valores de los bloques. Las conexiones de unión expresan igualdad entre los valores de restricción de los parámetros o de las propiedades de los valores y sus terminales.

En este sentido, los bloques de restricción pueden usarse para limitar los valores de las propiedades de los bloques. La especificación de los valores necesarios para la evaluación de la expresión delimitante de un bloque es usualmente descrita por la configuración de ese bloque, usando una especialización del bloque o una especificación de instancia.

Es importante resaltar que las restricciones pertenecen a cualquier elemento que sea un *namespace*, como un bloque, y que estas restricciones se muestran en un compartimiento especial etiquetado con el nombre de *constraints.* Aunque también puede ser representarse como un símbolo de nota adjunto al elemento modelo que restringe con el texto *constraint,* este último se muestra en el cuerpo de la nota (Ver Figura 2.14).

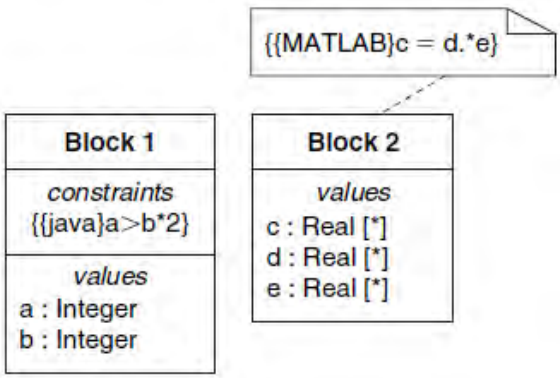


Figura 2.14 Formas de representar *constraints* ;
Fuente [19]

2.4 Modelo de comportamiento

En *SysML*, una **actividad** sirve para formalizar la descripción del comportamiento que especifica la transformación de las entradas en salidas a través de una secuencia de acciones controladas. El **diagrama de actividades** o *activity diagram* es la representación primaria para modelar un comportamiento basado en flujo. Las actividades proporcionan más características que los diagramas de flujo tradicionales, como la capacidad de expresar su relación con los aspectos estructurales del sistema (p.ej. bloques, partes) y la habilidad de modelar comportamientos de flujo continuo.

Este diagrama define las **acciones** en la actividad a lo largo del flujo de entrada/salida y el control entre ellos. Su sintaxis es la siguiente:

act *[tipo de elemento modelo]nombre de la actividad [nombre del diagrama]*

El tipo de diagrama es ***act***, el tipo de elemento modelo puede ser actividad o un **operador de control**. Este último genera valores de control mediante un parámetro de salida.

Las **acciones** o *actions* son los componentes básicos de las actividades y describen como estas se ejecutan. Cada acción puede aceptar entradas y producir salidas, llamadas ***tokens***. Los *tokens* son puestos en *buffers* de entrada y salida llamados pines o ***pins,*** hasta que estén listas para ser utilizadas. Estos *tokens* pueden corresponder a cualquier elemento que fluye, como por ejemplo información o un ítem físico (p.ej. agua). Aunque las acciones son el nivel más bajo del comportamiento de una actividad, hay ciertas clases de acciones como las denominadas **acciones de invocación**, que como su nombre indica, invocan acciones que se usan para componer las actividades en actividades jerárquicas.

El concepto de **flujo de objetos** (ver Figura 2.15) describe como los ítems de entrada y salida fluyen entre las acciones. Objetos que fluyen pueden conectar el pin de salida de una acción con el pin de entrada de otra permitiendo el paso de los *tokens.* El **flujo** puede ser **discreto o continuo**, en donde flujo continuo representa la situación cuando el tiempo entre *tokens* es aproximadamente cero.

Además, el concepto de flujo proporciona restricciones adicionales sobre cuándo y en qué orden las acciones entre actividades se ejecutarán. Un *token* en el **control flujo de entrada** permite habilitar una acción para comenzar la ejecución, mientras que un *token* de **salida** se presenta cuando la acción completa su ejecución. Cuando un flujo

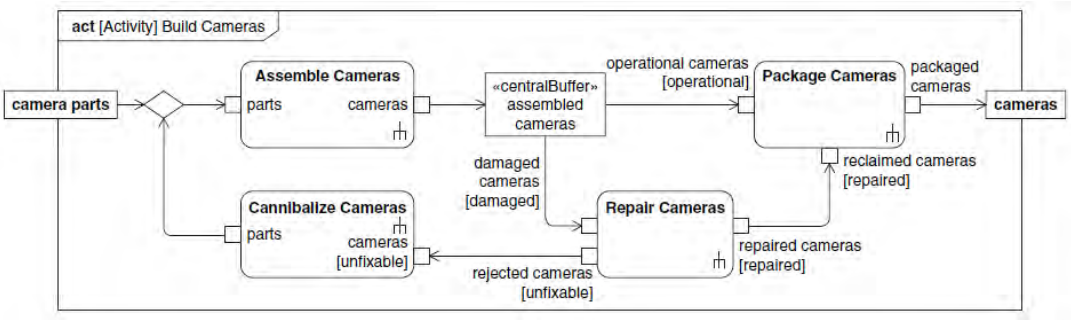


Figura 2.15 Ejemplo diagrama de actividades;
Fuente[19]

de control conecta una acción con otra, la acción a la que se dirige hacia la salida del flujo no puede comenzar mientras que la primera acción no haya sido completada.

Existen **nodos de control** como **junta** o *join*, **bifurcación** *o fork,* **decisión** o *decision*, **unión** o *merge*, inicial o *initial*, y **nodos de terminación** o *final nodes*, todos estos se usan para rutear los *tokens* de control con mayor especificación en la secuencia de acciones.

Existe el concepto de envío y recibimiento de señales, las cuales son un mecanismo para comunicar las actividades que se ejecutan en un contexto que abarque los diferentes bloques y a su vez para manejar eventos como interrupciones. Las **señales** o *signals* son usadas algunas veces como una entrada de control externa para iniciar una acción en una actividad que ya había comenzado a ejecutarse.

En la Figura 2.16 se muestra un diagrama de actividades que involucra varios de los conceptos mencionados.

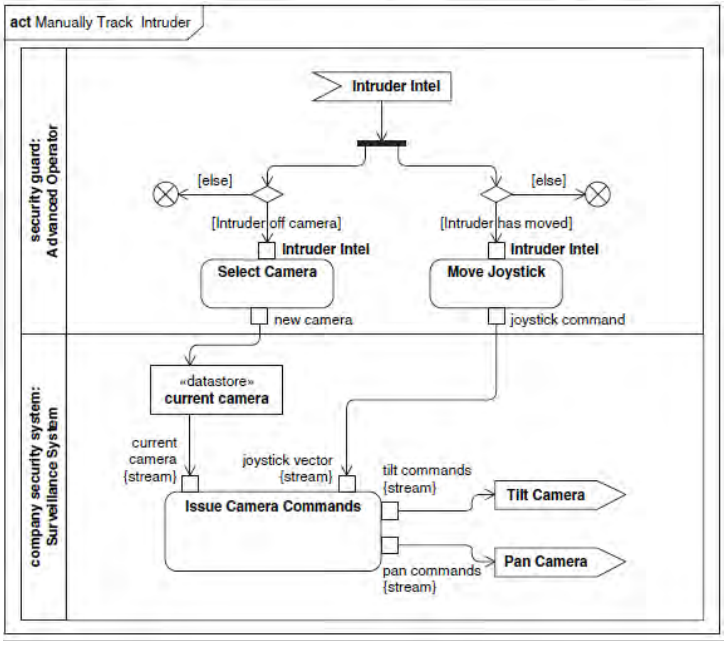


Figura 2.16 Diagrama de actividades más complejo;
Fuente[19]

2.5 Modelo de interacciones

En la sección 2.4 se modeló el comportamiento utilizando diagramas de actividades para representar una secuencia de acciones controladas que transformaban las entradas en salidas. Existen otras formas de modelar comportamiento, por ejemplo, cuando hay presencia de conceptos orientados a servicios, es decir, cuando una parte del sistema solicita un servicio de otra parte. Un **servicio** puede ser una interacción discreta entre

los componentes del software que ocurre cuando un componente solicita un servicio como una serie de operaciones a realizar.

Para representar la interacción entre elementos estructurales de un modelo puede usarse **diagramas de secuencia**. Estos elementos estructurales están representados por **líneas de tiempo**, que son una serie ordenada de **ocurrencias** específicas (enviar o recibir mensajes, crear o destruir un objeto, etc.). Dichas líneas de tiempo interactúan entre sí. La **interacción *o*** *interaction* puede ser entre el sistema y el medio que lo rodea o entre los componentes del sistema en cualquier nivel de la jerarquía del sistema a través de mensajes.

El diagrama de secuencias representa una interacción y su sintaxis es el siguiente:

sd *[interacción] nombre de la interacción [nombre del diagrama]*

El tipo de diagrama es ***sd*** y el elemento modelo solo puede ser interacción.

Las interacciones por si mismas pueden servir para manejar diversos escenarios o para permitir el re-uso de interacciones que siguen un patrón. Además, una interacción puede hacer referencia a otra para abstraer el nivel de detalle de una parte en múltiples líneas de tiempo o para referenciar una interacción entre las partes de una línea de tiempo en particular.

Como en otros tipos de comportamiento, una interacción puede ser un **clasificador** o *classifier behavior* esto quiere decir que comienza ejecutándose cuando una instancia del bloque es creada. Por otro lado, también puede ser **propietaria del comportamiento** u *owned behavior*, que significa que comienza la ejecución cuando es invocada. La interacción termina después de completarse la ejecución de su último fragmento.

Las interacciones poseen una cualidad estructural importante: **la línea de vida**, que representa la **línea de tiempo** más relevante de una **propiedad de una interacción** de un bloque. En *SysML* la interacción más básica se llama secuencia pobre, esta se muestra en la Figura 2.17 y lleva este nombre debido a que el orden de ocurrencias debe ser secuencial, no hay un orden entre ocurrencias en diferentes líneas de vida como en la Figura 2.18. En dicha figura se observa un símbolo rectangular vertical que corresponden a la ejecución, este concepto se llama **activaciones** o *activations***.** Las activaciones pueden ser de color blanco o gris.

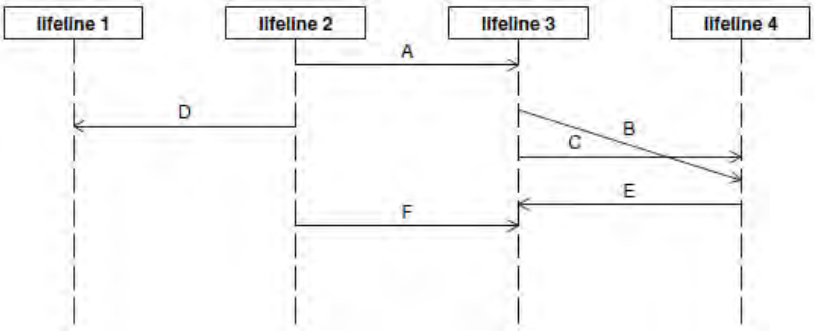


Figura 2.17 Secuencia pobre;
Fuente[19]

El concepto de **mensaje** puede representar tanto la invocación de un servicio o un envío de una señal en un componente del sistema, en ambos casos de la línea de vida emisora hacia la línea de vida receptora. Existen diferentes **tipos de mensajes** incluyendo **síncronos**, cuando el emisor espera una respuesta y **asíncrono**, cuando el emisor continúa sin esperar una respuesta.

Un **mensaje de creación** se representa por una línea punteada con una terminal con símbolo de flecha abierta. Un **mensaje de eliminación** posee una ocurrencia especial llamada ocurrencia de destrucción cuyo terminal posee 2 líneas cruzadas en forma de 'x'. Se ilustra estos conceptos en la Figura 2.19. Cada línea de tiempo representa una **instancia** y una interacción se ejecuta en el contexto de una instancia del bloque al que pertenece. Las **ocurrencias** aparecen cuando las instancias ejecutan sus comportamientos y envían y reciben peticiones que corresponden a llamadas de operaciones y señales.

Una secuencia de ocurrencias en un escenario de interés en particular se denomina **trazo** o *trace*. Cada interacción puede definir un conjunto de trazos válidos como también un conjunto de trazos inválidos. Un trazo válido se da cuando posee ocurrencias que son consistentes en la definición de su orden y es inválido cuando se usa un operando de interacción *neg* (se explica más adelante que significa).

Si se desea modelar ocurrencias más complejas que secuencias simples, las interacciones pueden incluir *contructs* especiales llamados fragmentos combinados. Un **fragmento combinado** tiene un operador y una serie de operandos, los cuales pueden ser interacciones de fragmentos primitivos. Hay cierto número de operandos de interacción que describen semánticas que ayudan a ordenar la secuencia, como paralelas, alternativas e iterativas.

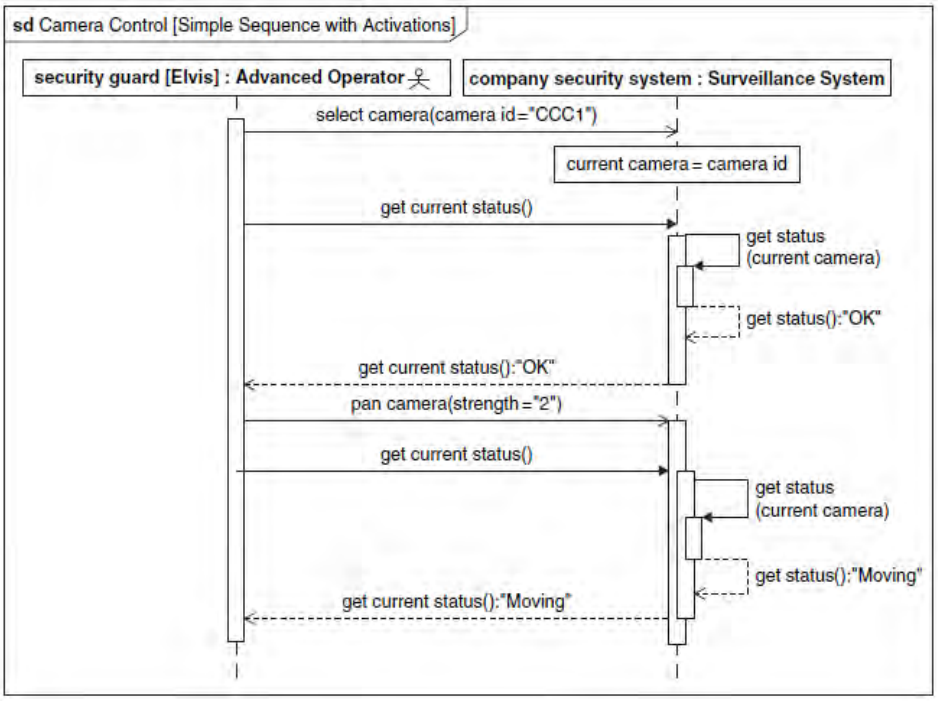


Figura 2.18 Secuencia con activaciones;
Fuente [19]

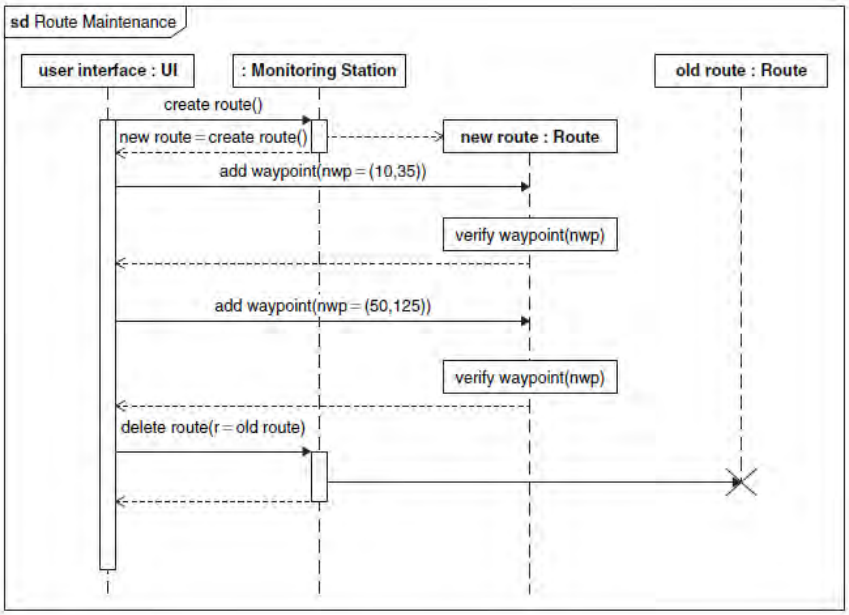


Figura 2.19 Creación y destrucción de mensajes;
Fuente [19]

Las **operandos de interacción** son: ***Seq***, hace referencia a la secuencia pobre, es la secuencia por defecto para cualquier operando, por ese motivo es raro que se encuentre explícito; ***Par***, un operador en donde sus operandos pueden ocurrir en paralelo, cada uno siguiendo una secuencia pobre; **Alt/Else**, indica que uno de sus operandos será seleccionado basado en el valor que contenga, es decir, primero se evalúa el valor que contiene y luego si contiene dicho valor entonces se selecciona; ***Opt***, es un operador de un solo elemento que equivale a un *alt* pero con solo un operador; ***Loop***, es un operador en donde el trazo representado por su operando se repite hasta alcanzar el valor de restricción que posee, que puede ser un límite superior (*upper bound*) o límite inferior (*lower bound*); ***Strict***, es como *seq* excepto que sus ocurrencias representadas por su operando son secuencias en un orden que atraviesa las líneas de vida; ***Break***, un operador cuyo operando es ejecutado en lugar de seguir el resto de la secuencia; ***Critical***, un operador en la cual la secuencia de operandos debe tener lugar sin la intervención de otros sucesos; ***Neg***, un operador que describe que los trazos en su operando son inválidos; ***Consider***, indica que solo considera mensajes específicos para un conjunto de operaciones o señales; e ***Ignore***, indica que no considera mensajes para un conjunto de operaciones o señales.

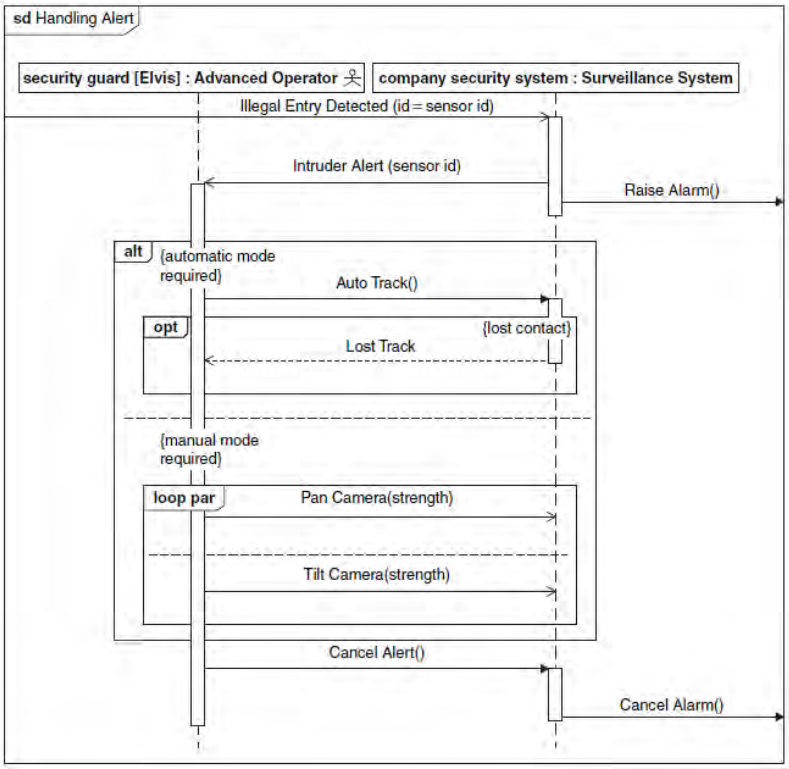


Figura 2.20 Uso de los operandos de interacción;
Fuente[19]

2.6 Modelo de máquinas de estado

Las máquinas de estado definen como el comportamiento de un bloque cambia mientras transita a través de diferentes estados. *SysML* cuenta con diagramas de máquinas de estado y su cabezal es el siguiente:

stm *[State Machine] nombre de la máquina de estado [nombre del diagrama]*

El tipo de diagrama es ***stm,*** y el tipo de elemento de modelo es siempre *State Machine*. El comportamiento de una máquina de estado está especificado en *SysML* por un conjunto de regiones, cada uno definiendo su propio conjunto de estados. Cada **región** está definida en términos de estados y pseudoestados. La diferencia entre estos es que una región no puede quedarse en un pseudoestado, ya que este último existe solamente para ayudar a pasar al siguiente estado.

Cabe mencionar que una región activa solo puede tener un estado activado dentro de él y que una máquina de estados puede tener múltiples regiones para describir algún tipo de comportamiento concurrente dentro de él.

Existen 2 pseudoestados importantes: el **pseudoestado arrancador** *(●)*, que se usa para especificar el estado inicial de una región y el **pseudoestado terminador** ***(X)*** que cuando es alcanzado indica que el comportamiento de la máquina de estado debe de terminar.

El concepto de estado hace referencia a una condición significativa en la vida del bloque. Esta condición, que puede expresarse en término de algún valor, representa un cambio significativo en como el bloque responde ante eventos y como su comportamiento se desarrolla. Un estado que debe de comprenderse, es el estado final (◉) que indica que la región ha sido completada y no más transiciones deben de suceder. Por otro lado, cada estado puede tener 3 comportamientos básicos denominados **actividades o comportamientos opacos**: de **entrada** o *entry***,** *de* **salida** *o exit* y de **realización** o *do.* Los primeros 2 se desarrollan cuando un estado entra o sale como su nombre lo indica y el tercero se desarrolla después de que el comportamiento de entrada se ha realizado. Los 3 comportamientos descritos se indican en *SysML* textualmente y seguidos de un ‘ / ’.

Una **transición** se representa mediante una flecha entre 2 estados, la cual apunta al siguiente estado. Las **transiciones fundamentales** son: **disparador** o *Trigger*; **espera** o *Guard* y **efecto** o *Effect*. La transición *trigger* indica el posible estimulo que puede

causar una transición y cuenta con **4** tipos de **eventos: señal** o *signal,* avisa cuando nuevo mensaje asíncrono correspondiente a una señal ha llegado; **tiempo** o *time***,** puede indicar tanto si ha pasado cierta cantidad de tiempo desde que se entró a un estado (tiempo relativo, se usa *after*) o si se ha alcanzado u instante determinado (tiempo absoluto, se usa *at*); **cambio** o *change*, indica que una condición ha sido satisfecha (se usa *when*) y de **llamada** o *call*, indica que una operación ha sido solicitada en el bloque propietario de la máquina de estado. La transición *guard* contiene una expresión (usando restricciones como en la sección 2.3 que deben evaluarse hasta que se cumpla con el valor *true* para que la transición ocurra. La transición *effect* es el comportamiento, de una actividad o de un comportamiento opaco, que es ejecutado durante la transición de un estado a otro. Se ilustran algunos de estos conceptos en la Figura 2.21.

Para mostrar gráficamente las transiciones *SysML* tiene algunas sintaxis especiales. Por ejemplo, si se desea representar un *trigger* se usa un rectángulo con un tríangulo faltante a la izquierda y si se desea representar una señal de envío de acción puede lograrse a través de un un rectángulo y un triángulo adicional a la derecha. Cualquier otro tipo de acción se muestra dentro de un rectángulo y escrito explícitamente con texto. Esto se ilustra en la Figura 2.22.

Existen otros pseudoestados que pueden usarse cuando la transición no es tan simple. Es el caso de **unión** o *junction* y **selección** o *choice*. El primero tiene el mismo símbolo que el pseudoestado arrancador (●) pero de menor tamaño. Se usa para construir transiciones compuestas. El segundo, tiene un símbolo de rombo y permite elegir un camino. Ambos conceptos se muestran en las Figura 2.23 y Figura 2.24.

Una máquina de estados también puede tener **puntos de entrada y de salida** que están situadas en el borde de un diagrama de máquina de estados o de un estado compuesto. Se señalan mediante círculos (el de entrada vacío y el de salida contiene una X) y son similares al pseudoestado *junction*. Se ilustra esto en la Figura 2.24.

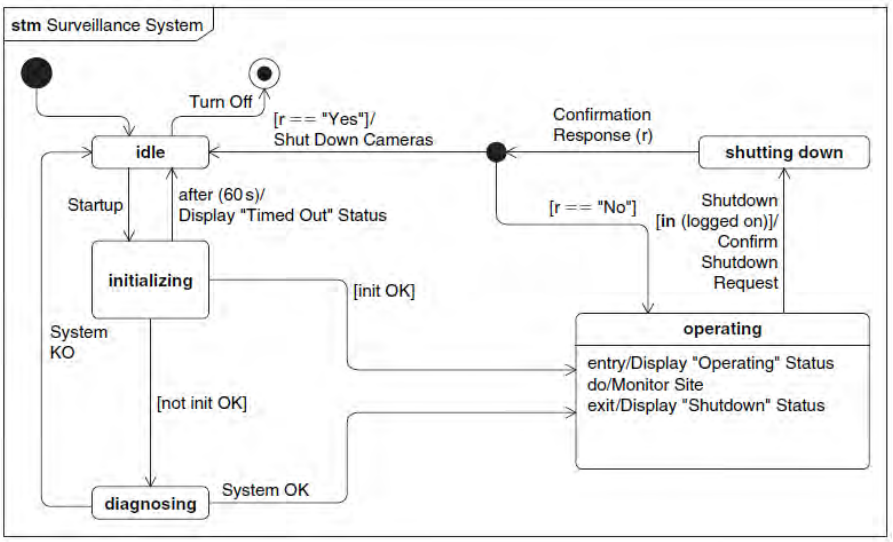


Figura 2.21 Ejemplo de diagramas de máquinas de estado;
Fuente [19]

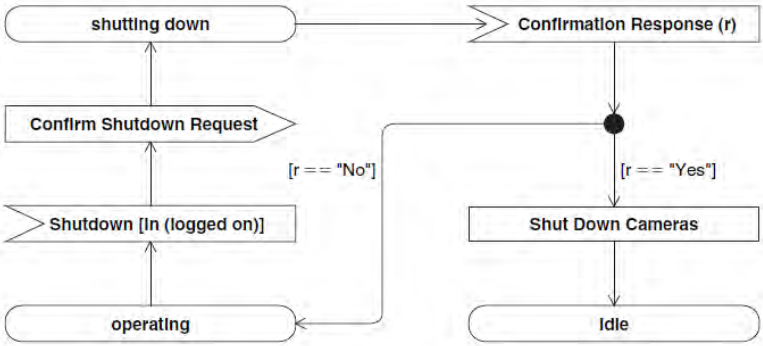


Figura 2.22 Transiciones en *SysML*; Fuente[19]

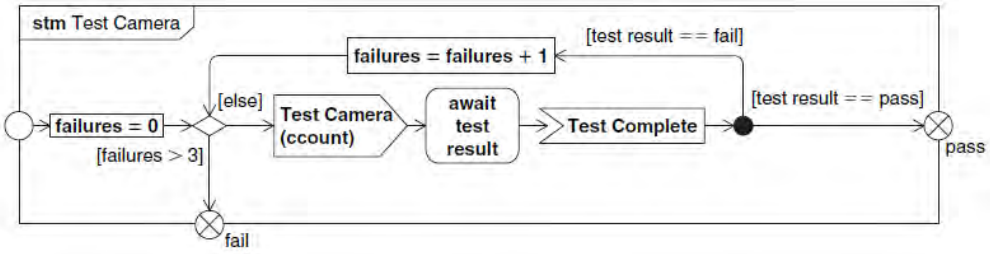


Figura 2.23 Pseudoestado unión; Fuente [19]

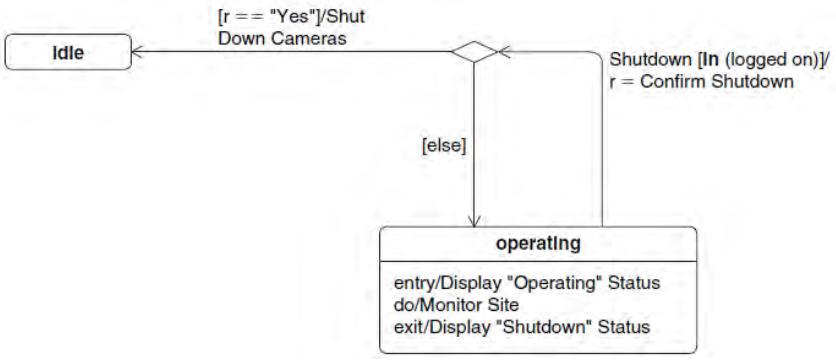


Figura 2.24 Pseudoestado selección; Fuente[19]

2.7 Modelo de funcionalidad

Modelar la funcionalidad de un sistema implica describir como el sistema se usa de manera en particular para alcanzar un número determinado de objetivos. Esto puede realizarse mediante los **diagramas de casos de uso** o *use cases*. El cabezal de un diagrama de casos de uso es el siguiente:

uc*[tipo de elemento modelo]nombre del elemento modelo [nombre del diagrama]*

El tipo de diagrama es **uc** el tipo de elemento modelo puede ser *package* o *block*.

Los usuarios del sistema pueden describirse mediante **actores** o *actors***,** ellos representan usualmente el rol de una persona, organización o cualquier entidad externa (p. ej. un sistema) que participa en el uso de algún sistema. Los actores pueden interactuar directamente con el sistema o indirectamente con otros actores. Un actor puede representarse mediante una persona de palitos () con su nombre debajo de esta figura o como un rectángulo conteniendo la palabra <<*actor*>>. Cabe resaltar que el término actor es relativo porque puede suceder, que un actor que es externo a un sistema sea interno a otro. Un ejemplo se muestra en la Figura 2.25, en donde se indica que un objeto padre (el actor operador avanzado) señala a su hijo (actor operador) con una línea y un triángulo al final de esta línea.

El sistema que provee funcionalidad, ayuda al diagrama *uses cases* y el sistema a desarrollar que se llama el **sistema bajo consideración** (también es llamado el sujeto) y se representa mediante un *block*.

Los **casos de uso** se representan mediante óvalos y el en el medio su nombre. Pueden tener multiplicidad y por defecto (si no se muestra) es "0..1". Los actores se relacionan con el caso de uso mediante asociaciones y referencias, la cual puede tener en su terminación alguna multiplicidad.

Las referencias pueden ser de **inclusión** o *include*, que incluye la funcionalidad de un caso de uso en otro; y de **extensión** o *extend*, que extiende la funcionalidad (como un comportamiento adicional) de un caso de uso. Por otro lado, las asociaciones no pueden tener flechas, pero las referencias sí (en línea punteada). Algunos de estos conceptos se muestran en la Figura 2.26.

Es importante mencionar que los casos de uso y actores pueden verse descritos en contextos diagramas de bloque interno) como en comportamientos. Esto quiere decir que se les puede encontrar en diagramas secuencia (ver sección 2.5 Figura 2.18), de actividades e incluso en diagramas de máquinas de estado.

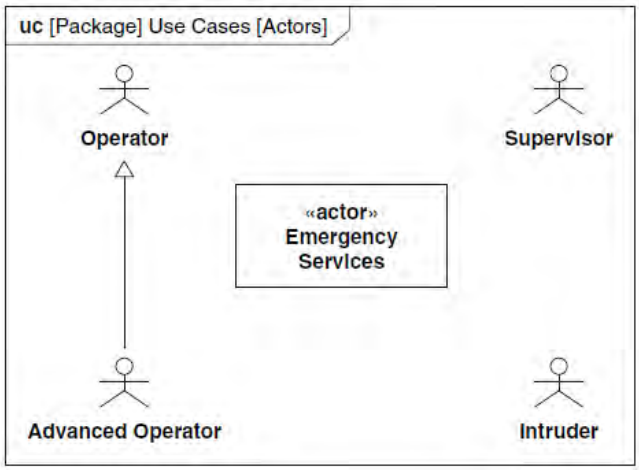


Figura 2.25; Ejemplo diagrama de casos uso;
Fuente [19]

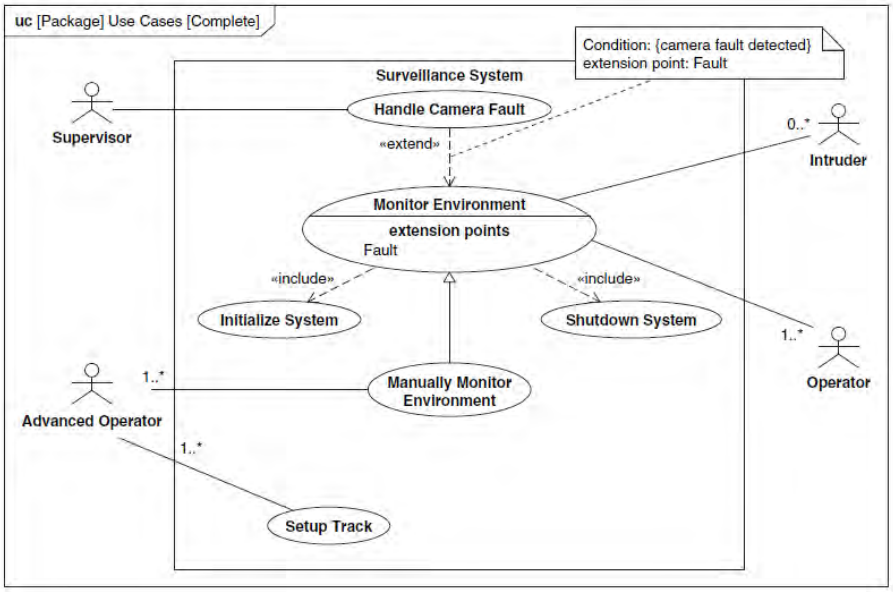


Figura 2.26 Otros conceptos de casos de uso;
Fuente [19]

CAPÍTULO 3

TRAZABILIDAD DE REQUERIMIENTOS DE DISEÑO USANDO *SysML*

3.1 Requerimientos y su relación con el diseño en SYSML

En la introducción y el capítulo uno del presente trabajo se explicó por qué los requerimientos son fundamentales para el diseño. Por esa razón, debe haber un nexo entre la creación de la lista de requerimientos y una herramienta que pueda modelar ak sistema en *SysML* para verificar su cumplimiento. Además, se debe recordar que todos los requerimientos (al provenir de la lista) en *SysML* son **requisitos textuales** o **requisitos basados en texto**. *SysML* permite utilizar requerimientos que especifiquen tanto:

- Una capacidad o condición que debe ser satisfecha;
- Una función que el sistema debe desempeñar;
- Un comportamiento que el sistema este condicionado a alcanzar.

3.2 Modelo de requerimientos en *SysML*

Un conjunto de requerimientos similares de un sistema, elemento o componente pueden ser agrupados en una **especificación**. Es decir, cada especificación puede contener múltiples requisitos que pueden modelarse en una estructura tipo árbol. Luego, estos requerimientos pueden vincularse con otros requerimientos pertenecientes a diferentes especificaciones y a elementos modelos que representen el diseño del sistema, análisis, implementación y testeo. Las especificaciones suelen estar organizadas en un modelo contenido en un paquete de estructura jerárquica.

Los requerimientos capturados en *SysML* pueden mostrarse en un **diagrama de requerimientos** o *requirement diagram*. Su sintaxis es:

req *[tipo de elemento modelo]nombre del elemento modelo [nombre del diagrama]*

El tipo de elemento modelo puede ser *package* o *requirement*. El nombre del elemento modelo es nombre del *package* o *requirement* y el nombre del diagrama es definido por el usario.

Un requisito basado en texto se representa en *SysML* usando el elemento modelo <<*requirement*>>. Cada requerimiento incluye propiedades predefinidas para un identificador único (**id**) y un texto en *string*, esto se ilustra en la Figura 3.1

La Tabla 3-1 muestra algunos estereotipos de requerimientos que pueden usarse en *SysML*. Existen formas alternativas para crear y especificar requerimientos. Sin embargo, no se abarcará aquellos procedimientos en este documento debido a que sería muy extenso. Más información puede encontrarse en la bibliografía [2,23]

Este diagrama puede contener un gran número de relaciones para un solo requerimiento, lo cual es bastante útil para representar la trazabilidad de un

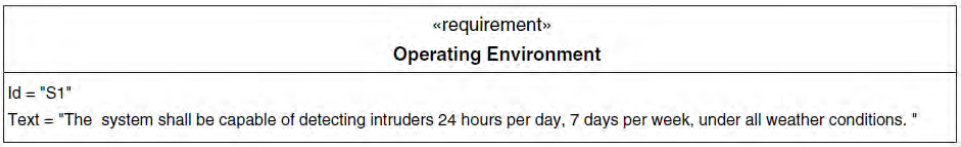


Figura 3.1 Requerimiento o *requirement*;
Fuente[19]

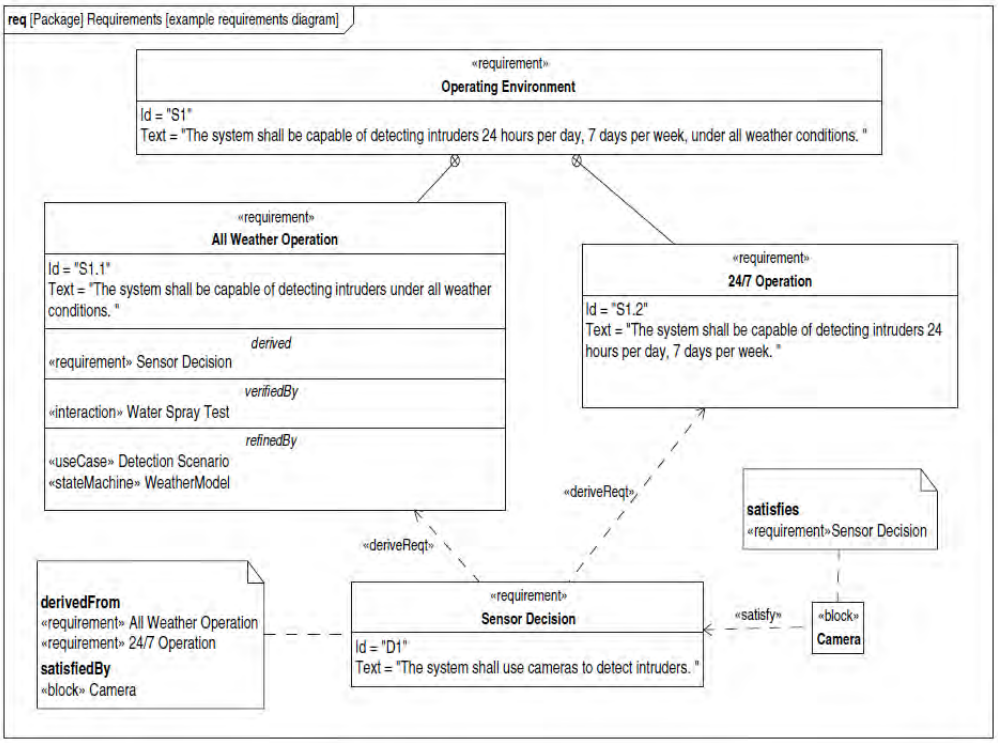


Figura 3.2 Diagrama de requerimientos en *SysML*;
Fuente[19]

requerimiento en particular. Por eso *SysML* cuenta con los conceptos (ver Tabla 3-2) que permiten examinar ese requerimiento través de vínculos como: **satisfecho** o *satisfied o satisfy*, **verificado** *o verified o verify*, **afinado** o *refined o refine y* ***trazado*** *o trace*. También existen las relaciones de **copia** o *copy* y **derivado** o *derive*, estos últimos solo pueden relacionar de un requerimiento a otro, los demás pueden relacionar a otros elementos modelo. En la Figura 3.2 se muestra un ejemplo de un diagrama de requerimientos ilustrando algunas relaciones mencionadas.

Tabla 3-1 Requerimientos Opcionales que pueden usarse en SysMl; Fuente:[19]

Estereotipo	Clase base	Descripción
<<extendedRequirement>>	<<requirement>>	Un estereotipo adicional que contiene atributos útiles para requerimientos.
<<functionalRequirement>>	<<extendedRequirement>>	Requerimiento que especifica una operación o un comportamiento de un sistema, o de una parte de un sistema que debe realizarse.
<<interfaceRequirement>>	<<extendedRequirement>>	Requerimiento que especifica los puertos que conectan sistemas y partes del sistema y que opcionalmente pueden incluir flujo de ítems que atraviesan conectores y/o restricciones de interfaz.
<<performanceRequirement>>	<<extendedRequirement>>	Requerimiento que mide cuantitativamente la salida de un sistema, o de la parte de un sistema, satisfaciendo el requisito de capacidad o condición.
<<physicalRequirement>>	<<extendedRequirement>>	Requerimiento que especifica las características y/o restricciones físicas de un sistema o parte de un sistema
<<designConstraint>>	<<extendedRequirement>>	Requerimiento que especifica una restricción en la implementación de diseño de un sistema, o parte de un sistema.

Tabla 3-2 Notación de las relaciones entre requerimientos; Fuente [19]

Relación	Designación
Satisfacción	<<satisfy>>
Verificación	<<verify>>
Afinación	<<refine>>
Requerimiento Derivado	<<deriveReq>>
Copia	<<copy>>
Trazado	<<trace>>

Representar las relaciones entre diferentes diagramas en *SysML* es un punto clave para lograr la trazabilidad. Nos importa saber cómo los diferentes diagramas pueden verse perturbados. Por eso, es importante saber que los vínculos entre los requerimientos y otros elementos modelos pueden representarse directamente si el requerimiento y los elementos modelos están en el mismo diagrama. Si es que no se encontrara en el mismo diagrama puede usarse un compartimiento o una notación tipo *callout*. Estos conceptos se ilustran en la Figura 3.3 - Figura 3.5

En todos los casos es clave reconocer la dirección de la flecha debido a que la mayoría de estas relaciones en *SysML* están basadas en la relación de dependencia de *UML* [2,23]. La flecha apunta desde elemento modelo dependiente (llamado el cliente) y va hacia el elemento modelo independiente (llamado el proveedor). Por ejemplo, en la Figura 3.3, el diseño de la cámara es un modelo dependiente del requerimiento, lo que significa que, si el requerimiento cambia, entonces el diseño debe de cambiar. Similarmente, un requerimiento derivado será dependiente del requerimiento del cual se deriva. En *SysML,* la dirección de la cabeza de la flecha es opuesta a lo que

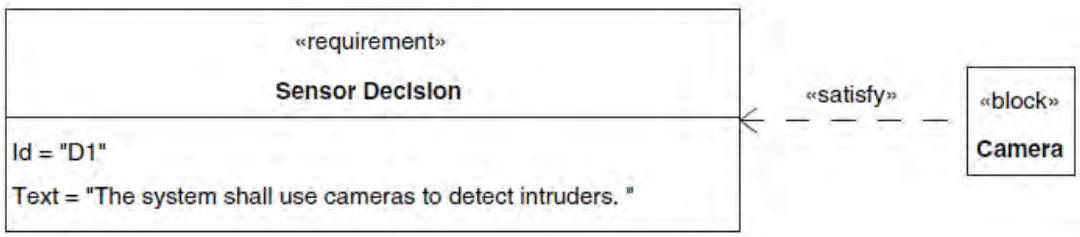


Figura 3.3 Notación directa de una relación de un requerimiento;
Fuente:[19]

típicamente se usa para requerimientos de flujo hacia abajo en donde, el requerimiento de nivel alto apunta al de nivel bajo.

Un concepto que ofrece SysML y vale la pena mencionar es el de *rationale* el cual es un elemento modelo que puede asociarse tanto con un requerimiento, una relación entre requerimientos o cualquier otro elemento modelo. Permite almacenar la razón en particular de una decisión. Además, a pesar de que este concepto se muestra para requerimientos, puede ser utilizado también para cualquier tipo de decisión. Por otro lado, existe el concepto de **problema,** que identifica como su nombre indica, el problema en particular que necesita ser resuelto. En ambos casos se representa por un símbolo de nota en la cual incluye la palabra `<<rationale>>` o `<<problem>>.` En la Figura 3.6 se muestra un ejemplo con el concepto de *rationale*.

No obstante, el diagrama de requerimientos posee una desventaja distintiva cuando el sistema a diseñar aumenta su número de requerimientos. Un número grande de señalizaciones se necesitan para relacionar todos los requisitos y el método tradicional para ver la lista de requerimientos es más compacto y efectivo. Además, las herramientas modernas de gestión de requerimientos suelen tener una base de datos y estos se muestran de una manera clara y sucinta en tablas o matrices. *SysML* también permite mostrar estos resultados de esa manera. Se ilustra un ejemplo en la Figura 3.7, en la cual se visualiza los mismos requerimientos que de la Figura 3.2. Otros ejemplos se muestran la Figura 3.8 y Figura 3.9.

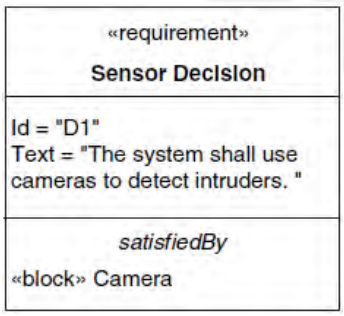


Figura 3.4 Notación en comportamiento; Fuente [19]

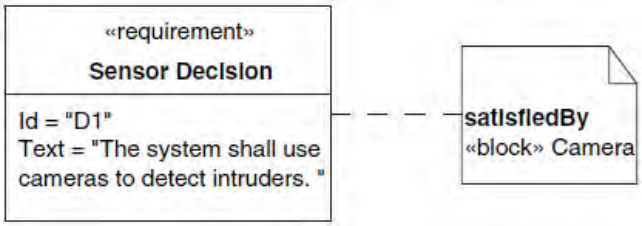


Figura 3.5 Notación tipo *callout;* Fuente [19]

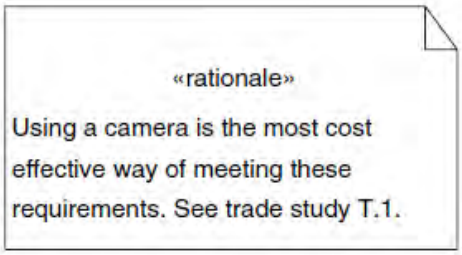


Figura 3.6 Concepto de *rationale* en *SysML*;
Fuente [19]

table [Package] System Specification [Decomposition of top-level requirements]

id	name	text
S1	Operating Environment	The system shall be capable of detecting intruders 24 hours per day…
S1.1	All Weather Operation	The system shall be capable of detecting intruders under all weather…
S1.2	24/7 Operation	The system shall detect intruders 24 hours per day, 7 days per week
S2	Availability	The system shall exhibit an operational availability (Ao) of 0.999…

Figura 3.7 Ejemplo de tabla de requerimientos;
Fuente[19]

table [Requirement] Camera Decision [requirements tree]

id	name	relation	id	name	Rationale
D1	Sensor Decision	deriveReqt	S1.2	24/7 Operation	Using a camera is the most cost-effective way of meeting these requirements. See trade study T1.
		deriveReqt	S1.1	All Weather Operation	Using a camera is the most cost-effective way of meeting these requirements. See trade study T1.

Figura 3.8 Tabla de requerimientos derivados;
Fuente [19]

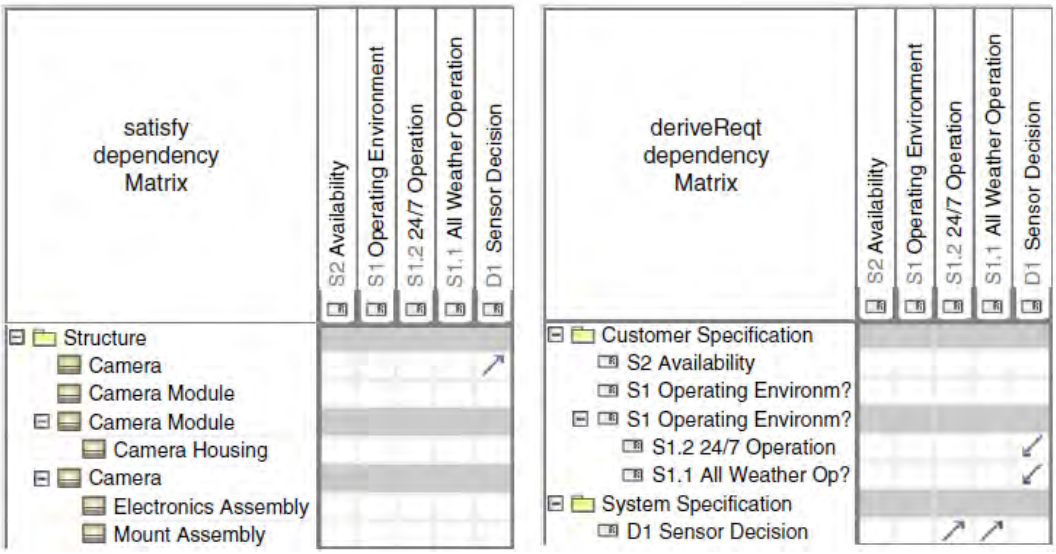


Figura 3.9 Matriz de relaciones de requerimientos;
Fuente [19]

CAPÍTULO 4
CASO DE ESTUDIO

En este capítulo se detalla cómo realizar el modelamiento de un sistema mecatrónico en *SysML,* haciendo uso de los modelos descritos en los capítulos 2.0 y 3.0 y el software *MagicDraw* [7], para garantizar la trazabilidad de requerimientos. Existen varios requisitos que deben verificarse con los modelos en *SysML*; sin embargo, esta tesis se centrará en la validación del modelo paramétrico y el de funcionalidad. En este sentido, se utiliza el *plugin ParaMagic* para vincular *Matlab/Simulink* y procesar el control del sistema. Adicionalmente, este *plugin* servirá para comunicar *MagicDraw* con *Excel,* en donde una hoja de cálculo mostrará si los requerimientos se están cumpliendo.

4.1 Sistema mecatrónico

El sistema mecatrónico para este caso de estudio en particular corresponde a un **Actuador Electro-Mecánico** (EMA). El EMA cumple como sistema mecatrónico ya que comprende parte electrónica, mecánica y de control, se muestra el diseño de control situado en un avión comercial que debe mover la carga de una superficie de control, para el ejemplo, un alerón.

4.1.1 Contexto

Un avión (desde un avión de alto rendimiento hasta el más simple) está gobernado por los principios básicos de control de vuelo [23], en dónde hay dos movimientos principales: traslación y rotación.

- Traslación es el movimiento que hace el vehículo aéreo cuando viaja de un punto hacia otro en el espacio.

Rotación, la rotación de un avión depende de 3 ejes: *pitch*, *roll* y *yaw* (ver Figura 4.1). En donde *roll* implica que el avión rota sobre su eje longitudinal (ver Figura 4.2); *pitch*

implica variar la nariz del avión de tal manera que este puede ascender o descender e *yaw* implica mover la dirección de la nariz del avión de tal manera que este pueda desplazarse en el plano donde se encuentra el eje longitudinal y el eje lateral. Los aviones son un sistema complejo que tienen varios parámetros que controlar para lograr un vuelo estable. Por ejemplo, es difícil volar si el avión necesita a ir a una velocidad alta porque esto produce cargas elevadas que deben controlarse en las superficies del avión.

Las superficies de control del vehículo aéreo permiten al piloto maniobrar el *pitch, roll* o *yaw.* Para vuelos comerciales el control del *pitch* debe darse por 4 elevadores situados en la cola del avión [23]. Cada elevador cuenta con un actuador de vuelo dedicado, recibiendo potencia gracias a un sistema hidráulico o electro-mecánico. El control del *roll* es posible debido a dos alerones situados en las alas del avión, de la misma manera cada alerón es alimentado independientemente. El control del *yaw* es proporcionado por 3 rudders, que como el *roll* y el *pitch* son independientes. Las superficies de control de un avión de la familia Airbus A320 se muestra en la Figura 4.3.

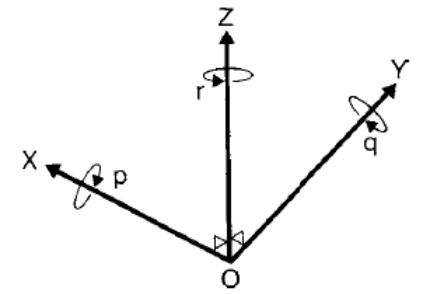


Figura 4.1 Definición de los ejes de control;
Fuente [23]

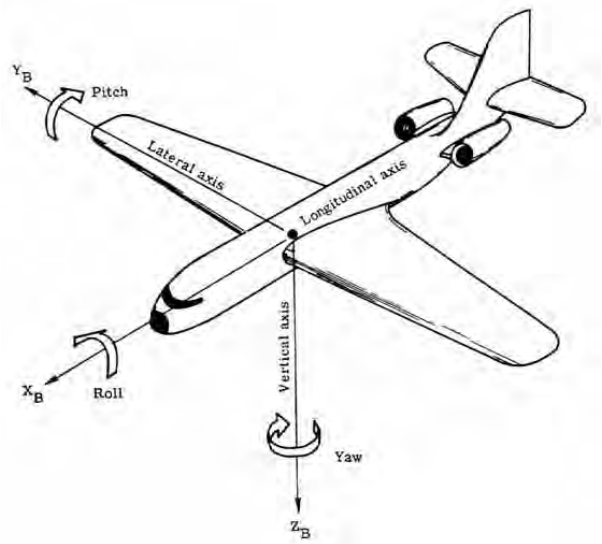


Figura 4.2 *Pitch*, *roll* y *yaw* situados en el avión;
Fuente: http://history.nasa.gov/SP-367/appendc.htm

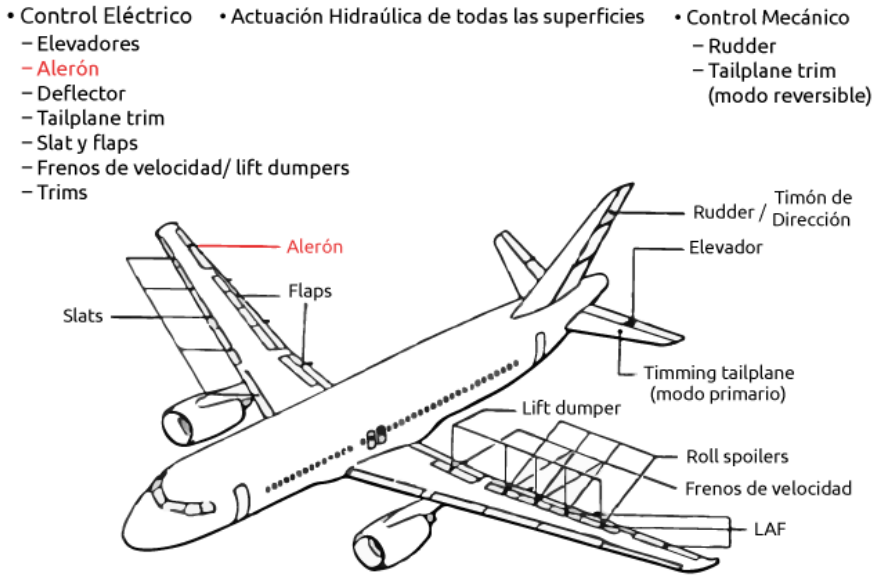


Figura 4.3 Ejemplos de superficies de control de un avión tipo A320;
Fuente [23]

El piloto se encuentra en la nariz del avión y realiza ciertas acciones en la cabina de vuelo, enviando entradas (información) al controlador del avión. Además, se sabe que la información que puede enviar varía según [19]:

- El control del *pitch* se realiza moviendo el timón de profundidad en un movimiento hacia adelante o hacia atrás.
- El control del *roll* se realiza moviendo el timón de profundidad hacia la izquierda o hacia la derecha.

- El control del *yaw* se realiza con 2 pedales de dirección, el pedal de la izquierda hace varia la dirección en ese sentido, mientras que el de la derecha en el otro.

El timón de dirección y los pedales de dirección envían al controlador señales electrónicas emitidas mediante cable, a esto se le llama tecnología *fly-by-wire* (FBW).

4.1.2 Actuador Electro-Mecánico (EMA)

La industria aeroespacial ha desarrollado sistemas de actuación que son alimentados eléctricamente usando FBW, resultando en lo que se llama *power-by-wire* (PBW) [24]. Estos sistemas son conocidos como actuadores electro-mecánicos (EMA) que están constituidos de tres partes principales [18]: un sistema eléctrico, un sistema mecánico y un sistema electrónico y de software.

Cabe resaltar que el EMA cuenta con una unidad que está constituida por microcontrolador (uC), el cual recibe la posición deseada entregada por la unidad de control y un *driver*, que permite regular la potencia para mover la carga. Además, el ángulo es monitoreado a través de un sistema de transformación diferencial variable de rotación (RVDT de sus siglas en inglés), el cual es un transductor electro-mecánico que posee como entrada corriente alterna y que su salida es linealmente proporcional al desplazamiento angular del eje. Un esquema del actuador electro-mecánico se muestra en la Figura 4.4.

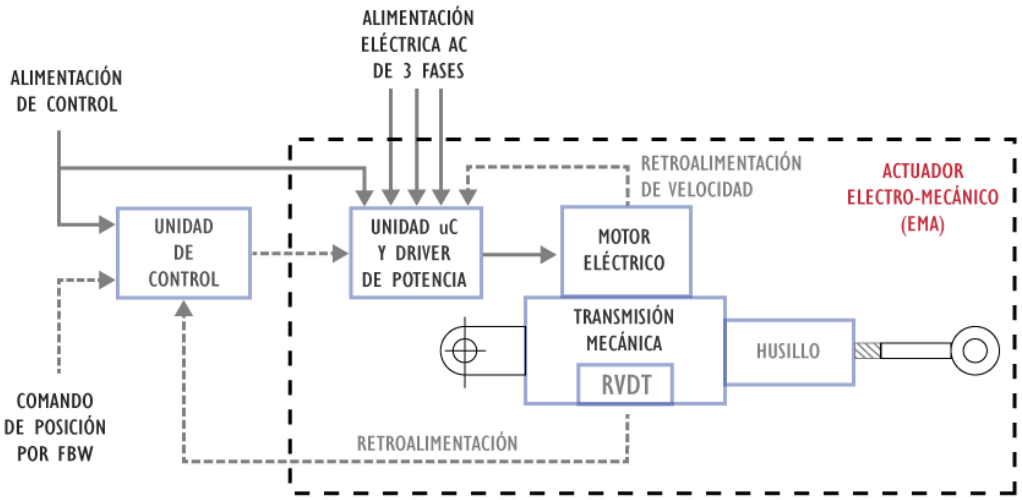


Figura 4.4 Actuador electro-mecánico(EMA);
Fuente adaptado de [23]

4.2 Modelación del sistema en *SysML*

El primer paso para modelar el EMA en *SysML* es definir los modelos que contendrá el sistema. Para poder realizar esto se utiliza un diagrama de paquetes (sección 2.1). En este caso de estudio, hay un modelo para requerimientos, un modelo para la estructura del EMA, un modelo de casos de uso del sistema, un modelo de comportamiento y un modelo paramétrico. En la Figura 4.5 se muestran los diagramas mencionados; en contraste con la Figura 2.1, el diagrama de casos de uso es independiente del diagrama de comportamiento dado que se realiza un análisis que explica la ubicación y el contexto del sistema analizar, el EMA.

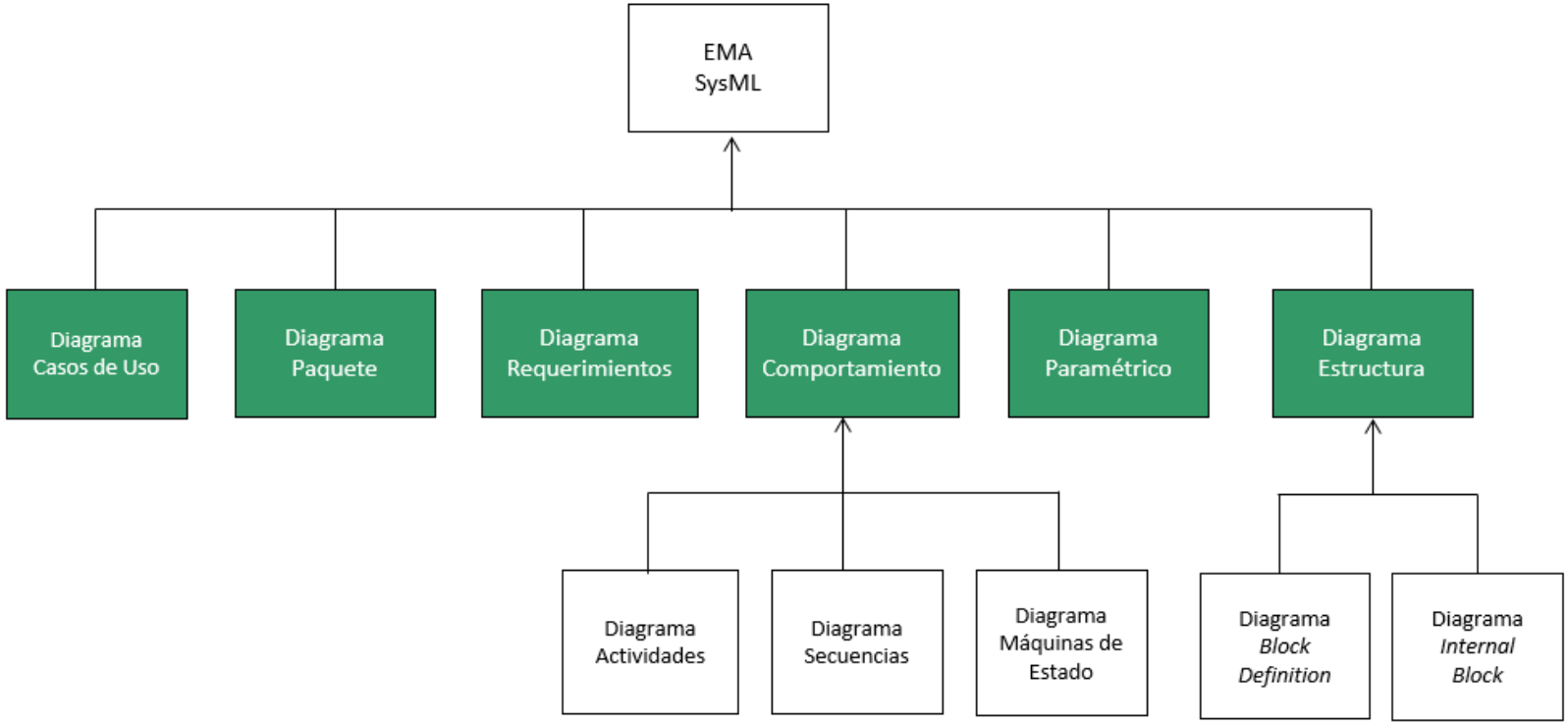


Figura 4.5 Diagrama de modelación en SysML;
Fuente propia

4.2.1 Diagrama de paquetes

En la Figura 4.6 se muestran los paquetes que tendrá el modelamiento. Todos estos están agrupados en un contenedor tipo paquete llamado *Data*. Los elementos modelo son:

i. Requerimientos: contiene los requisitos que el sistema debe de cumplir
ii. Estructura: contiene los elementos del actuador electro-mecánico
iii. Comportamiento: contiene los diagramas de actividades, secuencia y máquinas de estado
iv. Paramétrico: contiene el modelo matemático del actuador electro-mecánico

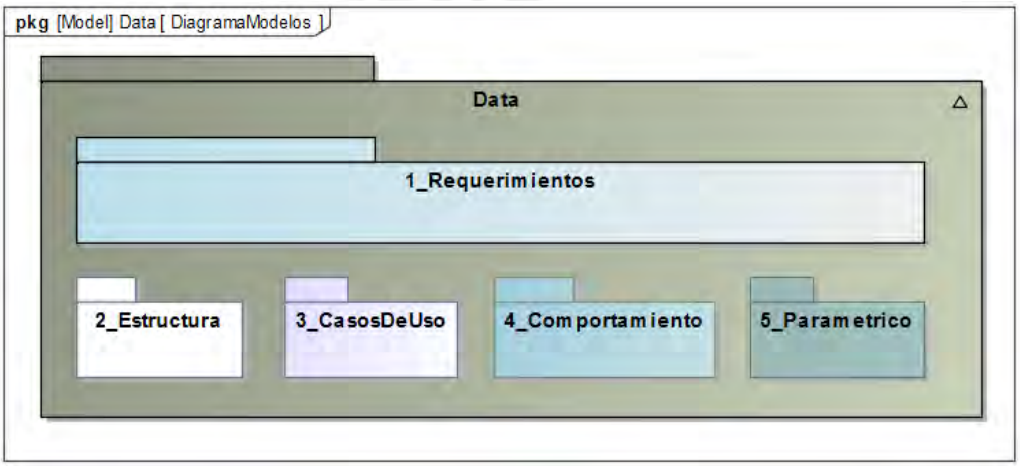


Figura 4.6 Diagrama de paquetes en *SysML;*
Fuente propia

El árbol de contenido en *MagicDraw* se muestra en la Figura 4.7. Se muestran 2 paquetes adicionales *Constraint Block Library* y *ParaMagic Profile* que se ha añadido por defecto debido a la instalación del *profile* del *plugin ParaMagic*.

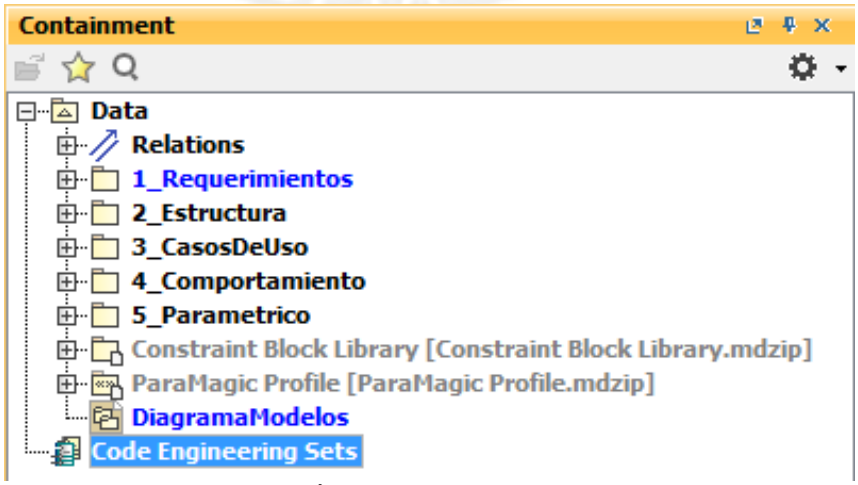


Figura 4.7 Árbol de contenido de MagicDraw;
Fuente Propia

4.2.2 Diagrama de requerimientos

Se presenta a continuación los requerimientos funcionales del sistema. Estos requerimientos comprenden 5 exigencias principales (mayor detalle en la sección 4.3) en la lista de requerimientos del actuador electro-mecánico:

- Actuar automáticamente acorde a las instrucciones del piloto
- Debe estar conforme a los estándares de seguridad
- El EMA debe poder acoplarse en el avión para mover un alerón
- Debe ser capaz de adaptarse en el medio ambiente que lo rodea
- Debe ser alimentado de manera suficiente en cuanto a voltaje y corriente

Adicionalmente se tienen requisitos derivados, que fueron añadidos para cumplir la primera exigencia. Por ejemplo, es necesario que el EMA deba mantener la incidencia acorde las instrucciones del piloto o también, que el sistema transforme la energía eléctrica en energía mecánica para poder mover la carga. En total se añadieron 9 requisitos derivados que desembocan en 2 subrequisitos necesarios para que se cumpla la primera exigencia. El diagrama de requerimientos en *SysML* se muestra en la Figura 4.8, los detalles de este diagrama se muestran en las Figura 4.9 - Figura 4.12 y en la Figura 4.13 se muestra la tabla generada a partir del diagrama.

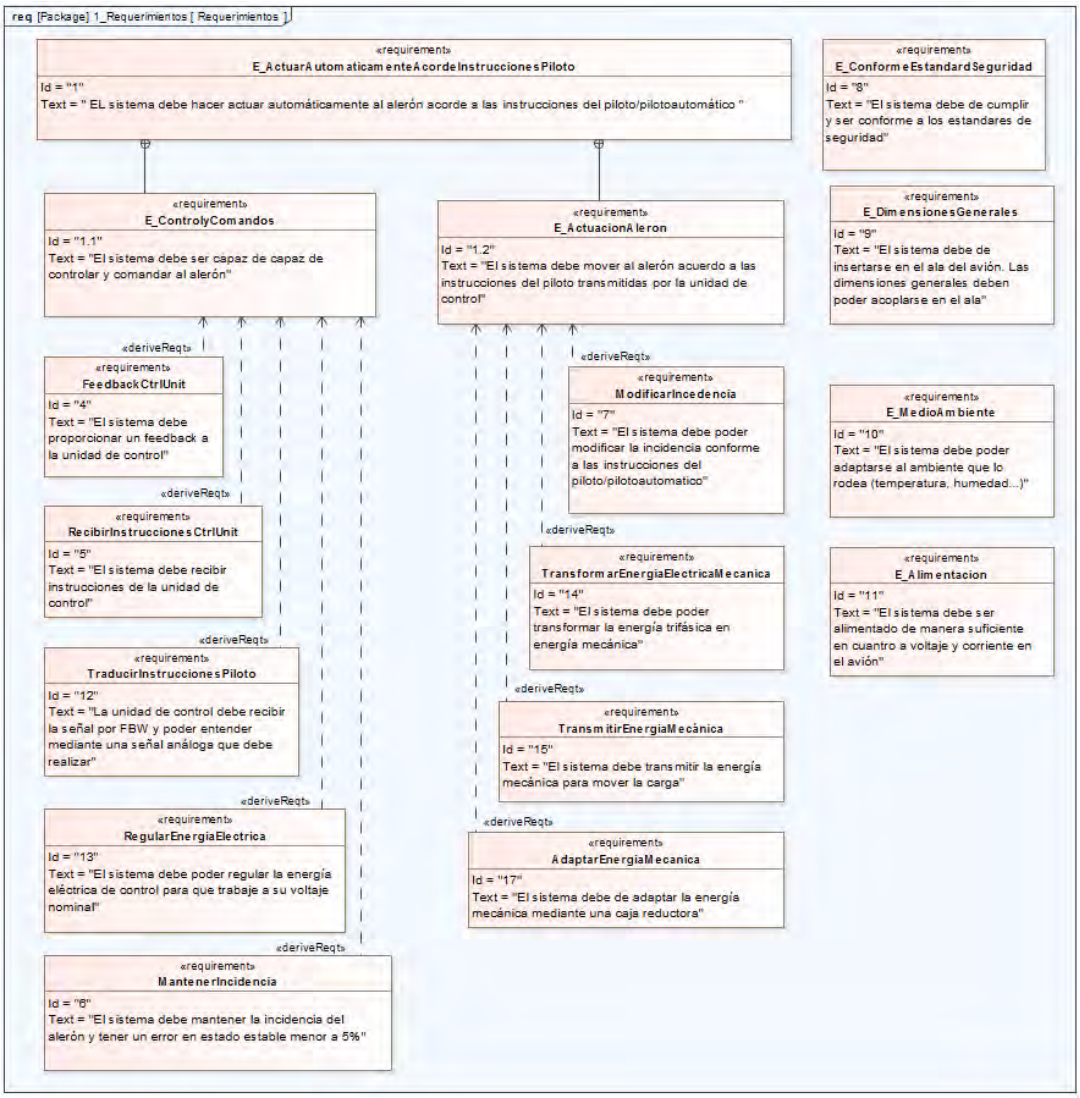


Figura 4.8 Diagrama de requerimientos en *SysML*;
Fuente propia

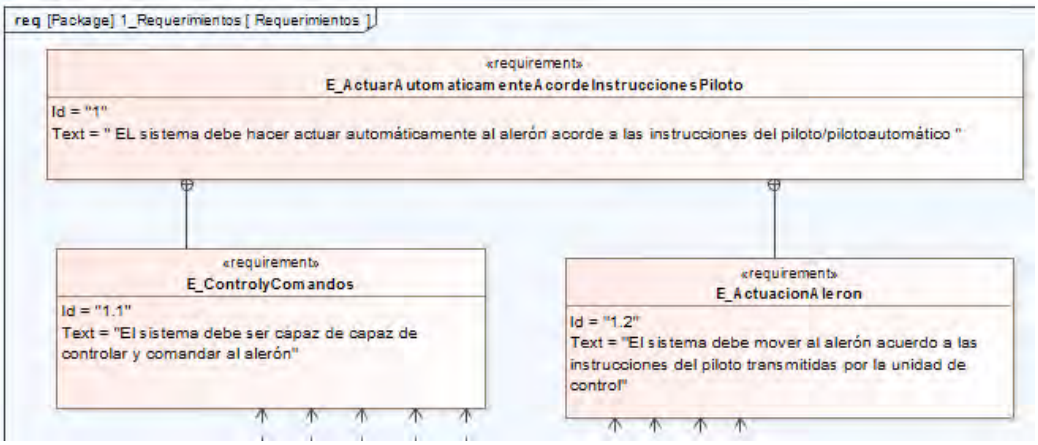


Figura 4.9 Requerimientos Principales;
Fuente propia

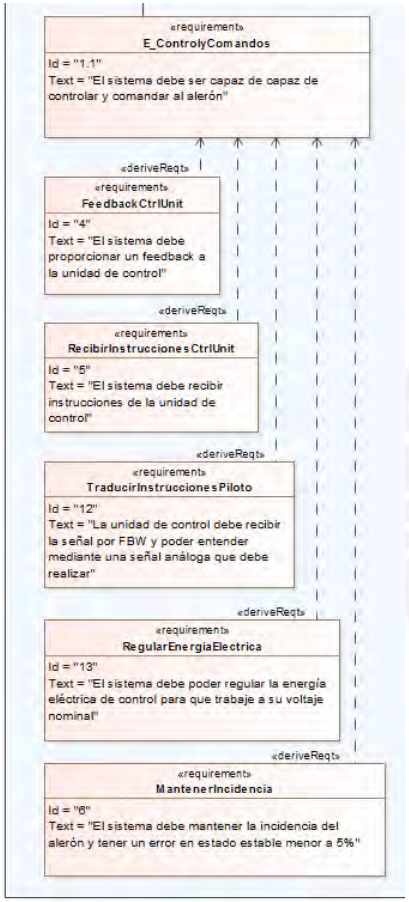


Figura 4.10 Requerimientos derivados de E_ControlyComandos; Fuente Propia

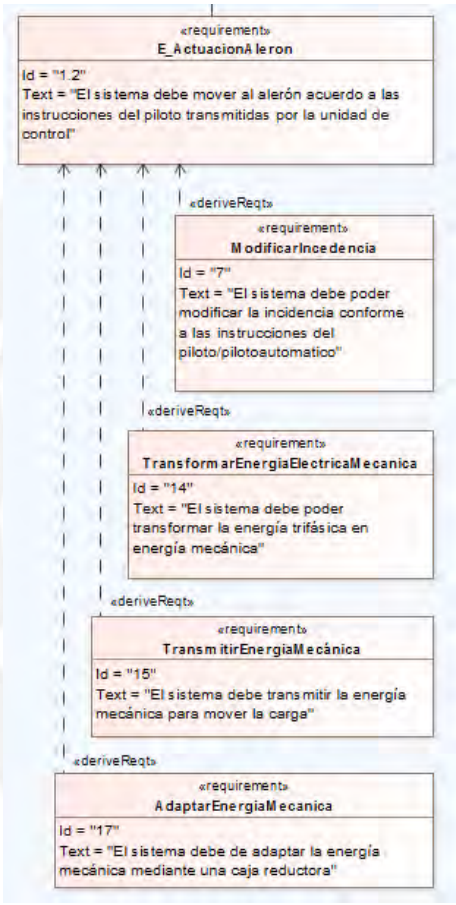


Figura 4.11 Requerimientos derivados de E_ActuacionAleron; Fuente Propia

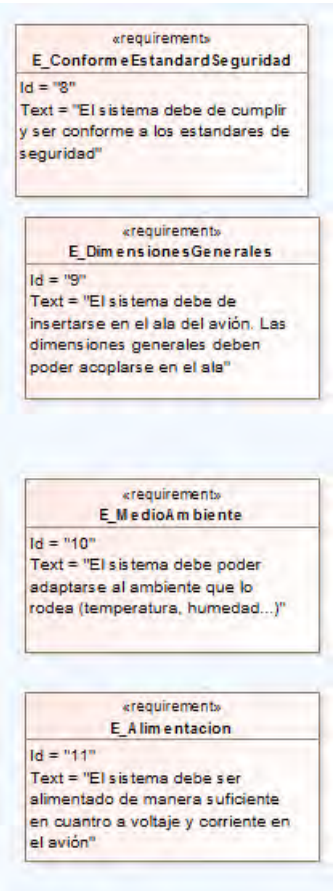


Figura 4.12 Otros requerimientos demandados; Fuente Propia

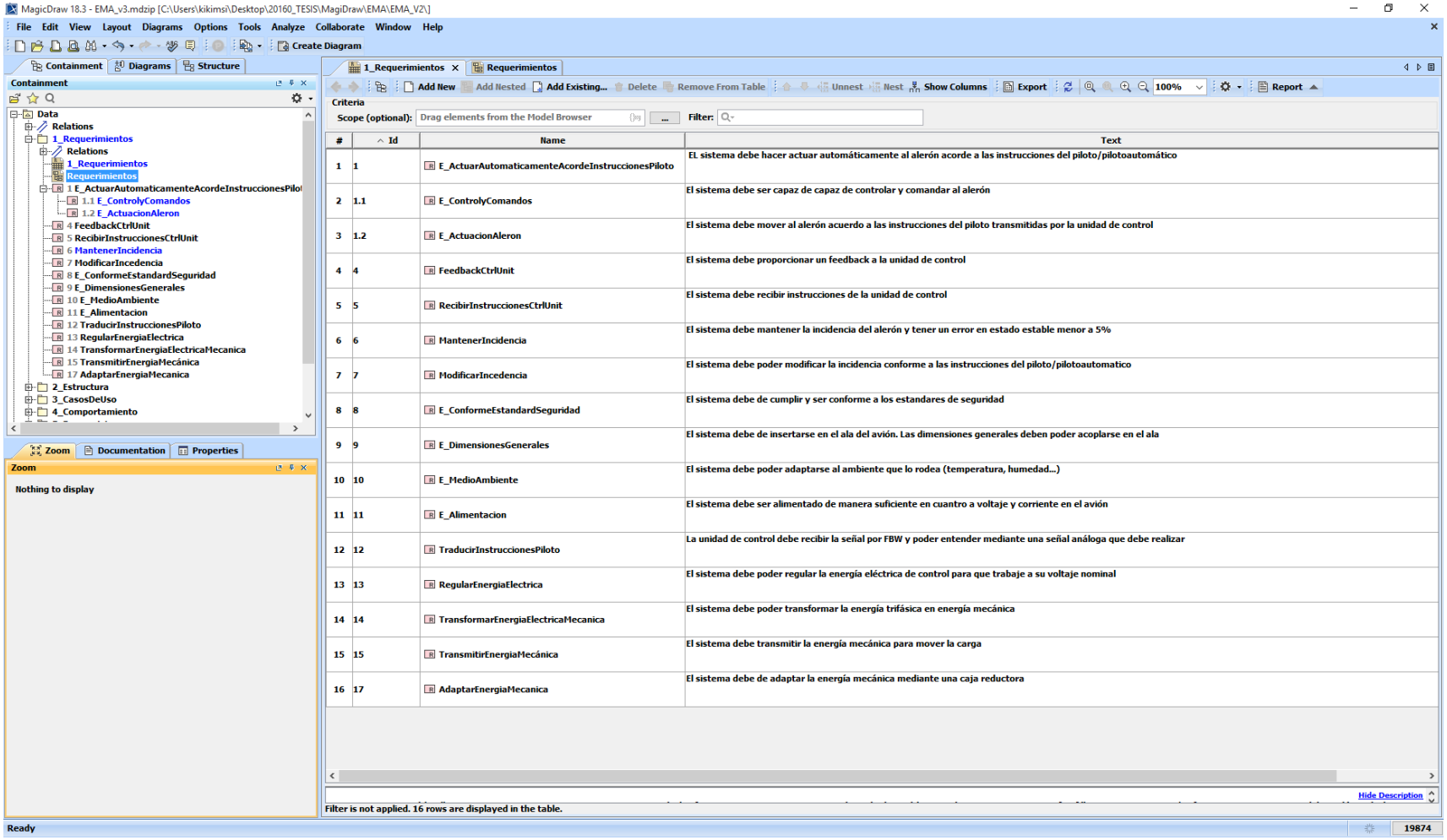


Figura 4.13 Tabla de Requerimientos en *SysML* dentro de *MagicDraw*;
Fuente propia

4.2.3 Diagrama de estructura

En la sección 4.1.2 se describieron 3 partes principales del actuador electromecánico. Se realiza un mayor detalle de estas 3 partes teniendo en consideración algunos modelos descritos por otros autores [25-27]. En general el EMA está compuesto por:

- Una unidad que contenga al microcontrolador que posee el sistema de control del EMA y el driver para transmitir la potencia a la transmisión, a esta unidad la llamaremos *uCEmbebidoDriverPotencia*.
- Un motor eléctrico con una caja reductora (motoreductor) y un sensor para medir la velocidad (encoder) que tomará el nombre de *MotoreductorConEncoder*.
- Una transmisión mecánica que permita mover la carga compuesta de un mecanismo de husillo de bolas, la cual llamaremos *TransmisionMecanica*.
- La superficie de control, que es la carga que se debe de mover, es decir, el alerón, a la cual se referirá como *Carga*.

La Figura 4.14 se muestra al bloque *SistemaEMA* que está relacionada con el *MotoreductorConEncoder*, el *uCEmbebidoDriverPotencia*, la *TransmisionMecanica* y la *Carga.*

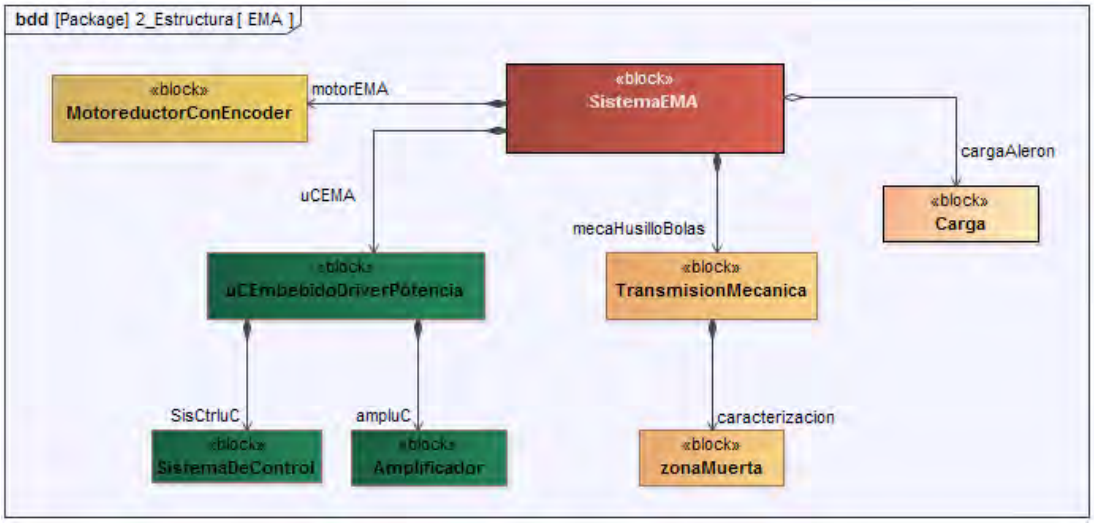


Figura 4.14 Diagrama de definición de bloques del EMA;
Fuente propia

4.2.4 Diagrama de funcionalidad

Primero se realiza el contexto general descrito en la sección 4.1.1 y 4.1.2 en un diagrama de definición de bloques que toma por nombre *Contexto* (ver Figura 4.15). Se añade a este diagrama 4 actores más que resaltan:

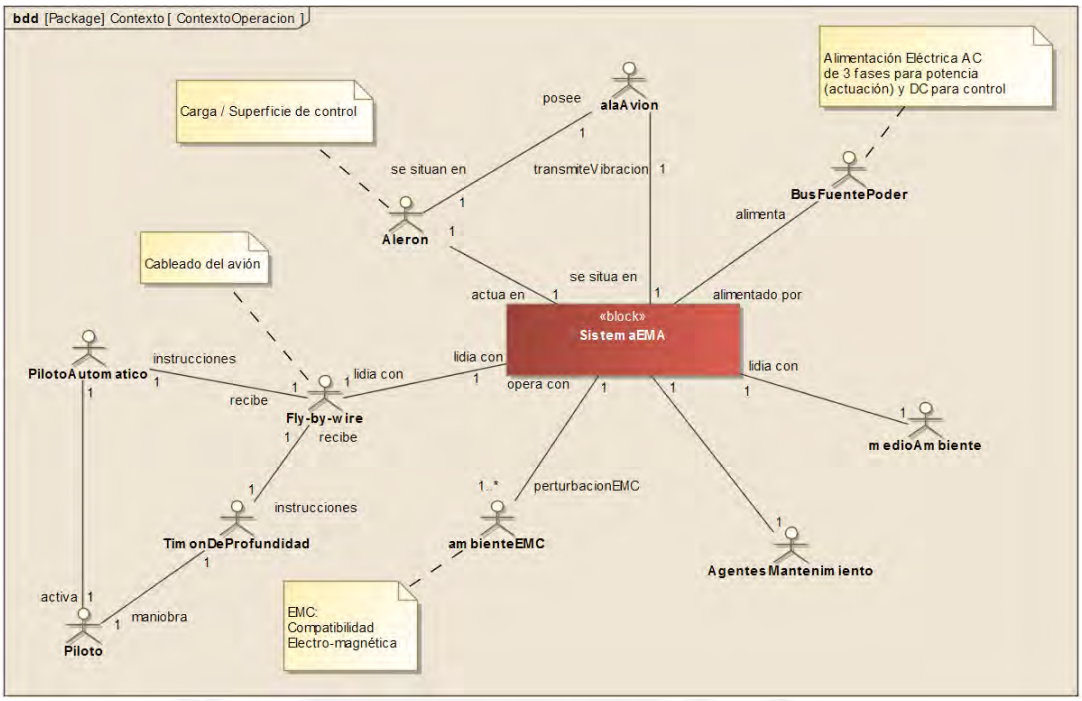


Figura 4.15 Contexto;
Fuente Propia

- La ubicación del alerón en el ala y que esta transmite vibración hacia el EMA
- Que el EMA debe poder adaptarse a las condiciones del medio ambiente
- Que el EMA recibe perturbaciones electro-magnéticas debido a otros componentes
- Que en algún momento se le hará mantenimiento al actuador

Luego se realiza el diagrama de casos de uso (ver Figura 4.16) que como se explicó en la sección 4.2.4 sirve para representar la funcionalidad en términos de como el sistema es usado por agentes externos (actores) para cumplir una serie de metas. Para el EMA, la tarea principal es controlar la incidencia del alerón lo que implica:

- Mover la carga y mantener la posición deseada, es decir, la actuación a
- Lidiar con la unidad de control del avión a la cual se le debe enviar información para que la visualice el piloto.

4.2.5 Diagrama de comportamiento

Para este caso de estudió se harán 3 diagramas de comportamiento: un diagrama de secuencias, un diagrama de máquinas de estado y un diagrama de actividades.

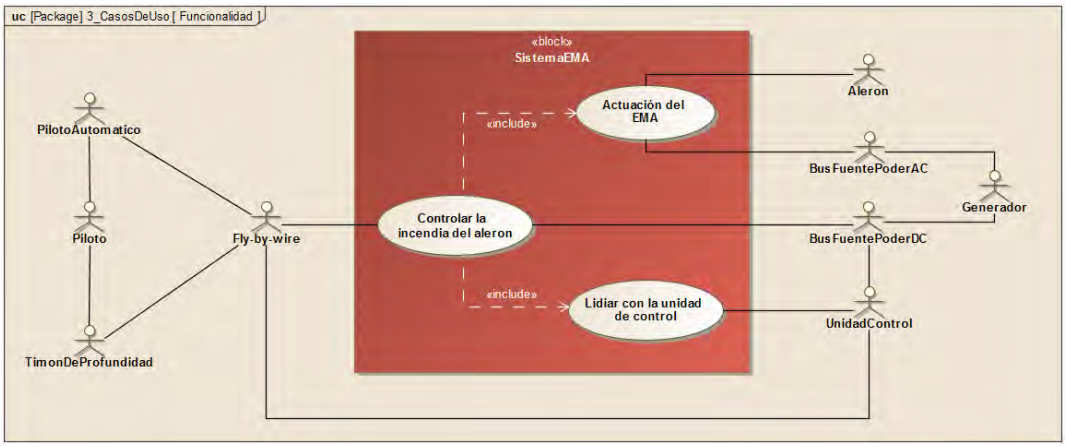


Figura 4.16 Diagrama de casos de uso del EMA;
Fuente: propia

El diagrama de secuencias se muestra en la Figura 4.17 en donde se describe que el primer paso para controlar la incidencia del alerón es encender/despertar el microcontrolador del estado *idle*, es decir, cuando no es usado por otro programa. Si se enciende el *idle* y la fuente de poder está bien conectada entonces las instrucciones piloto deben de traducirse, esto significa que el microcontrolador debe recibir las señales análogas y procesarlas. Luego, se debe adaptar la fuente de alimentación AC mediante el driver de potencia. Seguidamente, se debe de transformar esta energía en mecánica mediante el motor y adaptarla con la caja reductora para finalmente mover el alerón a través de la transmisión mecánica.

En la Figura 4.18 se muestra el diagrama de máquinas de estado. Se observa que el EMA tiene como primer estado el de apagado, luego si el avión se enciende, el EMA también, pasando por un estado previo de chequeo que todo esté funcionando correctamente, si es que no es así el EMA debe suspenderse. Por otro lado, si todo está bien pasa a un estado de *idle* hasta que reciba las instrucciones del piloto. Cuando recibe los comandos del piloto el actuador electro-mecánico debe empezar a mover el alerón hasta que alcance la posición deseado. Luego, debe mantener su posición y permanecer en el estado de *idle*. Si el avión se apaga entonces el EMA, pasa de un estado de reposo al de apagado con lo que concluye.

El diagrama de actividades se muestra en la Figura 4.19 en dónde hay 2 actividades principales **controlYcomandos** y **actuarEMA**.

El primero se observa a detalle en la Figura 4.20 en donde las instrucciones piloto deben de traducirse. Por otro lado, debe hacerse una lectura del ángulo de incidencia

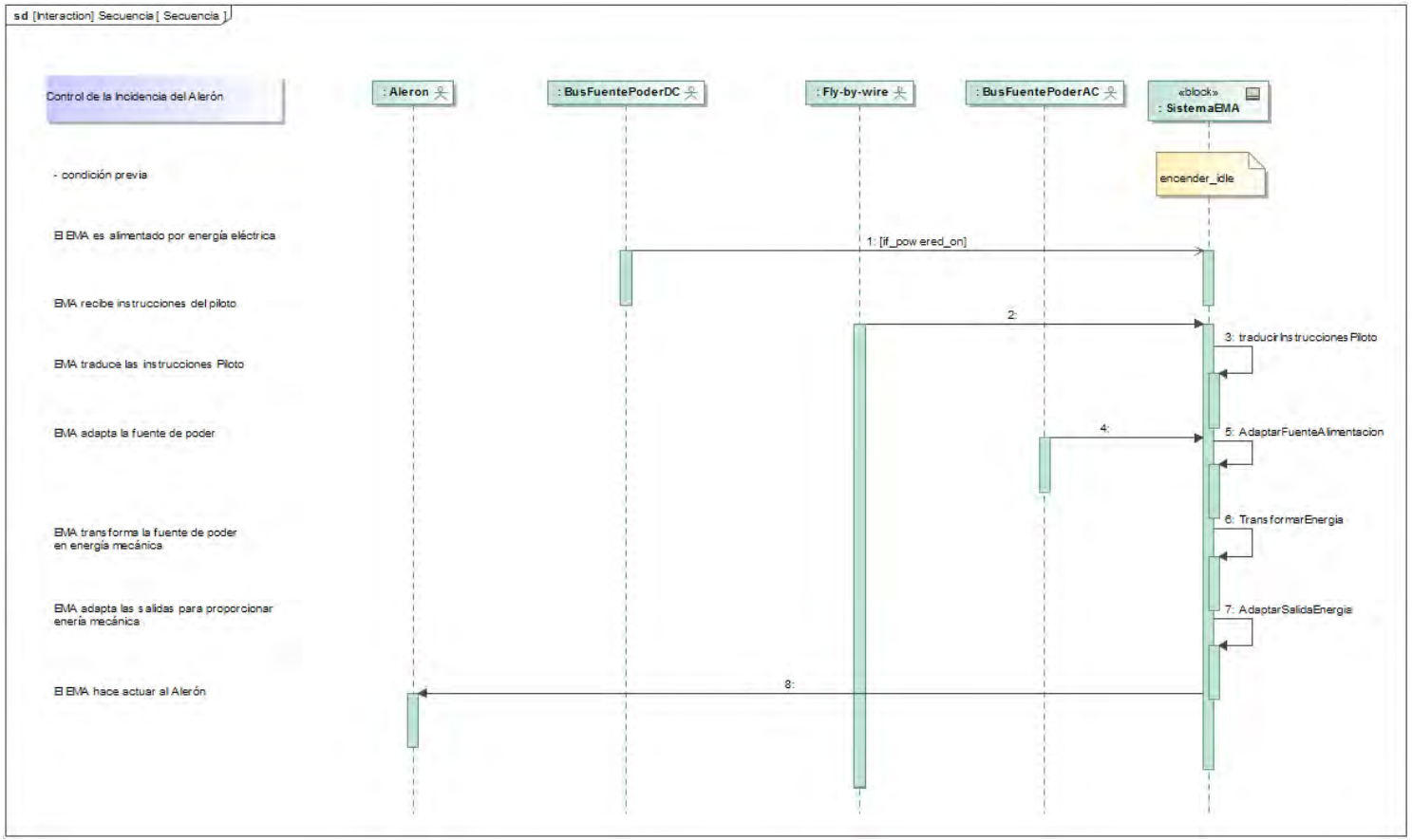


Figura 4.17 Diagrama de secuencias;
Fuente propia

del actuador electro-mecánico y a partir de eso regular la potencia del motor mediante el driver. Además, debe enviarse una señal a la unidad de control para que el piloto sepa si llegó a la posición deseada.

La segunda actividad concierne a la actuación del sistema es decir que, a partir de la potencia transmitida, la energía eléctrica debe de transformarse en mecánica. Luego se adapta la energía mediante la caja reductora del motor para generar más torque y mover la superficie de control a través de la transmisión. Además, debido a que el actuador cuenta con un sensor puede medirse el ángulo de incidencia. Esta actividad se detalla en la Figura 4.21.

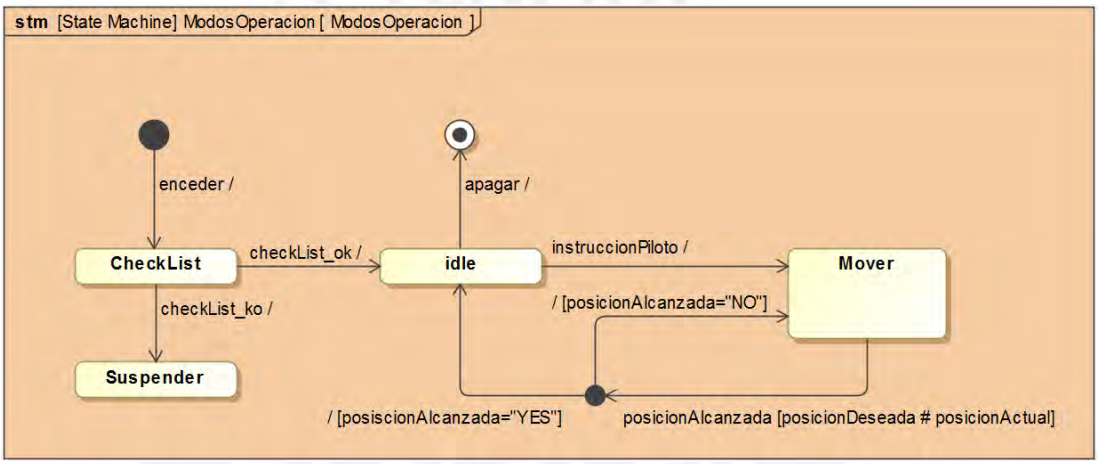


Figura 4.18 Diagrama de estados de máquina;
Fuente Propia

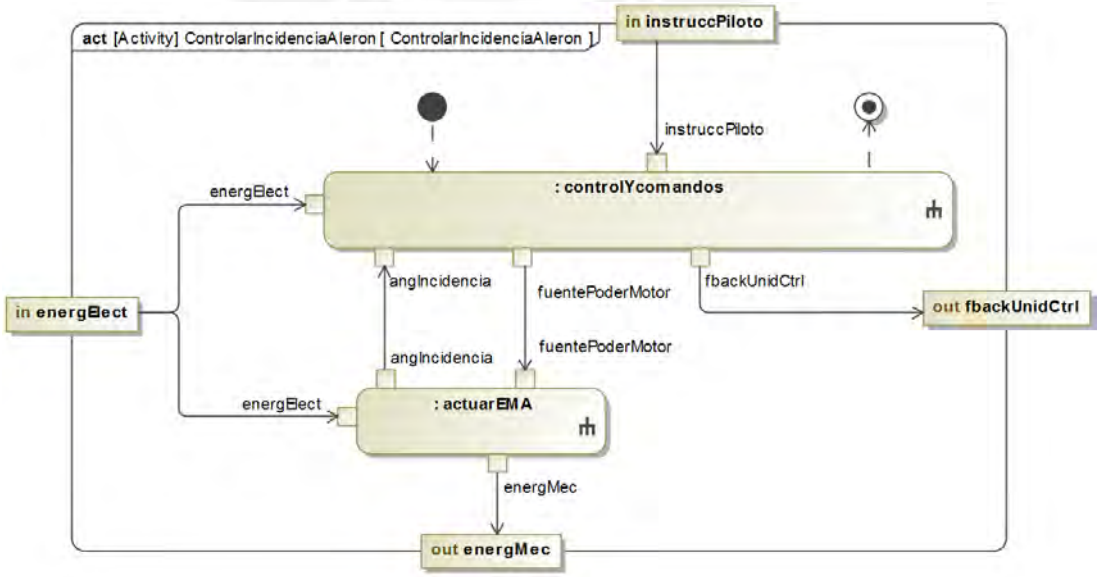


Figura 4.19 Diagrama de actividades del EMA;
Fuente propia

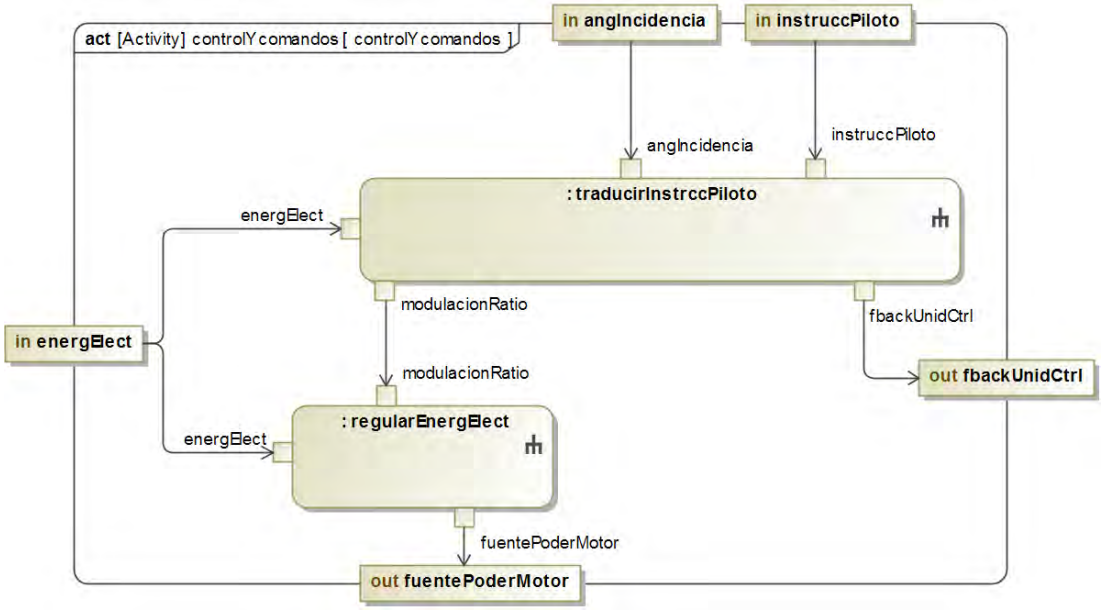


Figura 4.20 Diagrama de actividades de **controlYcomandos**;
Fuente propia

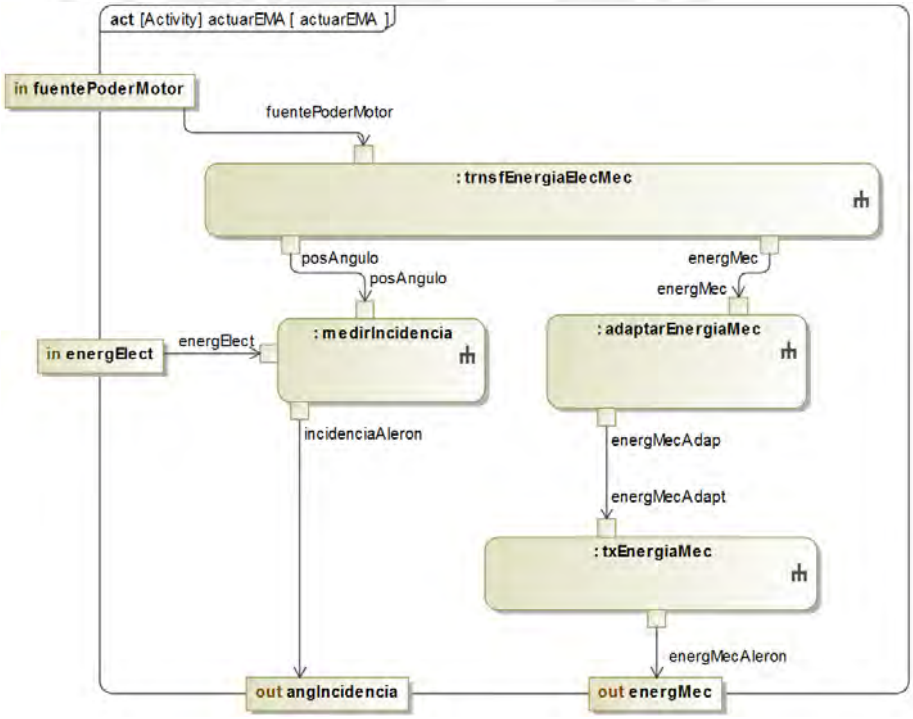


Figura 4.21 Diagrama de actividades **actuarEMA**;
Fuente propia

4.2.6 Diagrama paramétrico

El diagrama paramétrico se realizó tomando como referencia el modelo matemático de un EMA descrito por *Habibi* [24]. El sistema del actuador electro-mecánico que se mostró en Figura 4.4 puede verse como:

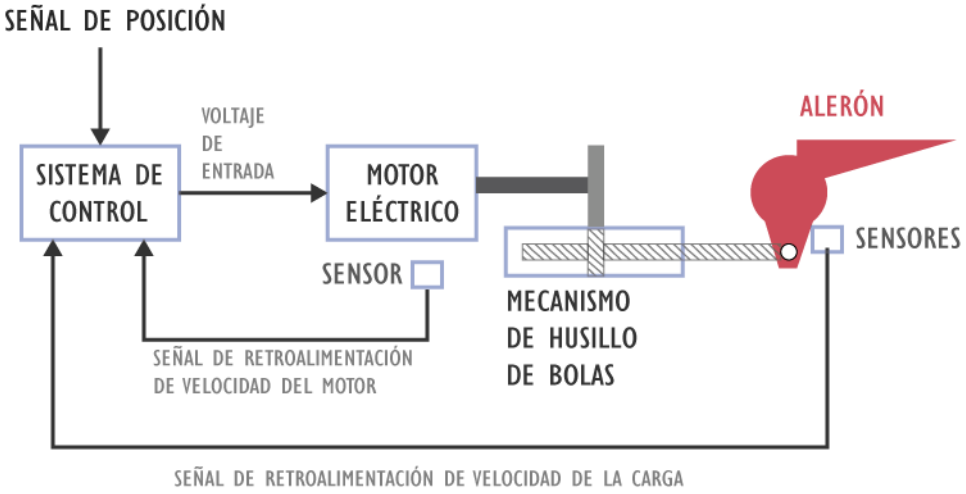


Figura 4.22 EMA,
Fuente adaptado de [24]

Además de los sistemas que componen al actuador se tienen los datos de la Tabla 4-1.

El diagrama paramétrico que relaciona los valores de la Tabla 4-1 se muestra en la Figura 4.23.

4.2.7 Integración de *SysML* y *Simulink*

La ejecución del modelo paramétrico puede efectuarse gracias al *plug in ParaMagic*. En primer lugar, el *profile* debe ser añadido al proyecto que se esté desarrollando en *MagicDraw*. Luego debe crearse una instancia referida al modelo paramétrico. En este caso de estudio, todos los valores de la Tabla 4-1 son importados a *MagicDraw* mediante *Excel*. En la Figura 4.24 se observa que los datos han sido importados y el error en estado estacionario aún no ha sido calculado.

Después se deben asignar las causalidades de los valores, para este caso el objetivo es alcanzar la posición deseada por el piloto (0.5 pulgadas) y retornar el error en estado estacionario. En la Figura 4.25 se observa la ventana de Paramagic que permite asignar dichas causalidades, se debe notar que el error en estado en estacionario aún no se calcula. Cunado se ejecuta el programa, haciendo clic en *solve*. Se mostrará la ventana de progreso (ver Figura 4.26), se abrirá el programa de *Matlab* y mediante un script (para mayor detalle ver Anexo B) correrá el programa en *Simulink* (ver Figura 4.27 y Figura 4.28). El valor del error en estado estacionario (valor en porcentaje) se almacenará en un documento tipo texto llamado output.txt. Finalmente, se actualizarán los resultados al modelo en *ParaMagic*, lo cual se muestra con éxito en la Figura 4.29

dando un error de 4.88 % y luego este resultado irá al modelo en *SysML*, es decir la instancia, como se muestra en la Figura 4.30.

Tabla 4-1 Nomenclatura del modelo matemático; Fuente propia

		Descripción	Símbolo	Valor	Unid.
Transmisión Mecánica	caracterización	Ancho	α	0.005	in
		Pendiente	m	1	
		Ganancia por zona muerta	$Gb1$	1	
	Constante resorte		kb	114350	lb/in
	Factor de amortiguamiento		cb	0	lb s/in
	Máx. desplazamiento		$xlim$	6	in
	Relación de Transmisión		N	163.4	rad/in
Motor	Velocidad máxima		$wmax$	8000	rpm
	Ganancia *feedback* velocidad		Kv	0.2	
Unidad uC y Driver	Sistema de Control	Control proporcional	c	260	
		Función de transferencia del lazo interno	$G2$	1	
		Ganancia efectiva después de compensación	$Gbcomp$	1	
		Ganancia de lazo interno	$Kinner$	40	
	Amplif.	Voltaje de saturación	$Vsat$	127	V
Carga	Peso		M	0.2588	lbm
	Fricción Viscosa		Bl	5	lb s/in
	Constante de resorte		Kl	1000	lb/in
Unidad Control	Posición deseada		pd ó xd	0.5	in

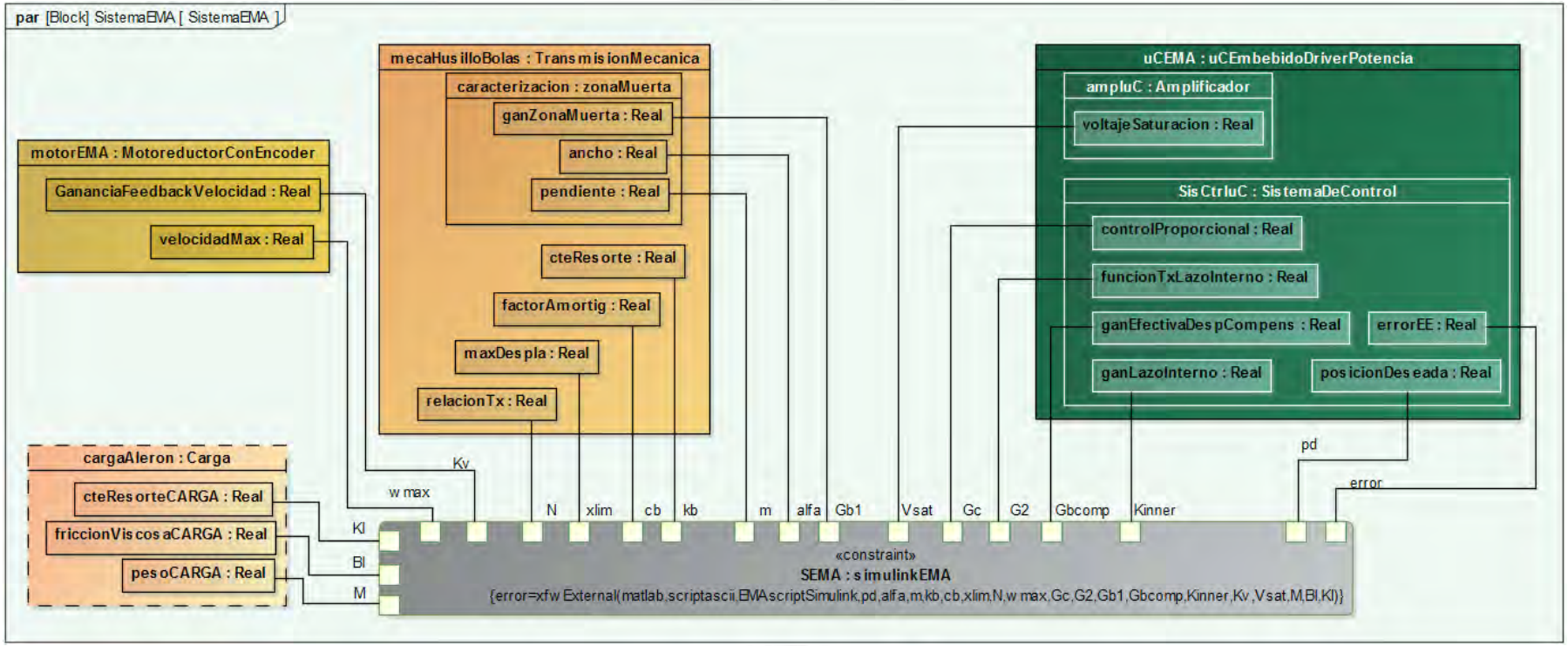


Figura 4.23 Diagrama Paramétrico del EMA;
Fuente propi

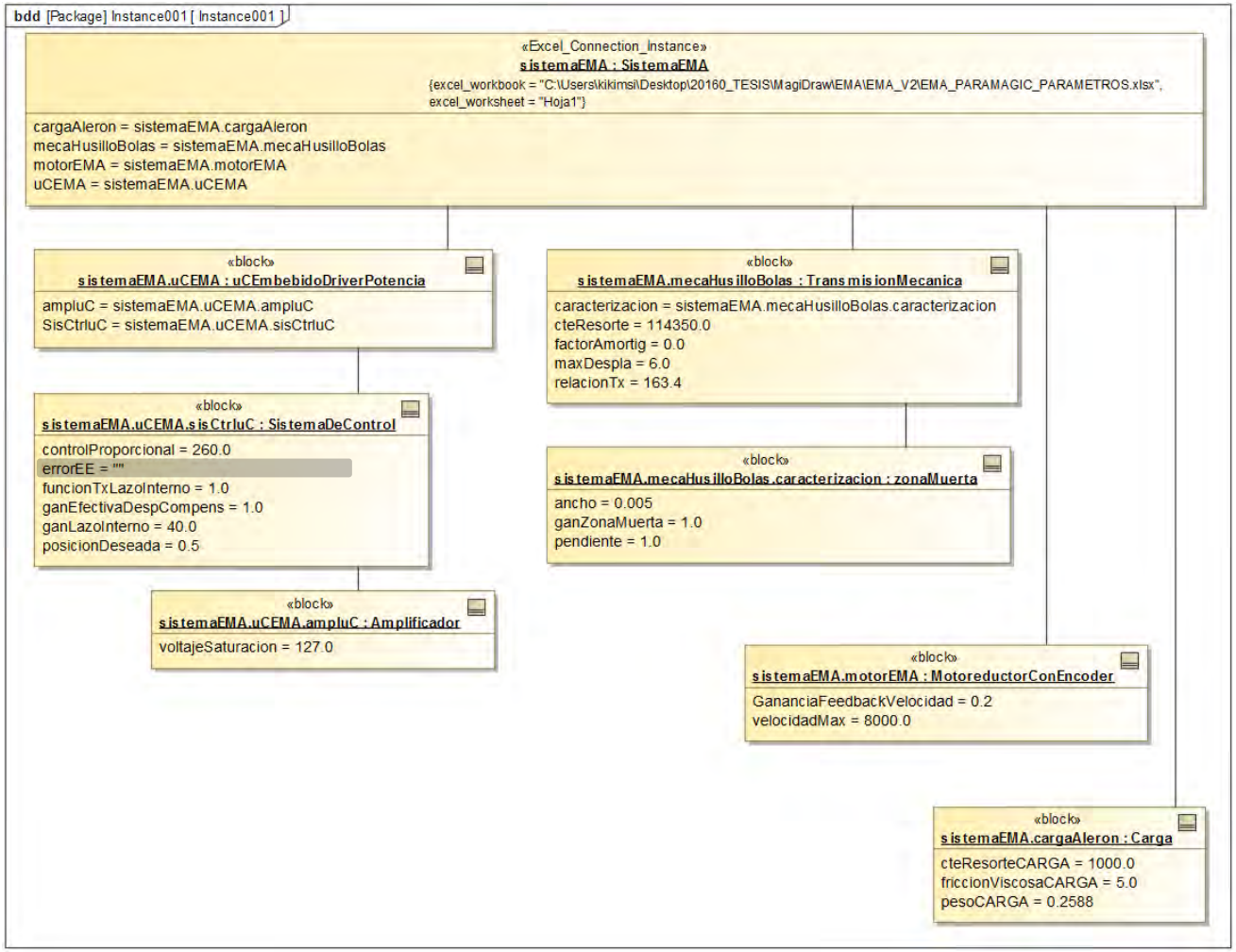


Figura 4.24 Instancia del modelo paramétrico; Fuente propia

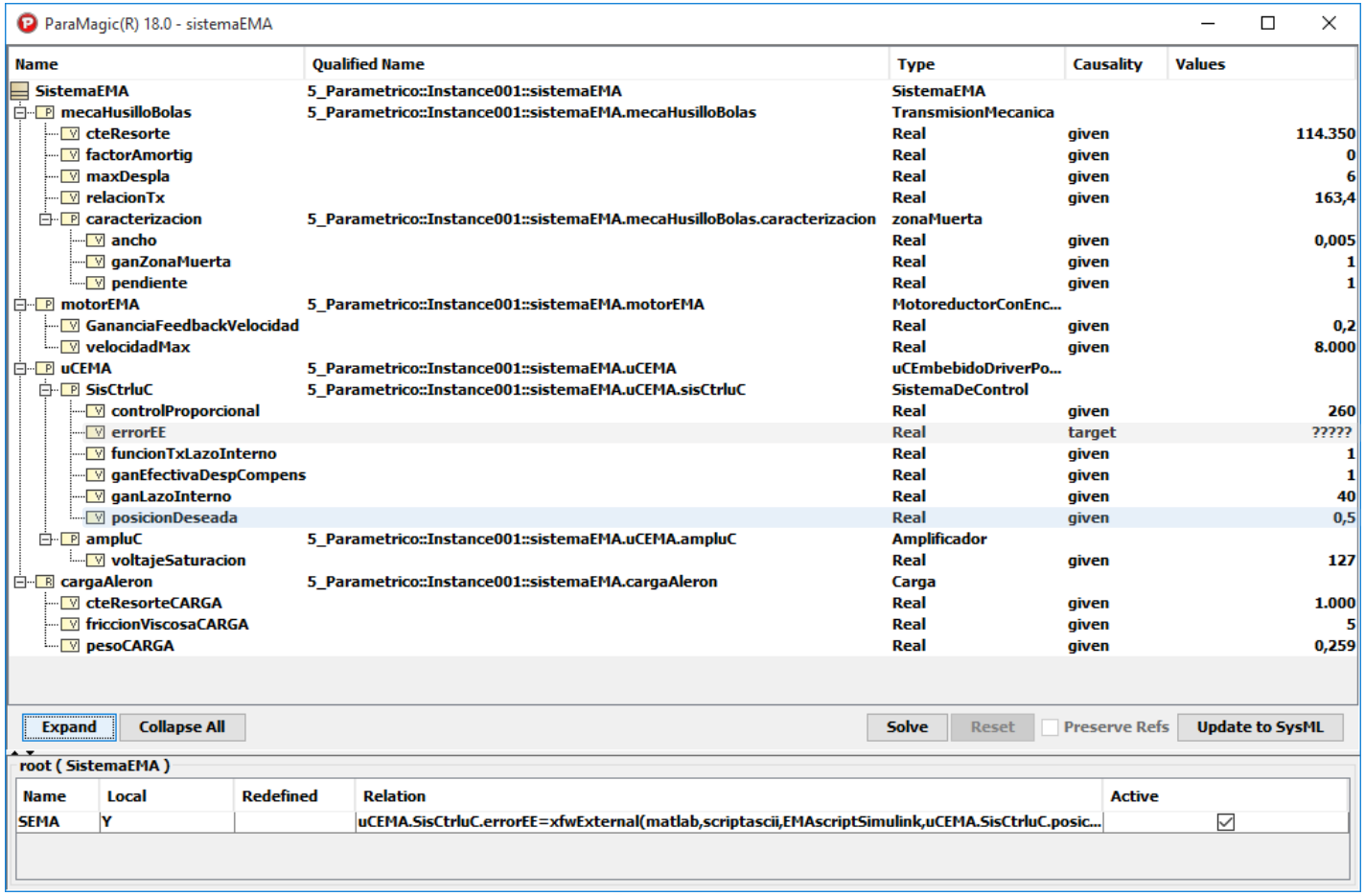


Figura 4.25 Asignación de causalidades;
Fuente propia

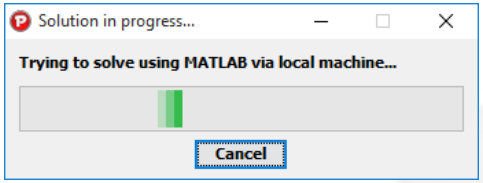


Figura 4.26 Ventana de solución en progreso de ParaMagic;
Fuente propia

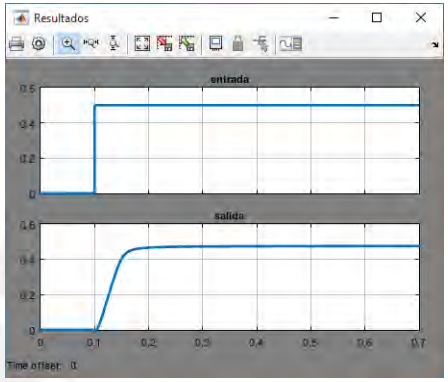


Figura 4.27 Resultado de Matlab/Simulink;
Fuente propia

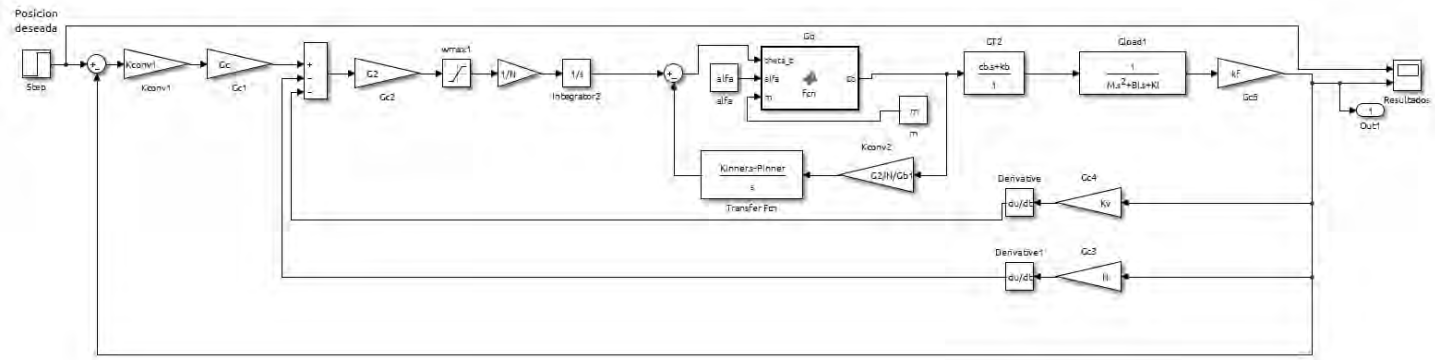


Figura 4.28 Modelo del EMA en *Simulink*;
Fuente propia

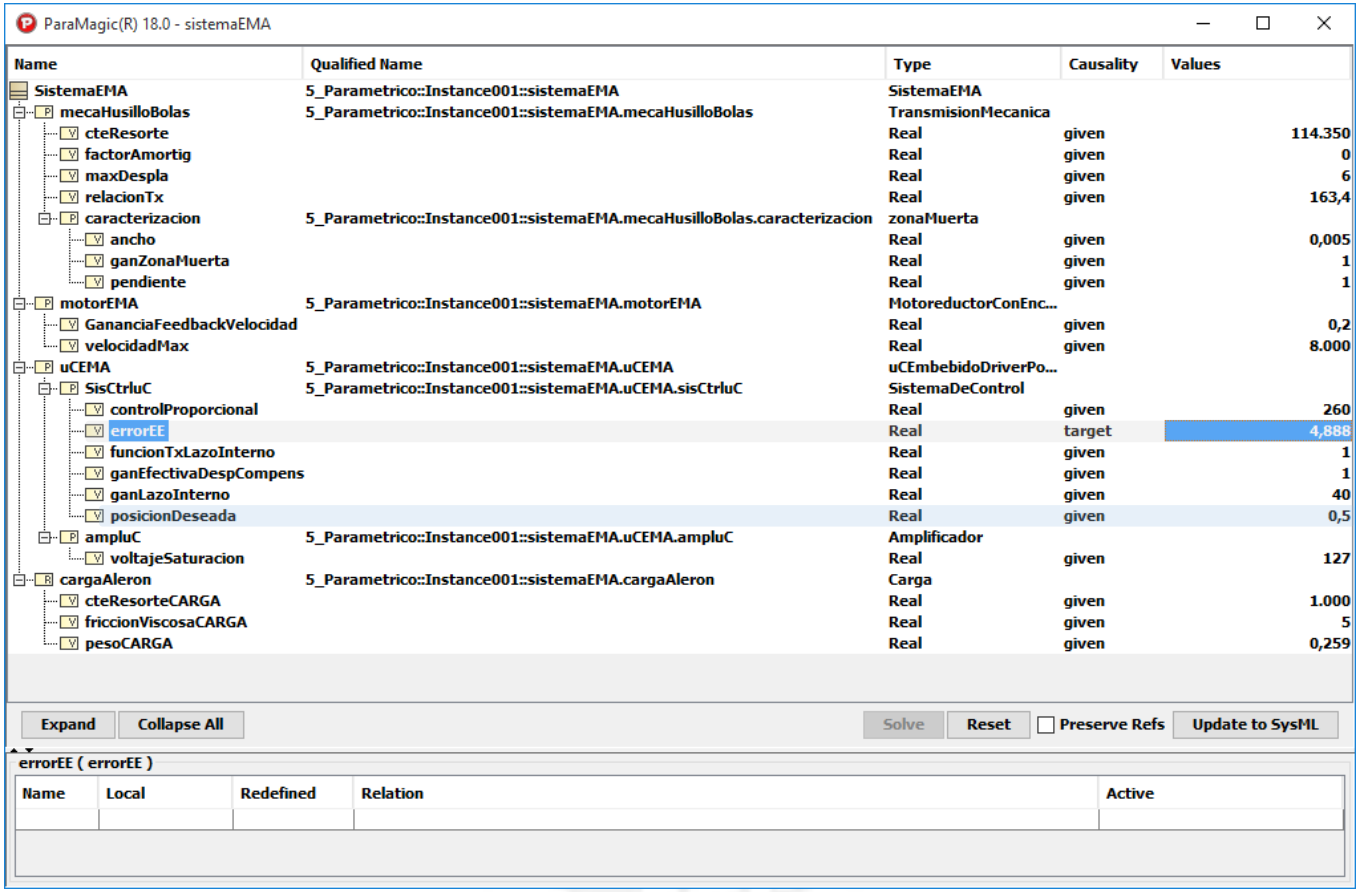


Figura 4.29 Actualización de datos en *ParaMagic*;
Fuente propia

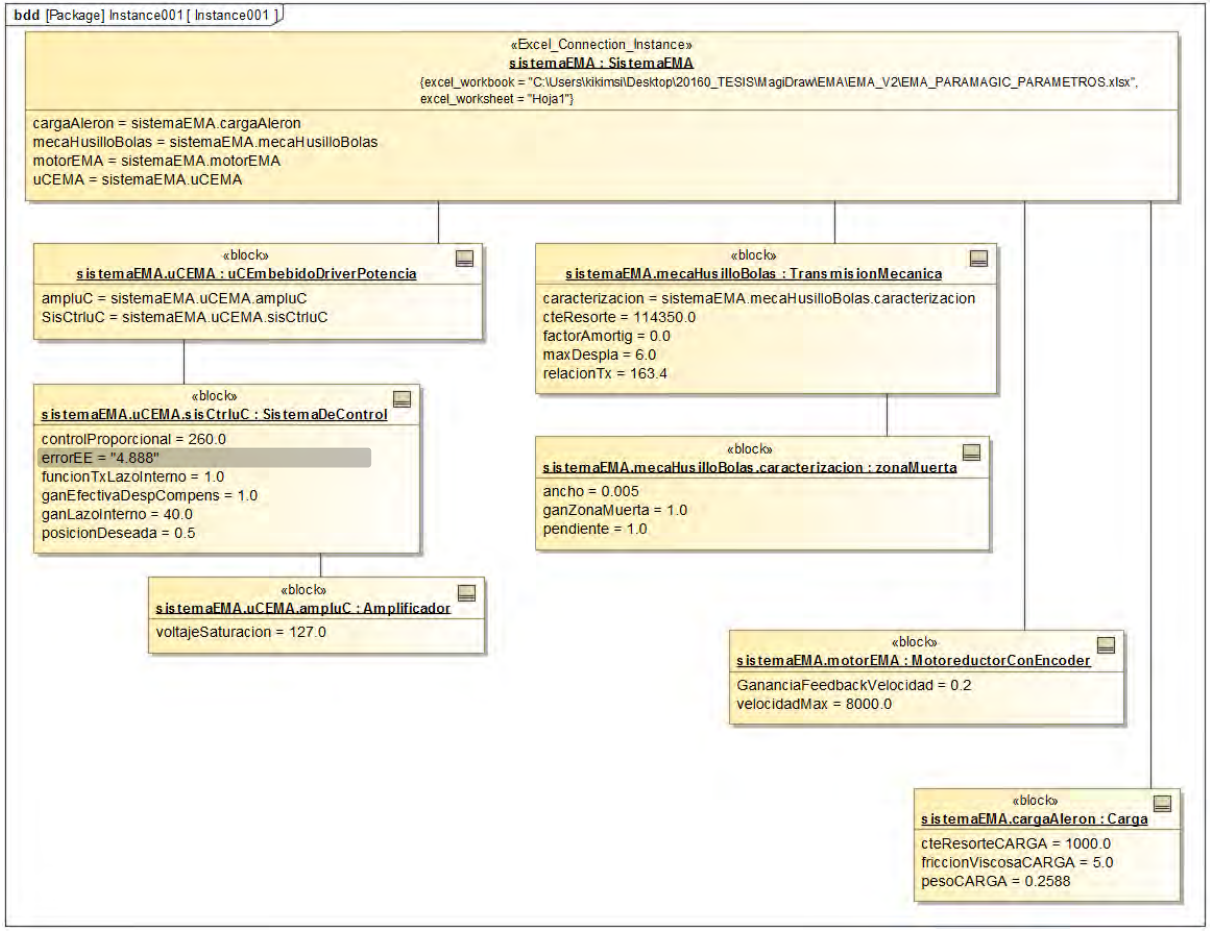


Figura 4.30 Actualización de datos de la instancia, **errorEE** = 4.88%; Fuente propia

Tabla 4-2 Visualización de EMA_REQUERIMIENTOS.xlsx en *Excel*; Fuente propia

A	B	C	D
1_Requerimientos			
#	Id	Name	Text
1	1	E_ActuarAutomaticamenteAcordeInstruccionesPiloto	EL sistema debe hacer actuar automáticamente al alerón acorde a las instrucciones del piloto/pilotoautomático
2	1.1	E_ControlyComandos	El sistema debe ser capaz de capaz de controlar y comandar al alerón
3	1.2	E_ActuacionAleron	El sistema debe mover al alerón acuerdo a las instrucciones del piloto transmitidas por la unidad de control
4	4	FeedbackCtrlUnit	El sistema debe proporcionar un feedback a la unidad de control
5	5	RecibirInstruccionesCtrlUnit	El sistema debe recibir instrucciones de la unidad de control
6	6	MantenerIncidencia	El sistema debe mantener la incidencia del alerón y tener un error en estado estable menor a 5%
7	7	ModificarIncedencia	El sistema debe poder modificar la incidencia conforme a las instrucciones del piloto/pilotoautomatico
8	8	E_ConformeEstandardSeguridad	El sistema debe de cumplir y ser conforme a los estándares de seguridad

Tabla 4-2 (Continuación)

A	B	C	D
1_Requerimientos			
#	Id	Name	Text
9	9	E_DimensionesGenerales	El sistema debe de insertarse en el ala del avión. Las dimensiones generales deben poder acoplarse en el ala
10	10	E_MedioAmbiente	El sistema debe poder adaptarse al ambiente que lo rodea (temperatura, humedad...)
11	11	E_Alimentacion	El sistema debe ser alimentado de manera suficiente en cuanto a voltaje y corriente en el avión
12	12	TraducirInstruccionesPiloto	La unidad de control debe recibir la señal por FBW y poder entender mediante una señal análoga que debe realizar
13	13	RegularEnergiaElectrica	El sistema debe poder regular la energía eléctrica de control para que trabaje a su voltaje nominal
14	14	TransformarEnergiaElectricaMecanica	El sistema debe poder transformar la energía trifásica en energía mecánica
15	15	TransmitirEnergiaMecánica	El sistema debe transmitir la energía mecánica para mover la carga
16	17	AdaptarEnergiaMecanica	El sistema debe de adaptar la energía mecánica mediante una caja reductora

El requerimiento con *id* 4 es satisfecho gracias al microcontrolador, que mediante una actividad envía una señal de *feedback* a la unidad de control. El requerimiento con *id* 5 es satisfecho gracias a las conexiones hechas por *fly-by-wire*. El requerimiento con *id* 6 debe ser verificado por el diagrama paramétrico del sistema de control y mantiene su incidencia gracias al driver de potencia. El requerimiento con *id* 12 es satisfecho por el microcontrolador que tiene puertos análogos. El requerimiento con *id* 13 es satisfecho debido al driver de potencia que tiene el EMA. Con el cumplimiento de los requerimientos mencionados el requerimiento 1.1 es satisfecho (ver Figura 4.31).

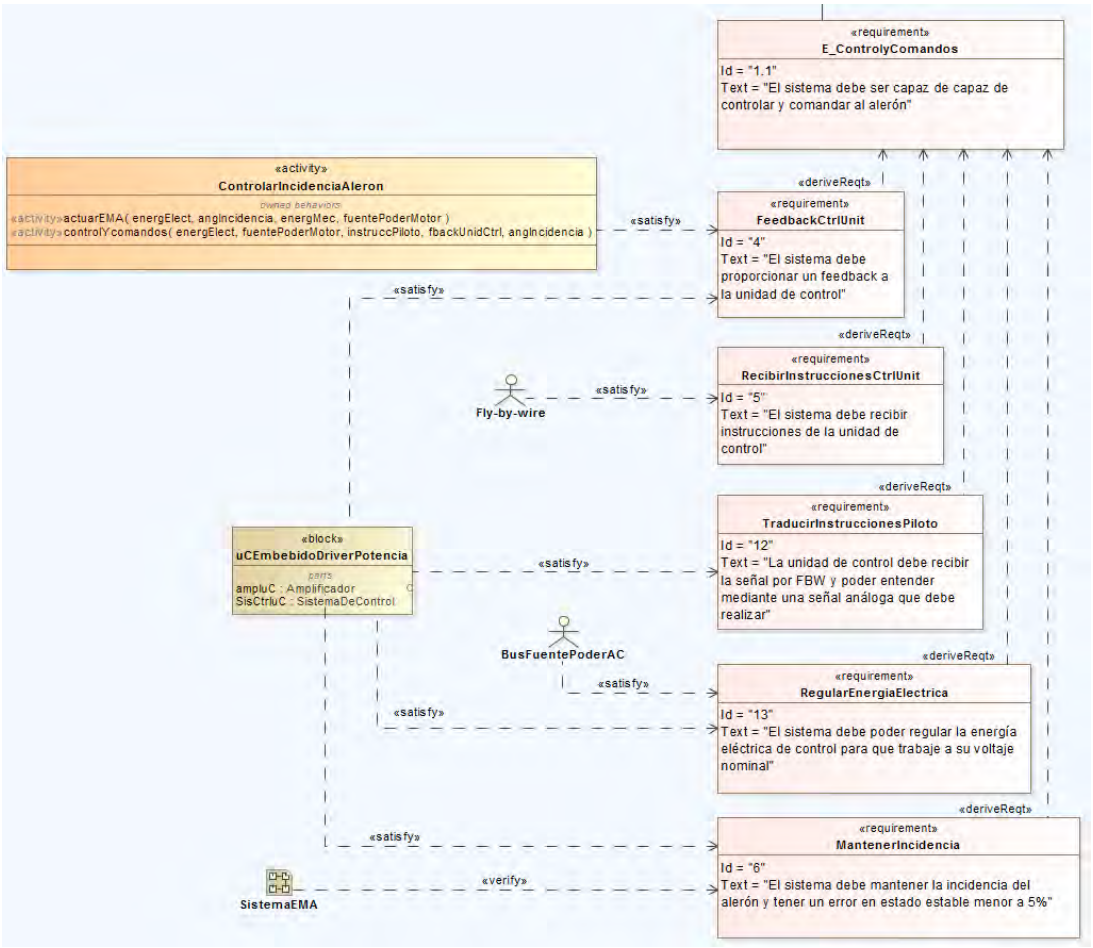


Figura 4.31 Trazabilidad de los requerimientos con *id* 1.1;
Fuente propia

El requerimiento con *id* 7 es satisfecho debido a las conexiones por *fly-by-wire* y el sistema de control. El requerimiento con *id 14* se satisface porque el EMA cuenta con un *driver* de potencia y un motor. El requerimiento con *id* 15 se cumple debido a la transmisión mecánica que mueve el husillo. El requerimiento con *id* 17 se satisface gracias a la caja reductora del motor. Con el cumplimiento de los requerimientos mencionados el requerimiento 1.2 es satisfecho (Figura 4.32).

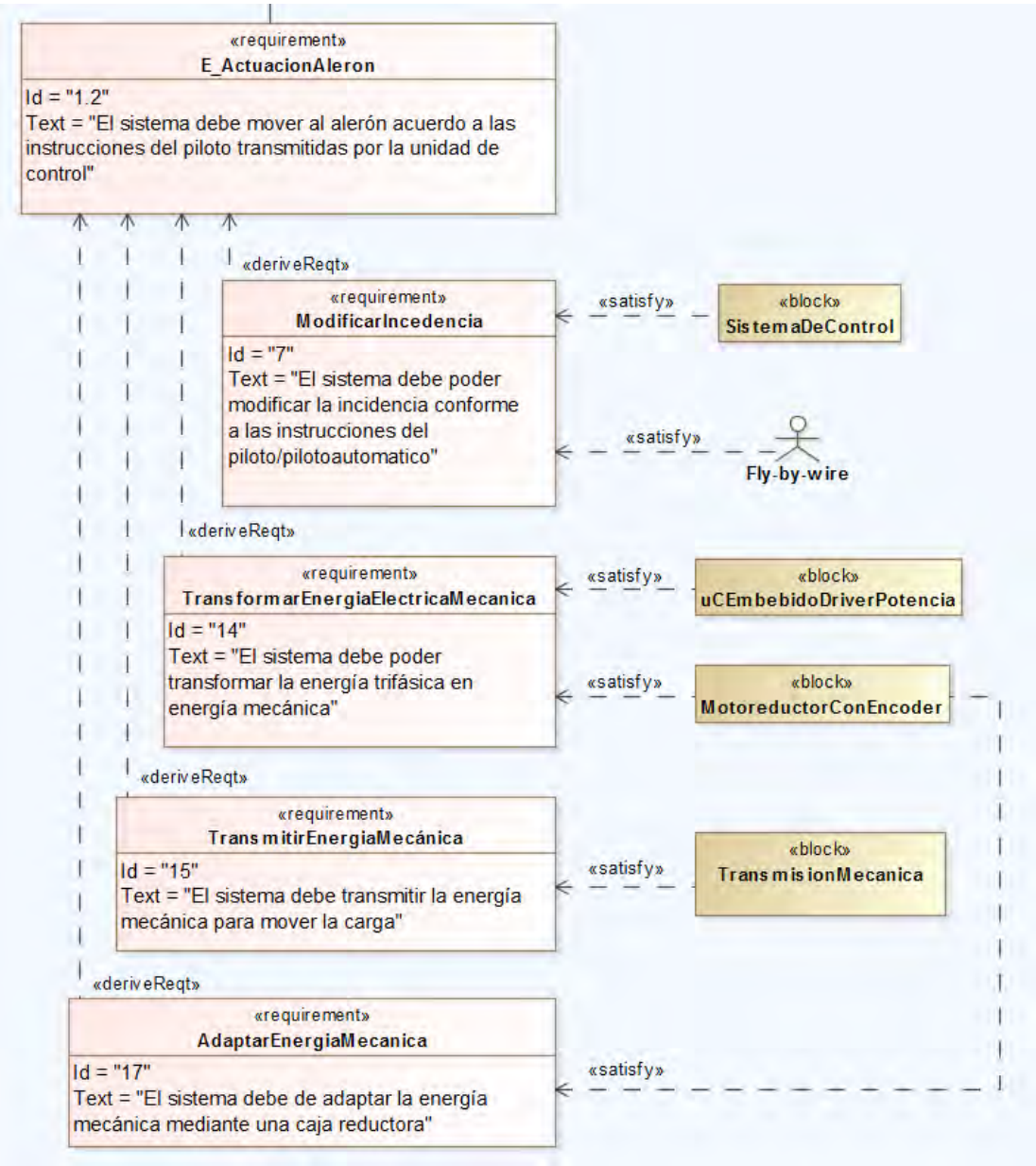


Figura 4.32 Trazabilidad de los requerimientos con *id* 1.2;
Fuente propia

El requerimiento con *id* 11 se satisface por el generador del avión. Sin embargo, no se puede estar seguro si los requerimientos con *id* 8, 9 y 10 son satisfechos debido a la falta de información (ver Figura 4.33). No obstante, para este caso de estudio nos centraremos en la trazabilidad del requerimiento con *id* 1. El siguiente paso es validar su trazabilidad.

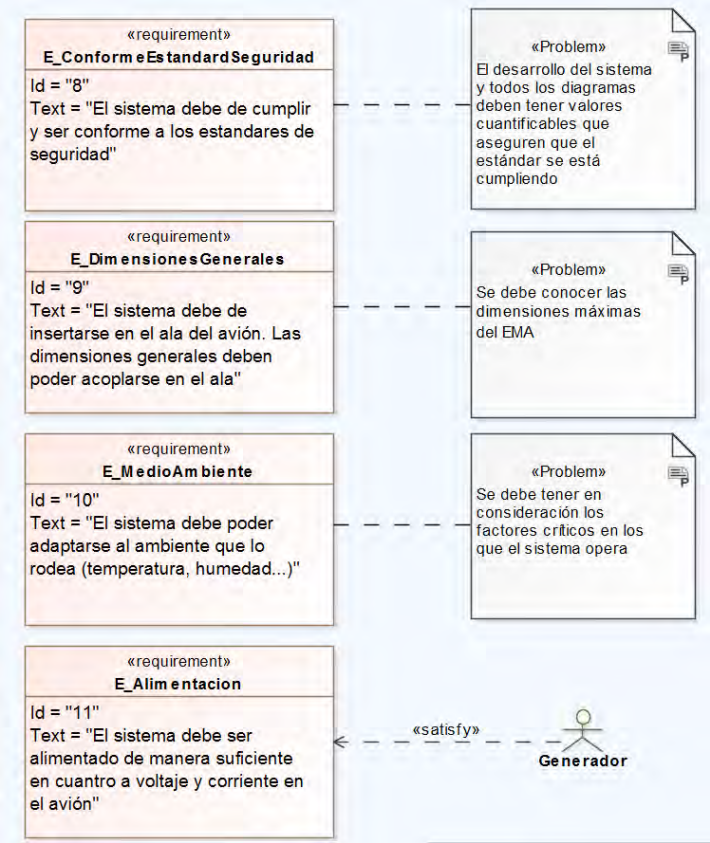


Figura 4.33 Trazabilidad de requerimientos con *id* 8, 9,10 y 11; Fuente propia

4.3 Verificación de requerimientos

La validación se hace agregando condiciones a la hoja de cálculo mostrada en la Tabla 4-2. Además, se debe resaltar que el único requerimiento con un valor cuantificable es el requerimiento con *id* 5, el cual dice que el sistema de control debe tener un error en estado estable menor a 5%.

Antes de realizar la validación, debe de configurarse el programa para que el valor del error se escriba en la hoja de cálculo. Para el ejemplo el error se escribe en la casilla 'E9'. Esto se muestra en la Figura 4.34.

La hoja de cálculo modificada se muestra en la Tabla 4-3. Se observa que se ha añadido la columna de filas (sombreada en amarillo) y la fila de columnas (sombreada en azul). Todos los requerimientos que se están cumpliendo están sombreados en verde y los que no están en rojo.

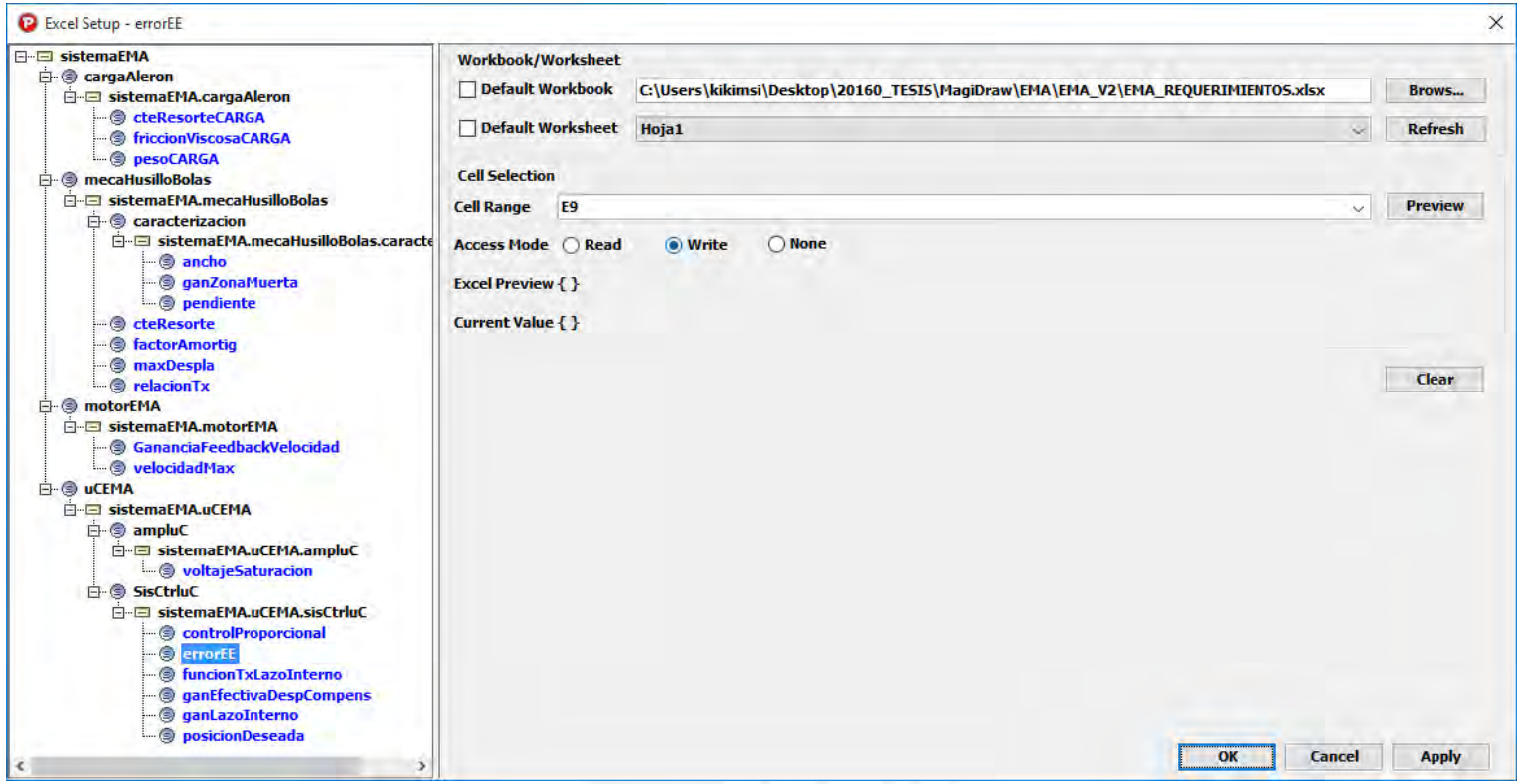


Figura 4.34 Configuración para escribir el valor en *Excel* mediante *ParaMagic*;
Fuente propia

Tabla 4-3 EMA_REQUERIMIENTOS.xlsx con condiciones para validar trazabilidad; Fuente propia

	A	B	C	D	E	F	G	H
1					Resultado	Condición		Cumple
2	1_Requerimientos							
3	#	Id	Name	Text				
4	1	1	E_ActuarAutomaticamenteAcordeInstruccionesPiloto	EL sistema debe hacer actuar automáticamente al alerón acorde a las instrucciones del piloto/pilotoautomático	1	1.1 y 1.2		1
5	2	1.1	E_ControlyComandos	El sistema debe ser capaz de capaz de controlar y comandar al alerón	1	4, 5, 6, 12 y 13		1
6	3	1.2	E_ActuacionAleron	El sistema debe mover al alerón acuerdo a las instrucciones del piloto transmitidas por la unidad de control	1	7,14,15 y 17		1
7	4	4	FeedbackCtrlUnit	El sistema debe proporcionar un feedback a la unidad de control				1
8	5	5	RecibirInstruccionesCtrlUnit	El sistema debe recibir instrucciones de la unidad de control				1
9	6	6	MantenerIncidencia	El sistema debe mantener la incidencia del alerón y tener un error en estado estable menor a 5%	4.888	<	5	1

Tabla 4-3 Continuación

	A	B	C	D	E	F	G	H
1					Resultado	Condición		Cumple
2	1_Requerimientos							
3	#	Id	Name	Text				
10	7	7	ModificarIncedencia	El sistema debe poder modificar la incidencia conforme a las instrucciones del piloto/pilotoautomatico				1
11	8	8	E_ConformeEstandardSeguridad	El sistema debe de cumplir y ser conforme a los estándares de seguridad				0
12	9	9	E_DimensionesGenerales	El sistema debe de insertarse en el ala del avión. Las dimensiones generales deben poder acoplarse en el ala				0
13	10	10	E_MedioAmbiente	El sistema debe poder adaptarse al ambiente que lo rodea (temperatura, humedad...)				0
14	11	11	E_Alimentacion	El sistema debe ser alimentado de manera suficiente en cuanto a voltaje y corriente en el avión				1
15	12	12	TraducirInstruccionesPiloto	La unidad de control debe recibir la señal por FBW y poder entender mediante una señal análoga que debe realizar				1

Tabla 4-3 Continuación

	A	B	C	D	E	F	G	H
1					Resultado	Condición		Cumple
2	1_Requerimientos							
3	#	Id	Name	Text				
16	13	13	RegularEnergiaElectrica	El sistema debe poder regular la energía eléctrica de control para que trabaje a su voltaje nominal				1
17	14	14	TransformarEnergiaElectricaMecanica	El sistema debe poder transformar la energía trifásica en energía mecánica				1
18	15	15	TransmitirEnergiaMecánica	El sistema debe transmitir la energía mecánica para mover la carga				1
19	16	17	AdaptarEnergiaMecanica	El sistema debe de adaptar la energía mecánica mediante una caja reductora				1

CONCLUSIONES Y RECOMENDACIONES

En esta tesis se propone integrar la normativa del proceso de diseño mecatrónico, con la normativa vinculada a la Ingeniería de Sistemas, buscando que ambas conversen bajo el objetivo de trazabilidad de requerimientos. Para tal fin se emplea *SysML,* herramienta que permite modelar a un sistema complejo en diferentes dominios y, aún más, permite vincular los requerimientos de diseño y sus diversos modelos.

En la introducción se analiza la situación actual del diseño, poniendo en contraste el enfoque basado en modelos y el proceso secuencial que predomina en las metodologías clásicas de diseño.

Luego, en el capítulo dos se detalla como un sistema puede descomponerse y modelarse. Los modelos organizados por paquetes, permiten tener una vista global de los otros modelos y la vinculación directa o indirecta con estos. Los modelos organizados por bloques, permiten definir los componentes.

Los modelos de funcionalidad, ofrecen a los diseñadores la posibilidad de contextualizar el uso del sistema mediante actores externos y también mediante sus elementos. Los modelos de comportamiento, enfatizan los estados que puede tener el sistema o parte de un sistema. Además, permiten aclarar, cómo mediante una secuencia de actividades el sistema funciona o se logra una meta de una parte del sistema. Asimismo, permiten detallar las propiedades de los elementos y como fluye la información entre y en los sistemas. Los modelos paramétricos permiten que los elementos tengan propiedades que limiten su comportamiento, esto es, a través de modelos matemáticos.

En el capítulo tres se exponen los vínculos que *SysML* permite entre los elementos del modelo y cómo pueden manifestarse a través de su notación. Dichos vínculos refuerzan la trazabilidad de requerimientos que deben de presentarse en las etapas tempranas de diseño y hasta la finalización del sistema. Finalmente, a través del estudio de un actuador electro-mecánico se verifica la viabilidad del uso de *SysML*

junto a *Matlab/Simulink* y *Excel* para garantizar la trazabilidad de requerimientos de diseño.

Cabe mencionar que, durante el desarrollo del caso de estudio, los modelos utilizados pueden llegar a ser simples para un especialista en aeronaves; sin embargo, la tesis no está centrada en el modelo del estudio sino en demostrar que es posible modelar un sistema mecatrónico en *SysML* y que sus requerimientos sean trazables. Es cierto que el modelo puede llegar a ser más complejo; pero también es cierto que *SysML* permite realizar modelados complejos, lo cual se detalló en el capítulo dos.

Conclusiones

La contribución principal de la presente tesis ha sido la demostración de que *SysML,* el lenguaje de modelado de sistemas, puede ser usado para garantizar de la trazabilidad de requerimientos a lo largo del diseño de un sistema mecatrónico. El desarrollo del caso de estudio trabajado permite:

- Reforzar la necesidad de acoplar conceptos de trazabilidad en las metodologías actuales en el campo de la ingeniería mecatrónica. Mediante el uso de palabras como: satisfecho, verificado, afinado y derivado se logra expresar la relación entre un requerimiento y un elemento modelo del sistema.
- El enlazar *SysML* con *Matlab/Simulink* y *Excel* para el caso del actuador electro-mecánico permite la verificación del cumplimiento de requerimientos a lo largo del diseño. Además, es posible utilizar este mismo concepto a otros sistemas mecatrónicos ya que *Matlab/Simulink* ofrece la actualización de parámetros claves durante el análisis matemático de un dominio del sistema. Por otro lado, las hojas de cálculo de *Excel* otorgan las cualidades para que los requerimientos renueven su estado en cualquier etapa del diseño.

Implementar la trazabilidad de requerimientos utilizando MBSE es provechoso para el diseño de un actuador electro-mecánico. Esta tesis puede servir de guía para implementarla en sistemas mecatrónicos. Además, si se cuenta con una lista de *requerimientos* bien fundamentada y con las herramientas de software apropiada, la implementación será de complejidad media dependiendo de cómo se modelan los sistemas.

Los costos de la licencia del software de *MagicDraw* asciende a 186 USD y el precio anual por uso es de 60 USD, adicional a este costo debe de comprarse los *plug-ins SysML* y *Paramagic*, ambos ofrecidos por *MagicDraw*.

Recomendaciones de trabajos futuros

A pesar de demostrar que los requerimientos pueden trazarse, la herramienta principal de trabajo debe poder vincularse con otras herramientas de diseño. Dado que los sistemas mecatrónicos son sistemas complejos que necesitan ayuda de otras herramientas de análisis en la ingeniería, en un trabajo futuro es necesario que herramientas CAD de mecánica como *Autodesk Inventor*, *SolidWorks* o *Creo* y herramientas de automatización de tarjetas de electrónica como *CadSoft Eagle* estén presentes en el software para una integración total de los modelos.

BIBLIOGRAFÍA

[1] D. Parulekar, "Why does your company need Mechatronics ?," *EY*, 2014. [Online]. Available: http://advisoryindiablog.ey.com/2014/07/28/mechanical-electronic-computing-why-does-your-company-need-mechatronics/. [Accessed: 30-Mar-2016].

[2] R. Sell and M. Tamre, "Integration of V-model and SysML for advanced mechatronics system design," *Proc. Res. Educ. Mechatronics Conf. REM05*, pp. 276–280, 2005.

[3] H. Ji, O. Lenord, and D. Schramm, "A Model Driven Approach for Requirements Engineering of Industrial Automation Systems.," *Eoolt*, pp. 9–18, 2011.

[4] H. Anacker, R. Dumitrescu, J. Gausemeier, P. Iwanek, and T. Schierbaum, "Methodology for the Identification of Potentials for the Integration of Self-optimization in Mechatronic Systems," *Procedia Technol.*, vol. 15, pp. 17–26, 2014.

[5] A. Czauderna, J. Cleland-Huang, M. Cinar, and B. Berenbach, "Just-in-time traceability for mechatronics systems," *2012 2nd IEEE Int. Work. Requir. Eng. Syst. Serv. Syst. RESS 2012 - Proc.*, pp. 1–9, 2012.

[6] R. Sell and M. Tamre, "Mechatronics System Design Process and Methodologies."

[7] N. Magic, "MagicDraw UML," 2014. .

[8] G. Pahl and W. Beitz, *Engineering design: a systematic approach*, vol. 1. Springer Science & Business Media, 2007.

[9] G. V.D.I, *VDI 2221 Systematic Approach to the Design of Technical Systems and Products*. Düsseldorf: VDI, 1987.

[10] J. Gausemeier and S. Möhringer, "Die neue Richtlinie VDI2206: Entwicklungsmethodik für mechatronische Systeme," *VDI. Ber.*, pp. 43–67, 2003.

[11] T. Doran, "IEEE 1220: For practical systems engineering," *IEEE Comput. Soc.*, vol. 39, no. 5, pp. 92–94, 2006.

[12] EIA, "EIA 632," *Process. Eng. a Syst.*, 1998.

[13] S. Engineering and S. Committee, *IEEE Guide — Adoption of ISO / IEC TR 24748-3 : 2011 Systems and Software Engineering — Life Cycle Management — Part 2 : Guide to the Application of ISO / IEC 15288 (System Life Cycle Processes)*, no. April. IEEE Computer Society, 2012.

[14] S. Sheard and J. G. Lake, "Systems engineering standards and models compared," *Int. Symp. Syst. Eng.*, vol. 1220, no. January, p. 8, 1998.

[15] K. Thramboulidis and S. Scholz, "Integrating the 3+1 SysML view model with safety engineering," *Proc. 15th IEEE Int. Conf. Emerg. Technol. Fact. Autom.*

ETFA 2010, 2010.
[16] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq, "A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies," *Inf. Softw. Technol.*, vol. 54, no. 6, pp. 569–590, 2012.
[17] C. Kilger, A. Reisch, and R. Indefrey, "The secret Behind Mechatronics," vol. 6, no. 2, pp. 44–53, 2014.
[18] F. Mhenni, "Safety Analysis Integration in a Systems Engineering Approach for Mechatronic Systems Design," École Centrale Paris, 2014.
[19] S. Friedenthal, A. Moore, and R. Steiner, *A practical guide to SysML: the systems modeling language*. Morgan Kaufamann, 2014.
[20] "Object Managaement Group (OMG) - Systems Modeling Language (SysML)." [Online]. Available: http://www.omgsysml.org.
[21] "Object Management Group (OMG)." [Online]. Available: http://www.omg.org/.
[22] T. Weilkiens, *Systems Engineering with SysML/UML: modeling, analysis, design*, vol. 1. Morgan Kaufamann, 2011.
[23] I. Moir and A. Seabridge, *Aircraft Systems: Mechanical, electrical, and avionics subsystems integration*. John Wiley & Sons, 2008.
[24] S. Habibi, J. Roach, and G. Luecke, "Inner-Loop Control for Electromechanical (EMA) Flight Surface Actuation Systems," *J. Dyn. Syst. Meas. Control*, vol. 130, no. 5, 2008.
[25] E. Balaban, "A Diagnostic Approach for Electro-Mechanical Actuators in Aerospace Systems," pp. 15–17, 2009.
[26] ISO EN, "9001:2008 Quality management systems—Requirements." 2008.