UNIVERSITAT JAUME·I

Modelado y simulación híbrida de sistemas de fabricación multi-etapa orientado a la evaluación de la calidad geométrica y la productividad

AUTOR

Sergio Benavent Nácher

DIRECTORES

Dr. Fernando Romero Subirón

Dr. Pedro Rosado Castellano

Enero de 2024

Programa de Doctorado en Tecnologías Industriales y Materiales

Escuela de Doctorado de la Universitat Jaume I

Modelado y simulación híbrida de sistemas de fabricación multi-etapa orientado a la evaluación de la calidad geométrica y la productividad.

Memoria presentada por Sergio Benavent Nácher

para optar al grado de doctor por la Universitat Jaume I

AUTOR

SERGIO BENAVENT NACHER - NIF: 20857327F

Firmado digitalmente por SERGIO BENAVENT NACHER - NIF:20857327F
Razón: Soy el autor de este documento
Fecha: 2024.01.11 18:04:59+01'00'

Sergio Benavent Nácher

DIRECTORES

FERNANDO| ROMERO| SUBIRON

Firmado digitalmente por FERNANDO|ROMERO|SUBIRON
Fecha: 2024.01.12 10:42:35 +01'00'

Dr. Fernando Romero Subirón

PEDRO| ROSADO| CASTELLANO

Firmado digitalmente por PEDRO|ROSADO|CASTELLANO
Fecha: 2024.01.12 10:18:07 +01'00'

Dr. Pedro Rosado Castellano

Castelló de la Plana, enero de 2024

Financiación

Contratos predoctorales:

- Contrato como investigador a cargo de fondos del grupo de investigación Ingeniería de Fabricación de la Universitat Jaume I.
  Periodo: 22/01/2018 - 20/07/2018

- Contrato como personal investigador en formación predoctoral, financiado por el programa de ayudas predoctorales para la formación de personal investigador, financiadas por la Universitat Jaume I.
  Periodo: 01/03/2019 – 31/08/2019
  Referencia: PREDOC/2018/19

- Contrato como personal investigador en formación predoctoral, financiado por la Generalitat Valenciana y por el Fondo Social Europeo (ACIF2019).
  Periodo: 01/09/2019 – 13/02/2021
  Referencia: ACIF/2019/095

- Contrato como profesor ayudante de la Universitat Jaume I.
  Periodo: 13/02/2021 - actualidad

Proyectos de investigación:

- Modelado e implementación del gemelo digital para sistemas ciber-físicos de producción (CPPS) orientado al aseguramiento de la calidad geométrica de producto.
  Código: 20I479.01/1
  Entidad financiadora: Universitat Jaume I.

- Metodología para el modelado e implementación del Gemelo digital de un sistema de fabricación multietapa orientado a la optimización del funcionamiento.
  Código: 22I597.01/1
  Entidad financiadora: Universitat Jaume I.

A mi familia.

Resumen

Durante los últimos años, la evolución del sector industrial y de los sistemas de fabricación ha iniciado un complejo y progresivo cambio de paradigma que apunta hacia una industria más inteligente, sensorizada y sostenible. Iniciativas como *Smart Factory*, *Industrie 4.0* o *Factory of the Future* apuntan hacia sistemas de fabricación más inteligentes, autónomos, reactivos y adaptativos. La creciente complejidad de los sistemas de fabricación ha promovido la adopción de la Ingeniería de Sistemas Basada en Modelos (*Model-based System Engineering* - MBSE) en la industria, un enfoque centrado en desarrollar, gestionar y controlar modelos que capturen los intereses de las diversas partes interesadas (*stakeholders*), incluyendo diferentes vistas (estructural, funcional, comportamental, etc.).

Desde el punto de vista del diseño de sistemas de fabricación, y en especial en sistemas de fabricación multietapa, otros aspectos potenciados en los últimos años por las citadas iniciativas son la flexibilidad y reconfigurabilidad, es decir, la capacidad de adaptación a diferentes escenarios o tipos de producto, pudiendo requerir pequeños ajustes en los equipos. Además, una parte importante de las iniciativas desarrolladas en los últimos años (*Lean Manufacturing*, *Zero Defect Manufacturing* o *Production Quality*, entre otros) están orientadas hacia la mejora de la calidad y la reducción de los desperdicios. A pesar de que la calidad puede ser analizada desde diversos puntos de vista, esta investigación se centra en las características geométricas del sistema de fabricación que tendrán una influencia directa en la calidad del producto. De forma más específica, es objeto de interés el análisis de las desviaciones geométricas del producto y su propagación o acumulación en el contexto de los sistemas de fabricación multietapa.

En este contexto, la presente investigación se centra en desarrollar y validar una metodología que defina métodos (lenguajes y procedimientos) y herramientas para el modelado y simulación de sistemas de fabricación multietapa durante el diseño, configuración y puesta a punto (*commissioning*) del sistema de fabricación. Estos modelos, centrados en la productividad y el control y la mejora de la calidad geométrica, requieren de simulaciones híbridas que combinen diversos dominios. En este sentido, la metodología también debe asegurar la consistencia de estos modelos de simulación con otros modelos del sistema, haciendo frente a uno de los mayores desafíos que supone la adopción del enfoque MBSE. Para ello, se explora el uso de un lenguaje propio de la ingeniería de sistemas como es SysML, definiendo modelos que aseguren la consistencia y la correcta construcción (*well-formedness*) en base a esta semántica específica. Una vez definidos y validados estos modelos, se propone implementar transformaciones automáticas de los modelos a un lenguaje propio de la simulación, como es el caso de Modelica, obteniendo una simulación compilable y ejecutable.

Abstract

During recent years, the evolution of the industrial sector and manufacturing systems has initiated a complex and progressive paradigm shift that points towards a more intelligent, sensorized and sustainable industry. Initiatives such as Smart Factory, Industrie 4.0 or Factory of the Future point towards manufacturing systems that are more intelligent, autonomous, reactive and adaptive. The increasing complexity of manufacturing systems has promoted the adoption of Model-based Systems Engineering (MBSE) in industry, an approach focused on developing, managing and controlling models that capture the interests of diverse interested parties (stakeholders), including different views (structural, functional, behavioral, etc.).

From the point of view of the design of manufacturing systems, and especially in multistage manufacturing systems, other aspects enhanced in recent years by the aforementioned initiatives are flexibility and reconfigurability, that is, the ability to adapt to different scenarios or types. of product, which may require small adjustments to the equipment. Furthermore, an important part of the initiatives developed in recent years (Lean Manufacturing, Zero Defect Manufacturing or Production Quality, among others) are oriented towards improving quality and reducing waste. Although quality can be analyzed from various points of view, this research focuses on the geometric characteristics of the manufacturing system that will have a direct influence on the quality of the product. More specifically, the analysis of product geometric deviations and their propagation or accumulation in the context of multistage manufacturing systems is of interest.

In this context, this research focuses on developing and validating a methodology that defines methods (languages and procedures) and tools for the modeling and simulation of multistage manufacturing systems during the design, configuration and commissioning of the manufacturing system. manufacturing. These models, focused on productivity and control and improvement of geometric quality, require hybrid simulations that combine various domains. In this sense, the methodology must also ensure the consistency of these simulation models with other models of the system, addressing one of the greatest challenges posed by the adoption of the MBSE approach. To do this, the use of a systems engineering language such as SysML is explored, defining models that ensure consistency and correct construction (well-formedness) based on this specific semantics. Once these models are defined and validated, it is proposed to implement automatic transformations of the models to a simulation language, such as Modelica, obtaining a compileable and executable simulation.

Resum

Durant els darrers anys, l'evolució del sector industrial i dels sistemes de fabricació ha iniciat un complex i progressiu canvi de paradigma que apunta cap a una indústria més intel·ligent, sensoritzada i sostenible. Iniciatives com *Smart Factory*, *Industrie 4.0* o *Factory of the Future* apunten cap a sistemes de fabricació més intel·ligents, autònoms, reactius i adaptatius. La complexitat creixent dels sistemes de fabricació ha promogut l'adopció de l'Enginyeria de Sistemes Basada en Models (Model-based System Engineering - MBSE) a la indústria, un enfocament centrat en desenvolupar, gestionar i controlar models que capturen els interessos de les diverses parts interessades (stakeholders), incloent-hi diferents vistes (estructural, funcional, comportamental, etc.).

Des del punt de vista del disseny de sistemes de fabricació, i en especial en sistemes de fabricació multietapa, altres aspectes potenciats en els darrers anys per les iniciatives esmentades són la flexibilitat i reconfigurabilitat, és a dir, la capacitat d'adaptació a diferents escenaris o tipus de producte, podent requerir petits ajustaments als equips. A més, una part important de les iniciatives desenvolupades en els darrers anys (*Lean Manufacturing, Zero Defect Manufacturing* o *Production Quality*, entre d'altres) estan orientades cap a la millora de la qualitat i la reducció de les deixalles. Tot i que la qualitat pot ser analitzada des de diversos punts de vista, aquesta investigació se centra en les característiques geomètriques del sistema de fabricació que tindran una influència directa en la qualitat del producte. De manera més específica, és objecte d'interés l'anàlisi de les desviacions geomètriques del producte i la seva propagació o acumulació en el context dels sistemes de fabricació multietapa.

En aquest context, aquesta investigació se centra a desenvolupar i validar una metodologia en què es defineixen mètodes (llenguatges i procediments) i eines per al modelatge i simulació de sistemes de fabricació multietapa durant el disseny, configuració i posada a punt (commissioning) del sistema de fabricació. Aquests models, centrats en la productivitat i el control i la millora de la qualitat geomètrica, requereixen simulacions híbrides que combinen diversos dominis. En aquest sentit, la metodologia també ha d'assegurar la consistència d'aquests models de simulació amb altres models del sistema, fent front a un dels desafiaments més grans que suposa l'adopció de l'enfocament MBSE. Per fer-ho, s'explora l'ús d'un llenguatge propi de l'enginyeria de sistemes com és SysML, definint models que asseguren la consistència i la construcció correcta (well-formedness) en base a aquesta semàntica específica. Un vegada definits i validats aquests models, es proposa implementar transformacions automàtiques dels models a un llenguatge propi de la simulació, com és el cas de Modelica, obtenint una simulació compilable i executable.

Agradecimientos

A toda mi familia por su apoyo incondicional a lo largo de toda mi formación y, en especial, durante los estudios de doctorado.

A mi chica, María José, que ha vivido junto a mí todos los momentos, buenos y malos, de los últimos años.

A mis directores, Fernando Romero y Pedro Rosado. Sin su consejo, paciencia e incansable ayuda nada de esto hubiera sido posible.

Y en general, a todas las personas con las que he compartido esta memorable etapa de mi vida.

Índice de contenidos

Listado de figuras

Listado de tablas

1. Introducción

1.1. Contexto y motivación

Durante los últimos años, la evolución del sector industrial y de los sistemas de fabricación ha iniciado un complejo y progresivo cambio de paradigma que apunta hacia una industria más inteligente, sensorizada y sostenible. Así, numerosas iniciativas han sido desarrolladas para promover una industria más distribuida y conectada, donde los sistemas de fabricación posean consciencia sobre sí mismos y sobre el entorno, y que sean más inteligentes, autónomos, reactivos y adaptativos. En esta línea encontramos iniciativas como *Smart Factory* [1], *Industrie 4.0* [2] o *Factory of the Future* [3], propuestas en las que los sistemas avanzados de fabricación son sistemas complejos capaces de trabajar con la mínima intervención humana posible, pero funcionando de forma compatible con la supervisión y la colaboración humana [4]. En esta línea, algunos investigadores proponen desarrollar sistemas de control con capacidad de decisión en tiempo real mediante la combinación de mecanismos predictivos y reactivos [5]. La incorporación de estas capacidades (predictivas y reactivas) requieren de la captura y procesamiento de datos, la evaluación del estado y las capacidades tecnológicas (*capabilities*) de los recursos, y la generación de conocimiento a partir de estos datos para guiar la toma de decisiones en el diseño, configuración o evolución del propio sistema de fabricación, incluido su sistema de control.

Por otra parte, la sostenibilidad es otra de las cuestiones principales abordadas en el desarrollo de la nueva industria. Si bien este aspecto puede adquirir muchos enfoques diferentes (económico, social, medioambiental, etc.), una parte importante de las iniciativas desarrolladas en los últimos años están orientadas hacia la mejora de la calidad y la reducción de los desperdicios, como es el caso de iniciativas como Lean Manufacturing [6], Zero Defect Manufacturing [7] o Production Quality [8]. En concreto, el paradigma Production Quality combina calidad, producción logística, y métodos y herramientas de mantenimiento con el fin de mantener el rendimiento y el nivel de servicio de las piezas conformes bajo control y mejorarlos con el tiempo, con un desperdicio mínimo de recursos y materiales [8]. Siguiendo estas directrices, la presente investigación se centra en la gestión de la calidad geométrica y su influencia sobre la productividad del sistema de fabricación, desarrollando modelos para el análisis de las desviaciones geométricas y su propagación o acumulación en el contexto de los sistemas de fabricación multietapa (Multi-stage Manufacturing Systems - MMS).

En este contexto, tanto el diseño y desarrollo integrado de producto, proceso y sistema de fabricación, como el modelado y la simulación para dar soporte a la verificación y validación de sistemas, han adquirido una especial importancia. Si bien la industria se ha apoyado tradicionalmente en el uso de documentos o de modelos de diferente naturaleza (físicos, abstractos, matemáticos, etc.), éstos se han desarrollado con un reducido o nulo nivel de integración. Sin embargo, la creciente complejidad de los sistemas de fabricación y de los tipos de modelos existentes ha promovido en las últimas décadas la adopción del enfoque de Ingeniería de Sistemas Basada en Modelos (Model-based System Engineering - MBSE) en la industria [9]. Este enfoque MBSE está centrado en desarrollar, gestionar y controlar modelos que capturen los intereses de las diversas partes interesadas (stakeholders), incluyendo diferentes vistas (estructural, funcional, comportamental, etc.) del sistema de interés. Las metodologías de trabajo alineadas con el enfoque MBSE manejan un conjunto heterogéneo de modelos para la gestión de conceptos de diferentes disciplinas y dominios, pertenecientes a diversos niveles de granularidad y creados con diferentes lenguajes y herramientas. De esta heterogeneidad emergen diferentes retos, por ejemplo, relacionados con la consistencia de los modelos o la transformación automática entre modelos, cuestiones especialmente importantes desde el punto de vista de la integración producto-proceso.

Poniendo el foco en el diseño de sistemas de fabricación, el futuro industrial apunta hacia sistemas cada vez más flexibles y reconfigurables, es decir, con una gran capacidad de adaptación a diferentes escenarios o tipos de producto, pudiendo requerir pequeños ajustes en los equipos [10]. Este incremento de las capacidades del sistema de fabricación permite abarcar una mayor variedad de productos con los mismos recursos o, por el contrario, reducir la cantidad de recursos necesarios para una determinada gama de producto. Sin embargo, esta estrategia requiere de mayores esfuerzos en las etapas de diseño y puesta a punto del sistema, una tarea en la que resulta de gran ayuda disponer de modelos y sistemas de simulación que permitan definir y validar las soluciones de diseño antes de ser trasladadas al sistema real.

Así pues, siguiendo las tendencias marcadas por las iniciativas y enfoques mencionados, esta investigación pone el foco de atención en el desarrollo de procedimientos de análisis, explorando el papel del modelado y la simulación del sistema de fabricación durante las etapas de diseño y puesta a punto (configuración) del sistema de fabricación. En concreto, el estudio está centrado en el análisis mediante simulación de diversas estrategias de control centradas en la variabilidad geométrica y la evaluación de la calidad geométrica y la productividad, siguiendo los principios del Production Quality. Por tanto, el estudio se centra en el desarrollo de modelos de simulación para el análisis del comportamiento con base matemática, un testeo virtual del sistema referente (sistema de fabricación) orientado a prever su comportamiento, estudiar soluciones alternativas y validar el sistema antes de su puesta en marcha, atendiendo a unos requerimientos impuestos en etapas iniciales de diseño.

Aunque la investigación está centrada en el análisis de etapas tempranas del ciclo de vida del sistema de fabricación (diseño y puesta a punto), los avances en materia de modelado y el desarrollo de modelos de simulación de este tipo de sistemas también

pueden facilitar en gran medida el desarrollo de aplicaciones más vinculadas a mecanismos reactivos. De hecho, las herramientas de modelado y simulación también están adquiriendo gran relevancia con investigaciones centradas en conceptos como gemelos digitales, sombras digitales y modelos digitales [11]. Sin embargo, estas cuestiones más vinculadas con el concepto de sistema de producción ciber-físico (Cyber-Physical Production System - CPPS) quedan fuera del alcance establecido en esta investigación, que aborda el estudio como sistema mecatrónico.

Conviene tener en cuenta además que, frente a sistemas complejos como los actuales sistemas de fabricación, el uso de herramientas de simulación específicas de un único dominio no es una solución válida, sobre todo si se pretende adoptar la tendencia integradora de los sistemas producto, proceso, sistema de fabricación. Como alternativa, es necesario adoptar otras herramientas con capacidad de integración de diferentes dominios, en los que desarrollar simulaciones híbridas que permitan modelar de manera combinada e integrada diversos aspectos del sistema, incluyendo la influencia que puede existir entre ellos. A pesar de los numerosos esfuerzos realizados en el campo del modelado y la simulación bajo estos paradigmas, la mayoría de estos trabajos están enfocados principalmente en el análisis de producción y mantenimiento con simulaciones de eventos discretos. Sin embargo, existe una falta de investigación orientada al desarrollo de marcos para la gestión de variaciones geométricas tanto en mecanismos predictivos como reactivos, especialmente en el modelado y simulación de los efectos de las variaciones geométricas para el aseguramiento de la calidad, la productividad y sus interrelaciones.

El tratamiento y el modelado de las variaciones geométricas ha merecido un gran interés de la comunidad científica en las últimas décadas, si bien la mayoría de los esfuerzos se han centrado en el modelado conceptual o descriptivo de la geometría y el toleranciado mediante modelos de presentación o de interpretación. Sin embargo, en el contexto de la industria conectada cobran especial interés los conocidos como modelos de representación, ya que poseen la capacidad adicional de ser implementados computacionalmente [12]. En lo referido a trabajos orientados al análisis de toleranciado, se han desarrollado técnicas y herramientas asistidas por computador para la representación y análisis de desviaciones geométricas [13], como pueden ser las aplicaciones CAT (Computer Aided Tolerancing). Otros modelos orientados a manejar los datos geométricos en el diseño y validación de sistemas mecatrónicos complejos han integrado teorías como el modelado geométrico TTRS (Technologically and Topologically Related Surfaces), dando soporte a conceptos de la norma GPS (Geometric Product Specification) y algunas técnicas de análisis [14] [15]. Sin embargo, hasta el momento no se han encontrado trabajos en la bibliografía que adopten un enfoque sistémico de esta cuestión y hagan uso de la riqueza semántica suficiente para soportar la conceptualización del producto y de su geometría de forma consistente y sin ambigüedad. La carencia de trabajos que aborden el modelado de sistemas de fabricación basado en lenguajes de sistemas (como SysML) y con una orientación al aseguramiento de la consistencia en el análisis o simulación de la variabilidad geométrica ha sido una de las principales motivaciones de esta investigación.

Conviene además destacar que la mayoría de los esfuerzos vinculados con el modelado de variaciones geométricas se han desarrollado en el contexto de los sistemas de fabricación uni-etapa, abordando el modelado de artefactos o ensambles, un enfoque a partir del cual se han desarrollado muchas propuestas entre las que podemos destacar [16], [17] o [18]. Sin embargo, las propuestas dirigidas a definir modelos de análisis/simulación de sistemas de fabricación multi-etapa y que integren la evaluación de la calidad geométrica y la productividad son muy escasas. A pesar de que existen modelos matemáticos que abordan la propagación de errores geométricos en MMS, como las técnicas relacionadas con la metodología Stream of Variation (SoV) [19] [20] [21], éstas no han sido integradas en modelos de simulación híbridos que permitan evaluar la propagación de estos errores y su influencia en otros aspectos propios de los MMS's, como la productividad.

Así pues, de acuerdo con las necesidades requeridas por las iniciativas y paradigmas de la industria y las carencias identificadas en la literatura, esta investigación propone el desarrollo de una metodología que permita definir modelos de simulación ejecutables, multidominio y multiescala, soportando la evaluación de las funcionalidades y modos de operación de MMS's, tanto en términos de calidad geométrica como productividad. Además, con el fin de asegurar la correcta definición de estos modelos de simulación y su consistencia con otros modelos del sistema de fabricación, se explora el modelado de estas simulaciones en lenguajes propios de la ingeniería de sistemas (como SysML), y su posterior transformación a lenguajes que soporten la ejecución de simulaciones. Además, tanto el modelado estructural y de comportamiento dinámico del MMS como la propagación de las desviaciones geométricas serán abordados considerando propuestas y modelos existentes, que serán analizados, comparados y adoptados (completa o parcialmente) para dar soporte al modelado y simulación del sistema.

1.2. Objetivos de la investigación

El propósito anteriormente enunciado se concreta en el desarrollo y validación de una metodología que defina un procedimiento apoyado por el uso de determinados lenguajes y herramientas (entornos, recursos, etc.) para el modelado y simulación de sistemas de fabricación multietapa durante el diseño, configuración y puesta a punto (commissioning) del sistema de fabricación, centrando estos modelos en el control y la mejora de la calidad geométrica del producto y la productividad del sistema, lo que requiere de simulaciones híbridas que combinen diversos dominios. En este sentido, la metodología no solo debe dar soporte a la definición de modelos de simulación ejecutables y multidominio, sino también debe asegurar la consistencia de estos modelos de simulación con otros modelos del sistema, haciendo frente a uno de los mayores desafíos que supone la adopción del enfoque MBSE. Para ello, se explora la definición de estos modelos de simulación en un lenguaje propio de la ingeniería de sistemas como es SysML. Concretamente, se explora la definición y uso de lenguajes específicos de dominio a través del mecanismo de creación de perfiles con el fin de incorporar la semántica propia de los dominios manejados y, de este modo, asegurar la consistencia y la correcta construcción (well-formedness) de los modelos en base a

esta semántica específica. Una vez definidos y validados estos modelos, se propone implementar transformaciones automáticas de los modelos a un lenguaje propio de la simulación, como es el caso de Modelica. Para apoyar y facilitar estas transformaciones, se explora tanto el uso de perfiles específicos como el mapeo entre librerías de elementos reutilizables definidos en ambos lenguajes. Finalmente, tras la transformación de modelo a texto (model-to-text transformation), se obtiene una simulación compilable y ejecutable.

1.3. Preguntas de investigación

A partir del objetivo general descrito anteriormente, se formulan tres preguntas de investigación iniciales que orientan los ejes principales de la investigación y a las que este trabajo dará respuesta a lo largo del documento. Se trata de tres preguntas bastante generales que, a medida que se profundice en el estudio de cada cuestión, derivarán en preguntas más detalladas, refinadas o derivadas. A continuación, se presentan estas tres preguntas de investigación, acompañadas de una breve descripción de los contenidos y cuestiones involucradas en cada una de ellas.

1. *En el contexto de los paradigmas y enfoques actuales, y centrados en la etapa de diseño de sistemas de fabricación, ¿existen metodologías (procedimientos, lenguajes y herramientas) para desarrollar modelos de simulación que permitan el análisis conjunto de la calidad y productividad de sistemas de fabricación multietapa?*

   En primer lugar, es fundamental conocer las propuestas metodológicas existentes orientadas al diseño de sistemas de fabricación y su análisis, especialmente aquellas que abordan el modelado y análisis de la geometría y la calidad geométrica y/o la productividad. Estos trabajos previos serán el punto de partida para identificar aspectos de interés que puedan ser aprovechados e integrados en la propuesta desarrollada, o bien poner de relieve carencias o aspectos problemáticos que solventar a la hora de desarrollar la metodología propia. Resulta conveniente aclarar en este punto que a lo largo del documento se hace uso del término "metodología" para referirse no sólo a la descripción de un procedimiento o unos pasos a seguir, sino que también incluye la selección o definición de lenguajes y herramientas que se utilicen a lo largo de dicho procedimiento. En este sentido, más allá de revisar las propuestas metodológicas, será conveniente revisar los lenguajes tanto del ámbito de la ingeniería de sistemas (como SysML) como los lenguajes de análisis y simulación de sistemas (como Modelica), así como los diversos entornos y plataformas que soportan estos lenguajes.

2. *¿Qué lenguajes y plataformas son los más adecuados para soportar el modelado de sistemas de simulación híbridos? ¿Qué ventajas puede ofrecer el desarrollo de una arquitectura que contemple tanto modelos SysML de diseño y análisis como modelos de simulación ejecutables Modelica, así como la transformación entre ellos?*

   Esta pregunta está orientada a seleccionar lenguajes y herramientas que den soporte al modelado de sistemas de simulación híbridas, con el fin de desarrollar

en ellas simulaciones que integren el análisis del comportamiento del sistema de simulación tanto a nivel de flujo logístico (flujo de materiales) como de propagación de las desviaciones geométricas del producto a lo largo del sistema multietapa. Como se verá en capítulos posteriores del documento, los lenguajes de simulación como Modelica soportan este tipo de modelos, pero en general se trata de lenguajes que carecen de mecanismos para considerar la semántica propia del dominio específico analizado y así validar o asegurar la consistencia de los mismos. Con el fin de superar esta limitación, se explora el uso de lenguaje SysML, que a través de la creación de perfiles y la definición de reglas en cada estereotipo sí da soporte a estas comprobaciones de consistencia específica de dominio. Así, los modelos implementados en SysML y validados pueden ser posteriormente transformados de forma automática en modelos de simulación ejecutables, implementados en lenguajes y entornos propios de la simulación. En esta estrategia, será fundamental adoptar un enfoque MBSE que permita manejar de manera adecuada los diversos tipos de modelos y lenguajes que se contemplen en la propuesta metodológica.

3. *¿Qué conceptos (semántica específica de dominio) es necesario considerar en los modelos de simulación híbrida de sistemas de fabricación, centrados en la calidad geométrica y la productividad? ¿Cómo se puede implementar esta semántica en el modelo para asegurar su consistencia estructural y comportamental? ¿Los perfiles SysML pueden servir como mecanismo para asegurar la consistencia y correcta definición de los modelos?*

   Como se introduce en la pregunta anterior, la definición de perfiles SysML es un mecanismo que permite extender el lenguaje para incorporar conceptos específicos de un dominio e incorporar reglas sobre los nuevos estereotipos. Este mecanismo se puede emplear para asegurar la consistencia de los modelos en los que el perfil se aplica, tanto a nivel estructural como comportamental. Para ello, resulta fundamental hacer una profunda reflexión sobre la semántica del dominio de interés, identificando una serie de conceptos clave que serán definidos como estereotipos y una serie de restricciones entre dichos conceptos, que se traducirán en restricciones definidas en cada uno de los estereotipos considerados. En el contexto de esta propuesta, los principales dominios a tratar serán: a) la definición geométrica de artefactos, incluyendo las desviaciones geométricas y aspectos propios del análisis de las mismas; b) modelado de sistemas de fabricación multietapa, abordando tanto la parte estructural como comportamental; y c) transformaciones de modelos definidos en base a diferentes metamodelos.

1.4. Metodología de investigación

La metodología adoptada para el desarrollo de esta investigación sigue un procedimiento que incluye las siguientes etapas:

- *Estudio bibliográfico. Se realiza una amplia revisión de la literatura y normativa publicada en los últimos años sobre diversos tópicos. Los principales campos de interés que se abordarán en esta revisión bibliográfica se enumeran a continuación:*

- MBSE, los tipos de modelos habituales en el contexto de la fabricación y su integración.
- Modelado geométrico de artefactos, incluyendo las desviaciones geométricas y su propagación en sistemas multietapa.
- Modelado de sistemas de fabricación, tanto a nivel estructural como comportamental, incluyendo el modelado del control.
- Semántica propia de diferentes enfoques y herramientas de análisis sobre variabilidad, geometría, sistemas multicuerpo, etc. así como las representaciones de sistemas/dispositivos electromecánicos (embedded devices) y los tipos de modelos (matemáticos, de sistemas y ontológicos) que las han sustentado.
- Semántica propia del modelado del comportamiento de sistemas, especialmente la simulación de eventos discretos.
- Lenguajes de modelado y simulación existentes. Estudio de técnicas y metodologías para la adquisición de capacidades propias de un ingeniero de sistemas, para modelar a varios niveles, tanto a nivel de metamodelo (M2, creación y extensión de lenguajes) como modelos de aplicación (M1).
- Herramientas y entornos de modelado y simulación híbrida multidominio, seleccionando la más adecuada según unos criterios predefinidos.
- Mecanismos y lenguajes que den soporte a la transformación de modelos.

- *Desarrollo de experiencias previas*. Desarrollo de pruebas para explorar las posibilidades de diversos tipos de modelos, lenguajes y entornos de trabajo.

- *Desarrollo e implementación de la metodología propuesta*. Esta fase cuenta con su propio procedimiento, que se concreta en los siguientes pasos:
  - Definir el escenario y el propósito de la metodología, identificando el contexto de uso, las partes interesadas, y los dominios y artefactos involucrados.
  - Establecer un procedimiento para el modelado consistente de sistemas de simulación.
  - Seleccionar un conjunto de lenguajes que den soporte al procedimiento propuesto.
  - Seleccionar y poner a punto las herramientas necesarias para soportar el procedimiento propuesto.
  - Definir lenguajes específicos de dominio mediante la extensión de lenguajes más generales existentes (creación de perfiles SysML), incluyendo los conceptos y las restricciones que den soporte a la semántica específica del dominio.
  - Definir librerías de elementos de modelado reutilizables.
  - Definir reglas de transformación de modelos.

- *Evaluación*. Definición de un conjunto de casos de estudio que permitan aplicar la metodología propuesta en diferentes escenarios y analizar la capacidad de detectar inconsistencias durante el modelado. Una vez obtenidos los modelos de simulación ejecutable, analizar los resultados obtenidos y valorar la capacidad de estos modelos para analizar el sistema y comparar estrategias durante el diseño del sistema de fabricación.

1.5. Esquema de contenidos

El documento se estructura en ocho capítulos (sin incluir la presente introducción) que recogen diversos contenidos y aspectos, descritos brevemente a continuación:

- Capítulo 2: Bases de la investigación
  Revisión del estado del arte de las bases de la investigación, incluyendo los tipos de modelos presentes en el contexto de la fabricación y haciendo especial énfasis en aquellas propuestas vinculadas con el modelado de la geometría.

- Capítulo 3: Modelo de simulación híbrida: desviaciones geométricas y productividad en sistemas de fabricación.
  Modelado de simulaciones híbridas para explorar las posibilidades de algunos modelos, lenguajes y entornos. Se limita al caso de procesos de ensamble, integrando el análisis del flujo logístico y la propagación de desviaciones geométricas de acuerdo a la técnica Stream of Variation.

- Capítulo 4: Metodología para el modelado consistente de simulaciones.
  Propuesta metodológica final, explorando el uso de modelos SysML para el diseño de las simulaciones, que posteriormente serán transformados a modelos de simulación ejecutables definidos en un lenguaje apropiado.

- Capítulo 5: Especificación de lenguajes de dominio.
  Definición de los lenguajes de modelado específicos de dominio, explotando el mecanismo de creación de perfiles SysML. La propuesta incluye la definición de tres perfiles que abordan los siguientes aspectos: a) modelado de sistemas de simulación para el análisis del flujo de materiales en sistemas fabricación multietapa, abordando tanto su estructura como el comportamiento discreto característico de la fabricación por lotes; b) la definición geométrica de artefactos, incluyendo las desviaciones geométricas y aspectos propios del análisis de tolerancias; y c) modelado de sistemas de simulación para el análisis combinado del flujo de materiales y la calidad geométrica en sistemas fabricación multietapa.

- Capítulo 6: Transformaciones entre modelos
  Definición de un perfil SysML que de soporte a los principales conceptos del metamodelo de Modelica, con el fin de facilitar las transformaciones de modelos. Implementación de un modelo en lenguaje Epsilon que soporte la transformación automática de modelos SysML a modelos en formato texto Modelica.

- Capítulo 7: Definición de librerías
  Creación de librería como repositorio de elementos de modelo reutilizables, tanto en el entorno de modelado SysML como en el entorno de simulación Modelica.

- Capítulo 8: Caso de estudio
  Aplicación de la propuesta metodológica a un caso práctico, en el que se simula un sistema de ensamble multietapa con varios tipos de productos, rutas alternativas para cada producto y recursos comunes.

- Capítulo 9: Conclusiones y trabajos futuros
  Síntesis de las conclusiones generales e identificación de los trabajos futuros que extenderán esta línea de investigación.

1.6. Resultados

Además del presente documento, que recoge de forma integrada y resumida el material resultante de la investigación, estos esfuerzos han derivado también en diversas comunicaciones en congresos y artículos en revistas a través de las cuales se han publicado progresivamente los avances y aportaciones de la investigación. A pesar de que en la fecha en la que se presenta esta tesis quedan algunos artículos pendientes de publicación, a continuación, se enumeran los trabajos ya publicados:

CONGRESOS:

- Modeling Manufacturing Resources: An Ontological Approach. 15th IFIP WG International Conference, 2018. [22]
- How to Restructure PPDRC and MIRC According to DOLCE. 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2018) [23]
- Feature in product engineering with single and variant design approaches. A comparative review. 8th Manufacturing Engineering Society International Conference [24]
- Metodología basada en SysML y Modelica para la simulación de sistemas de fabricación. II Congreso Internacional Online sobre Tecnología e Ingeniería. [25]
- Control strategies comparison for a multi-stage assembly system using simulation. 9th Manufacturing Engineering Society International Conference (MESIC 2021) [26]
- A Modelica Library to Simulate Geometrical and Dimensional Deviations in Process Assemblies. 10th Manufacturing Engineering Society International Conference (MESIC 2023) [27]

ARTÍCULOS:

- Feature-Based Framework for Inspection Process Planning. MATERIALS [28]
- Revisión del estado actual y perspectivas futuras sobre el modelado de máquinas herramienta. 3C Tecnología [29]
- Multidomain simulation model for analysis of geometric variation and productivity in multi-stage assembly systems. Applied Sciences. [30]
- SysML4TA: A SysML profile for consistent tolerance analysis in a manufacturing system case application. Applied Sciences. [31]

2. Marco teórico

El presente capítulo recoge de manera resumida los aspectos fundamentales sobre los que se plantea y desarrolla la investigación presentada en este documento. Con el fin de estructurar el capítulo y organizar los diversos contenidos presentados, se identifican los siguientes campos de interés, dedicando una sección a cada uno de ellos.

- *Ingeniería de Sistemas Basada en Modelos (MBSE)*. Se trata de un enfoque holístico para la implementación de la ingeniería de sistemas que se basa en el uso de modelos, como fuente de datos, en todas las actividades de ingeniería a lo largo de todo el ciclo de vida del producto. En esta sección se analizan diversos tipos de modelos, especialmente aquellos relevantes para la investigación, y se comentan aspectos como el aseguramiento de la consistencia y la implementación de transformaciones automáticas entre modelos.

- *Modelado de artefactos técnicos*. Uno de los principales aspectos tratados en esta investigación es el modelado de la geometría (tanto nominal como desviada) de artefactos técnicos, ya sean productos, recursos de fabricación o ensambles de proceso. Por ello, en la sección dedicada a estas cuestiones se presentarán diversos enfoques y propuestas para el modelado de la geometría (especialmente las geometrías relevantes desde el punto de vista funcional) y sus desviaciones, así como para su propagación en procesos de fabricación multietapa.

- *Diseño Integrado de Producto, Proceso y Recurso*. Se trata de un enfoque que contempla la integración de los tres elementos en un supersistema, en adelante denominado "sistema PPR". Adoptando este enfoque, no solo se caracterizan los componentes, sino que se enfatiza en las interacciones entre ellos y emerge el reto de garantizar la consistencia de los modelos creados en este supersistema.

- *Evolución de los sistemas de fabricación*. Se desarrolla un breve repaso por los principales paradigmas que han marcado los grandes cambios en el diseño de sistemas de fabricación, haciendo especial mención al paradigma operativo de los sistemas reconfigurables y sus implicaciones tanto en los recursos de fabricación como, de forma especial, en el diseño y validación de los sistemas de control.

2.1. Ingeniería de sistemas basada en modelos

La creciente complejidad de los sistemas técnicos actuales requiere de metodologías que permitan abordar los desafíos que presentan el diseño, desarrollo y validación de estos sistemas. Esta tendencia ha promovido la adopción de la Ingeniería de Sistemas

basada en Modelos (MBSE), un enfoque holístico centrado en el desarrollo, gestión y control de modelos que capturen, a través de diferentes vistas (estructural, funcional, comportamental, etc.), los intereses de los diversos actores involucrados (*stakeholders*). De esta manera, el enfoque MBSE propone sustituir los procesos de ingeniería centrados en documentos por otros basados en modelos, un cambio que resulta especialmente ventajoso de cara al desarrollo de simulaciones y experimentos con modelos ejecutables, entre otras muchas aplicaciones, como se indica en [32].

Este enfoque ha sido ampliamente aplicado en múltiples campos de la ingeniería, incluidos los sistemas de fabricación y producción [33], que son el contexto principal de esta propuesta. En general, las metodologías MBSE manejan un conjunto heterogéneo de modelos, debido al manejo de conceptos de diferentes disciplinas y dominios, pertenecientes a diferentes niveles de granularidad y creados con diferentes herramientas informáticas y lenguajes. De esta heterogeneidad surgen diferentes desafíos, como son los relacionados con la consistencia y la integridad de los modelos o con las transformaciones automatizadas entre diferentes modelos, cuestiones que serán tratadas a lo largo de esta sección.

2.1.1. Conceptos básicos de modelado

Con el fin de establecer un marco de trabajo claro y concreto, es conveniente establecer una serie de definiciones de conceptos básicos sobre el modelado de sistemas que resultan fundamentales en el desarrollo de esta investigación. Algunos de estos conceptos han sido ampliamente tratados en la literatura y pueden existir diversos enfoques o formas de acercarse a ellos, de modo que resulta necesario presentar los conceptos que han sido adoptados junto con las definiciones tal y como han sido entendidas y consideradas a lo largo de esta investigación.

En primer lugar, un *sistema* es un conjunto de entidades (naturales, sociales o técnicas) que actúan e interactúan entre sí con algún fin [34]. Tal y como se representa en la Figura 2.1, es común diferenciar entre sistemas estáticos (no cambian a lo largo del tiempo) y dinámicos (cambian con el tiempo), clasificando estos segundos entre discretos (cambian de forma instantánea en determinados puntos temporales) y sistemas continuos (cambian continuamente con respecto del tiempo). En el contexto de esta investigación, los sistemas tratados son fundamentalmente sistemas técnicos discretos, es decir, artefactos ideados o ingeniados, tanto físicos como software, que presentan un comportamiento discreto, es decir, cambian en determinados instantes del tiempo. Por ejemplo, a lo largo de la investigación serán analizados diversos sistemas físicos (artefactos técnicos) presentes en el ámbito de la fabricación, como el propio producto o los equipos de fabricación (máquinas, utillajes, herramientas, etc.).

Adoptando el enfoque de sistemas, el producto y el sistema de fabricación son partes de un supersistema construido para integrar estos sistemas y describir las interacciones entre ellos, un planteamiento que será considerado en esta investigación. Frente al modelado de sistemas físicos, también es posible modelar sistemas software, como aplicaciones, bases de datos, programas de simulación. En este sentido, la presente investigación se centra en la creación de modelos (fundamentalmente descriptivos) para el diseño y desarrollo de sistemas de simulación ejecutables

(software). Así pues, el sistema de interés que es objeto de estudio puede variar en función de la etapa considerada, y será debidamente identificado en cada momento.

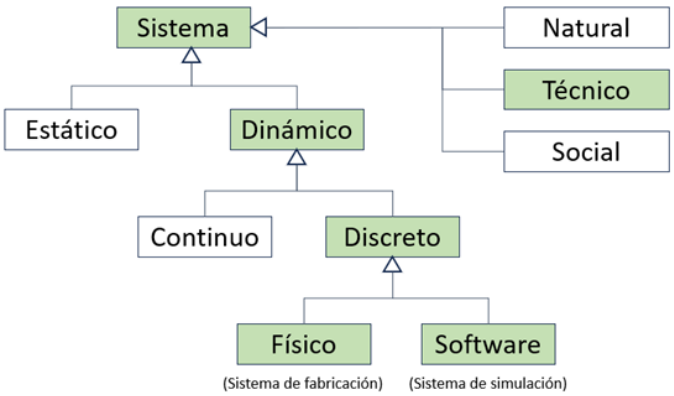


Figura 2.1 Tipos de sistema

Por su parte, un *modelo* es una representación simplificada de un sistema que es de interés, y el modelado es la actividad de construir modelos. El modelado afronta el reto de capturar de manera adecuada todos los aspectos de interés desde un determinado punto de vista y, por tanto, implica la selección de las características relevantes. Aunque existen modelos físicos (representaciones materializadas, como las maquetas), nuestra investigación aborda únicamente el desarrollo de modelos abstractos, creados para apoyar actividades tanto de diseño (modelos de especificación) como de análisis. Estos modelos abstractos están construidos haciendo uso de un determinado lenguaje de modelado. Aunque de forma general se entiende que el lenguaje más universal y puro es el lenguaje matemático, en el ámbito del modelado existe una gran variedad de lenguajes desarrollados para dar soporte a la creación de diferentes tipos de modelos en multitud de dominios. Sin embargo, una característica común a todos los lenguajes es que su especificación define la sintaxis y la semántica del lenguaje. De forma general, se entiende como sintaxis el conjunto de reglas que gobiernan la correcta construcción de modelos con dicho lenguaje, si bien en algunas referencias del ámbito del modelado como [35] proponen la diferenciación entre:

- *Sintaxis abstracta*. Define el léxico, es decir, los conceptos manejados en el lenguaje (vocabulario), así como sus propiedades y sus relaciones (taxonomía). La sintaxis abstracta suele estar definida generalmente mediante un metamodelo, en el que se definen los conceptos del lenguaje y sus relaciones.

- *Sintaxis concreta*. Define la forma en que los conceptos, propiedades y relaciones son codificados o representados en los modelos construidos con el lenguaje. Como existen lenguajes de modelado textuales (por ejemplo, Verilog o Modelica) y otros gráficos (por ejemplo, UML, UPDM, BPMN, MARTE, SoaML o IDEFx), la sintaxis concreta puede definir, según el caso, el formato y estructura del código (modelos textuales) o la simbología/notación gráfica y las reglas que rigen las representaciones gráficas (modelos con diagramas).

Por su parte, la semántica apunta al significado de los modelos y los elementos que lo forman, un aspecto vinculado con la interpretación de los conceptos y terminología empleada, y que difícilmente pueden implementarse de manera formal. Por este

motivo, en el ámbito del modelado, la semántica suele referirse a la limitación del significado haciendo uso de restricciones que generalmente limitan aspectos de naturaleza sintáctica, o recurriendo al mapeo entre conceptos de diferentes lenguajes. En ese sentido, en [35] se propone la diferenciación entre:

- *Semántica estática*. Define una serie de restricciones que deben cumplir todos los modelos y que dan soporte a las tareas de verificación, comprobando si el modelo está correctamente construido. En muchas ocasiones, estas restricciones están definidas en el metamodelo, o bien implementadas en ficheros independientes que pueden ser ejecutados sobre un modelo como parte de su verificación.

- *Semántica dinámica*. Definen el significado de los modelos de un lenguaje mapeando sus elementos a partes de otros modelos o códigos implementados en otros lenguajes. Estas relaciones son fundamentales en el desarrollo de traductores o transformaciones de modelos, estableciendo vínculos entre conceptos análogos en dos lenguajes diferentes. Cabe mencionar que en el ámbito del software el término "semántica dinámica" también se emplea para referirse a aspectos vinculados con la ejecución de modelos (semántica operacional, axiomática, etc.), pero este aspecto no será abordado en esta investigación.

Así pues, como se ha indicado anteriormente, la especificación de un lenguaje generalmente incluye la definición de un metamodelo (modelo de modelos) en el que se detalla la sintaxis abstracta y, con frecuencia, las reglas que soportan la semántica estática. La instanciación de estos metamodelos permite obtener modelos particulares definidos con dicho lenguaje. En base a estos principios, la OMG propuso la arquitectura de niveles [36] que se representa en la Figura 2.2 y se describe a continuación.

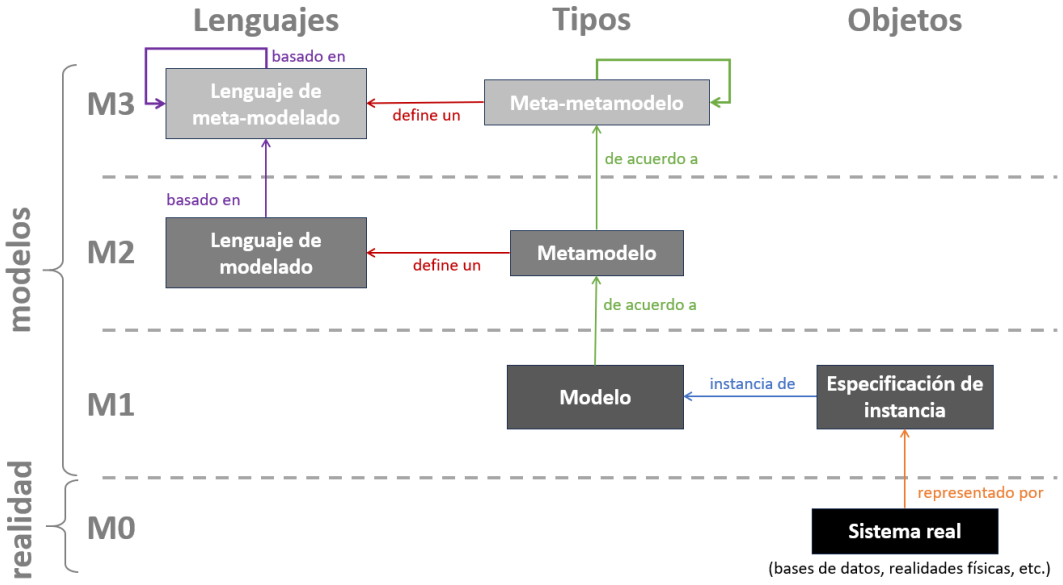


Figura 2.2. Niveles de modelado. Elaboración propia basada en [37]

Tal y como se representa en la Figura 2.2, en el nivel M0 se sitúan los sistemas reales que se pretenden representar, ya sean realidades físicas, bases de datos, etc. Estos

sistemas reales y concretos son representados en un modelo de objetos o de especificación de instancias, pertenecientes al nivel M1. Sin embargo, en este mismo nivel M1 también se pueden definir modelos (llamados en ocasiones modelos de usuario) que representan a un conjunto de elementos de un mismo tipo que comparten características estructurales y comportamentales. Así, las especificaciones de instancias son concreciones (instancias) de las clases definidas en el modelo de usuario (M1). Además, el modelo de usuario (M1) se define de acuerdo a un metamodelo (M2), que describe un determinado lenguaje. Tanto el Lenguaje como el Metamodelo que lo describe son entidades del nivel M2. Por último, se suele considerar un tercer nivel M3 para definir un Meta-metamodelo, un Meta-lenguaje y las respectivas relaciones con los elementos de nivel M2, si bien también es frecuente mostrar relaciones recursivas en los elementos de nivel M3 para soportar posibles modelos y lenguajes por encima de los indicados. Para una mayor información sobre estos niveles y las relaciones entre ellos se recomienda consultar [38].

Aunque esta arquitectura fue propuesta en el contexto del desarrollo del lenguaje UML, en el ámbito de esta investigación se considera válida para los modelos y lenguajes que se van a tratar, haciendo referencia a estos niveles a lo largo del documento. Sin embargo, más allá de esta arquitectura, esta investigación también adopta una clasificación de tipos de modelos (y los lenguajes apropiados para cada uno de ellos) muy común en la literatura, identificando dos categorías fundamentales que son los modelos descriptivos y los modelos de análisis, tal y como se representan en la Figura 2.3.

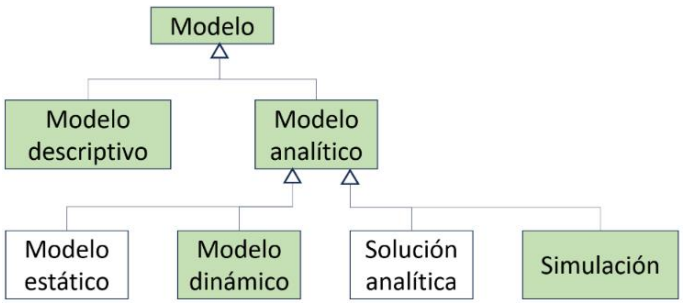


Figura 2.3. Tipos de modelos

Por una parte, los modelos descriptivos están orientados a representar formalmente un dominio o sistema, de manera que pueda ser interpretado tanto por humanos como por ordenadores. Estos modelos dan soporte a la definición de información muy variada, incluyendo aspectos de estructura, comportamiento, requerimientos, etc. Son empleados fundamentalmente en tareas de diseño, tanto en la especificación como en la integración de sistemas, facilitando la comunicación formal entre los *stakeholders* [39]. Dos ejemplos claros de esta categoría son los modelos UML y SysML.

Por otro lado, los modelos de análisis están más orientados a la cuantificación matemática de determinadas características y la representación del comportamiento esperado del sistema en base a relaciones lógicas y matemáticas. Entre los modelos de análisis se pueden diferenciar los modelos estáticos, orientados a estudiar propiedades que son independientes del tiempo (válido en cualquier instante del tiempo), y los

modelos dinámicos, en los que se estudian características que varían con el paso del tiempo. Además de la influencia del tiempo en las características analizadas, los modelos analíticos se pueden clasificar en función del tipo de solución obtenida. En el estudio de sistemas simples es frecuente definir un modelo matemático que permita obtener una solución analítica. Sin embargo, en el caso de sistemas complejos (como los estudiados en esta investigación), estos análisis deben estar soportados por modelos de simulación. Entre los lenguajes más comunes en el ámbito de la simulación se pueden citar Simulink o Modelica, entre muchos otros.

Vistas algunas generalidades y conceptos básicos sobre el modelado, el siguiente punto abordará con mayor detalle algunas metodologías basadas en el enfoque MBSE del ámbito de la fabricación, identificando algunos tipos de modelos manejados a lo largo del ciclo de vida del supersistema formado por el producto y el sistema de fabricación.

2.1.2. Metodologías MBSE en el ámbito de la fabricación

En las últimas décadas, numerosas metodologías basadas en el enfoque de sistemas han sido propuestas para abordar el proceso de diseño y desarrollo de sistemas de todo tipo, incluidos tanto sistemas mecatrónicos (como los productos o sistemas de fabricación) como sistemas software (como los sistemas de simulación). Muchas de estas metodologías se apoyan en el uso de representaciones gráficas, como son las representaciones en forma de cascada o en espiral, mediante las cuales se sintetizan algunos de sus elementos básicos y sus relaciones. Esta investigación centra el foco de atención en aquellas metodologías que adoptan una representación en V, una de las más extendidas en la ingeniería de sistemas [40]. La Figura 2.4 muestra un ejemplo de representación en V, similar a los propuestos por [41] y [42], y publicado en [25], en el que se describe gráficamente el flujo de actividades de diseño y desarrollo de un sistema, como puede ser el caso de un sistema de fabricación.

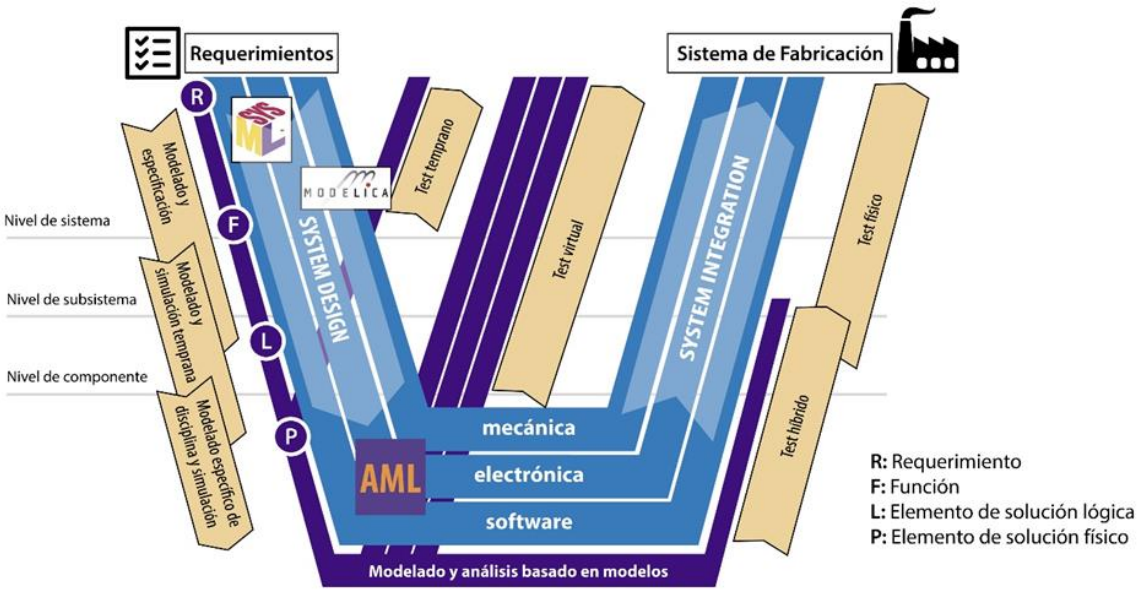


Figura 2.4. Representación en V aplicada a sistemas de fabricación [25]

En las primeras metodologías que adoptaron este tipo de representación en V únicamente se considera la parte representada en azul claro en la Figura 2.4, donde la rama descendente representa la etapa de diseño, desde el nivel de sistema al detalle de cada componente, abordando diferentes dominios. Posteriormente, los diversos componentes y dominios se integran en la rama ascendente, que incluye además la verificación y validación, a diferentes niveles de composición, en base al ensayo de prototipos físicos. Más recientemente, con la adopción del enfoque MBSE se incorporó toda una serie de actividades adicionales relacionadas con el modelado, representadas en azul oscuro en la Figura 2.4. Estas nuevas consideraciones propias del enfoque MBSE no solo afectan a las tareas de especificación del diseño, sino que también influyen, y de manera más importante, en las tareas de análisis, mayoritariamente soportadas por herramientas informáticas existentes o desarrolladas ex profeso, definidas mediante metamodelos y que manejan modelos de análisis. Así pues, con la adopción de la MBSE, las nuevas actividades vinculadas al modelado tienen una importante presencia tanto en etapas iniciales de diseño (rama descendente) como en las tareas de verificación y validación de las soluciones (ramas ascendentes). Además, la capacidad de desarrollar tareas de verificación de manera virtual (sin disponer de una solución física) permite la incorporación de nuevas ramas ascendentes en etapas tempranas, en las que los ensayos del sistema mediante simulaciones (software), de diferente nivel de detalle y alcance, adquieren una especial relevancia.

Es de especial interés para esta investigación la adopción del enfoque MBSE en etapas tempranas del diseño, tanto en el diseño a nivel de sistema como en el diseño y simulación disciplinar (mecánica, electrónica, etc.). En este sentido, la Figura 2.4, sitúa en las etapas tempranas del diseño el desarrollo de modelos descriptivos, representados con el icono de SysML. Estos modelos permiten definir un principio de solución con diferentes niveles de abstracción, detalle o granularidad, además de establecer requisitos, definir comportamientos y especificar cualquier tipo de información de manera formal para ser accesible para cualquier ingeniero involucrado en el desarrollo del sistema. Uno de los lenguajes más extendidos como estándar para este tipo de modelado descriptivo de sistemas es SysML. Dada la importancia de SysML en el ámbito de esta investigación, se dedica el punto 3.1.1 a comentar algunas de sus características fundamentales.

En lo referente a las actividades orientadas a la verificación y la validación en el diseño de sistemas mecatrónicos, estas tareas están vinculadas frecuentemente con modelos de análisis, en muchas ocasiones modelos de simulación ejecutables, ejemplificados con el logotipo del lenguaje Modelica en la Figura 2.4. En la bibliografía y en el mercado existen numerosas propuestas (lenguajes, modelos, aplicaciones, etc.) específicas que tratan de abordar el análisis de determinados aspectos de un sistema. Por poner algunos ejemplos específicos del ámbito de la fabricación, algunas aplicaciones relevantes son AutoMod, Enterprise Dynamics, FlexSim, Plant Simulation, ProModel, WITNESS, entre otros [43]. Sin embargo, el uso de estas aplicaciones específicas presenta en ocasiones importantes dificultades en la integración de la información y los modelos debido a incompatibilidades. Frente a estas aplicaciones específicas, otras más genéricas han adquirido importancia en los últimos años, convirtiéndose en estándares *de facto*. Por ejemplo, centrados en la simulación de sistemas dinámicos,

se pueden destacar algunas herramientas como Simulink, Ansys, SimMechanics en el ámbito de los sistemas contínuos, GPSS o Verilog para el análisis de eventos discretos, o OpenModelica y AnyLogic como principales herramientas que soportan simulaciones híbridas (discreta y contínua) [44]. Dada la naturaleza de los sistemas de simulación objeto de estudio en esta investigación, que abordan tanto el comportamiento discreto vinculado a los análisis de productividad, como el análisis de las desviaciones geométricas y su propagación, este trabajo se centrará en explorar las capacidades de uno de estos lenguajes, Modelica, al que se le dedica el apartado 3.1.2 de este documento.

Como se ha comentado, las metodologías que adoptan la representación en V han sido ampliamente utilizadas en el diseño y desarrollo de sistemas mecatrónicos, como por ejemplo de un producto, de un sistema de fabricación, o incluso para abordar el desarrollo integrado del producto y su sistema de fabricación, tal y como se propone en [45]. Sin embargo, en estas propuestas no se visualizaba en detalle el diseño y desarrollo de los sistemas de simulación que, como sistemas que son, también pueden representarse mediante su propia V. Esta orientación centrada en el desarrollo de los modelos de simulación multidominio fue presentada en [25] y es adoptada como eje central de esta investigación. Para ilustrar este enfoque, la Figura 2.5 muestra la representación en V del diseño y desarrollo de un sistema físico (parte izquierda de la figura), destacando las tareas de modelado y la simulación temprana y del desarrollo de los correspondientes test tempranos. En paralelo, se muestra el detalle de la representación en V aplicada al diseño y desarrollo del sistema de simulación (parte derecha de la Figura 2.5), que aborda análisis multidominio (logística, calidad geométrica, etc.) hasta llegar a elevados niveles de detalle y realismo.

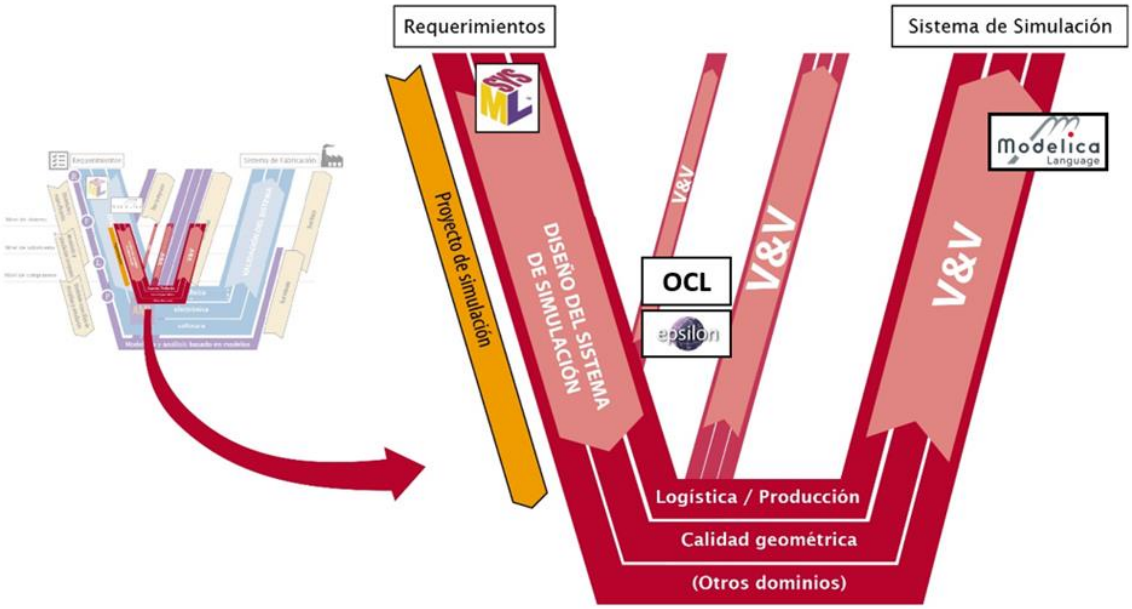


Figura 2.5. Representación en V aplicada a sistemas de simulación. Basado en [25]

De forma similar a lo comentado para el desarrollo de sistemas mecatrónicos (Figura 2.4), en el diseño de los sistemas de simulación (Figura 2.5) también consideran modelos descriptivos (modelos SysML, por ejemplo) que dan soporte a su diseño, así como tareas de verificación en etapas tempranas de su desarrollo. En este caso, la

verificación de los modelos se realiza comprobando el cumplimiento de determinadas reglas que soportan aspectos de sintaxis abstracta o de semántica estática, generalmente definidas en el propio metamodelo del lenguaje empleado. El cumplimiento de restricciones de estas reglas permite asegurar la consistencia a diferentes niveles, tanto inter-modelo (relaciones entre diferentes modelos) como intra-modelo (correcta construcción de un modelo). Sin embargo, dadas algunas limitaciones de los lenguajes propios de los sistemas de simulación, en ocasiones esta gestión de la consistencia debe estar soportada en la verificación temprana de los modelos descriptivos, por ejemplo, implementando reglas OCL (Object Constraint Language) que pueden ser comprobadas en los modelos SysML. Estos aspectos serán tratados con mayor detalle en el siguiente apartado.

2.1.3. Consistencia y transformaciones de modelos

La creciente complejidad de los sistemas técnicos y consecuente adopción del enfoque MBSE se ha traducido en la necesidad de manejar un conjunto heterogéneo de modelos para gestionar los conceptos pertenecientes a diversas disciplinas y dominios. Estos modelos pueden presentar diferentes niveles de granularidad y haber sido creados con una gran variedad de lenguajes y herramientas, además de contener una gran cantidad de elementos, y relaciones o dependencias entre partes del modelo que representan diferentes puntos de vista. Fruto de esta complejidad, resulta difícil gestionar los modelos, detectando y corrigiendo posibles errores. Estos errores son conocidos como inconsistencias, un término que surgió inicialmente en la ingeniería de software y que se ha extendido a otros ámbitos de la ingeniería basada en modelos. Así pues, de la complejidad y la heterogeneidad de los modelos manejados surgen diferentes desafíos, por ejemplo, los relacionados con la coherencia y la integridad de los modelos, así como con las transformaciones automatizadas entre diferentes modelos.

Ante la necesidad de diagnosticar y gestionar las inconsistencias continuamente, la gestión de los modelos requiere de un chequeo automatizado, dado que la comprobación manual de la consistencia es una tarea compleja, que consume mucho tiempo y con un riesgo de error elevado. Este chequeo debe comprobar la consistencia tanto entre las construcciones o vistas de un modelo (consistencia intra-modelo) como entre diversos modelos (consistencia inter-modelo) desarrollados típicamente en los sistemas multidisciplinares. Para ello, numerosos enfoques y herramientas fueron propuestos inicialmente en el ámbito del software para gestionar la consistencia de los modelos y, en la medida de lo posible, evitar o detectar inconsistencias.

Algunas de estas propuestas se centran en prevenir o limitar el riesgo de introducir inconsistencias en los modelos, por ejemplo, creando un modelo central en el que almacenar toda la información de interés, accesible y utilizada por otros modelos, pero sin capacidad de modificarla y, por tanto, limitando el riesgo de introducir nuevos errores. Sin embargo, este tipo de estrategias preventivas no son suficientes y es necesario considerar mecanismos que comprueben la consistencia del modelo durante su desarrollo mediante el chequeo de determinadas condiciones o reglas que deben cumplirse.

En este sentido, gran parte de estas propuestas identifican la adecuada elección del lenguaje de modelado como uno de los factores clave para identificar inconsistencias, asumiendo que los metamodelos de los lenguajes más habituales (como UML) ya son consistentes y que cuentan con reglas que describen las condiciones que un modelo debe satisfacer para que se considere un modelo válido (por ejemplo, buena formación sintáctica, coherencia entre diferentes diagramas e incluso coherencia entre diferentes modelos). Por tanto, en este tipo de estrategias las inconsistencias se detectan cuando no se cumple alguna de las reglas definidas a nivel de metamodelo (semántica estática). Sin embargo, esta estrategia no es válida para comprobar la consistencia entre modelos (inter-modelo) definidos con diferentes lenguajes. Para ello, es necesario capturar la semántica común y establecer relaciones entre los metamodelos (semántica dinámica).

Aunque este tema ha sido ampliamente estudiado en el dominio del software, en los últimos años se ha vuelto más relevante en el diseño de otros sistemas complejos, como los mecatrónicos. De hecho, numerosos autores aseguran que el aseguramiento de la consistencia es uno de los principales retos de la industria en la actualidad [46], tanto para la creación de modelos individuales como para la integración de modelos. En este ámbito, diferentes cuestiones han ganado relevancia en los últimos años, como la verificación de consistencia entre modelos de diferentes dominios [47], la gestión de la consistencia en sistemas automatizados de producción manufacturera [48], el modelado de dependencias entre modelos en el diseño mecatrónico [49], etc.

Una de las líneas de investigación más interesantes es la centrada en la integración de modelos descriptivos y de análisis, estableciendo relaciones que minimicen la actual separación entre el modelado de arquitecturas funcionales (tradicionalmente soportado por modelos descriptivos) y el modelado de comportamiento físico (más vinculado a modelos de análisis), una línea de trabajo especialmente importante en el uso de MBSE para el desarrollo mecánico, como se comenta en [50] y se aborda también en [51] y [52].

Esta necesidad de integración adquiere mayor relevancia al considerar que los lenguajes típicos para el modelado de simulaciones carecen de mecanismos para soportar la semántica específica de un dominio, una cuestión fundamental en el aseguramiento de la consistencia de los modelos. Por ejemplo, en los modelos de análisis de tolerancia, se debe validar la consistencia considerando conceptos relacionados con la geometría de los artefactos y su definición matemática, tanto para geometrías individuales como para sus relaciones. Aunque la sintaxis y la semántica de los lenguajes de simulación (como Modelica) están perfectamente definidas, no poseen mecanismos suficientes para validar la consistencia en base a semánticas específicas de dominio. Como alternativa, algunos lenguajes descriptivos como UML y SysML incorporan estas capacidades a través del mecanismo de creación de perfiles, que permiten ampliar el metamodelo incorporando conceptos (sintaxis abstracta) y definir restricciones que limiten la semántica estática de los nuevos conceptos incorporados, por ejemplo, mediante la definición de reglas implementadas en lenguaje OCL.

Ante esta limitación de los modelos de análisis, surge la necesidad de diseñar los sistemas de simulación mediante modelos descriptivos en los que poder aplicar

mecanismos para el aseguramiento de la consistencia y la consideración de la semántica estática propia de los dominios específicos bajo estudio. En esta línea, surge la posibilidad de crear lenguajes específicos de dominio que incorporen reglas OCL que den soporte a diversas comprobaciones de carácter sintáctico y semántico, como se propone en [35]. Estas expresiones OCL permiten navegar por los modelos, realizar consultas y realizar comprobaciones lógicas, sin efectos colaterales, es decir, que no se altera el modelo. Este tipo de expresiones se emplean, por ejemplo, en la propia especificación de lenguajes como UML o SysML, y puede definirse igualmente en el desarrollo de lenguajes específicos de dominio que incorporen, mediante las respectivas reglas OCL, la semántica estática propia del dominio de interés.

Ahora bien, si el diseño de los sistemas de simulación se realiza mediante modelos descriptivos en los que gestionar la consistencia con los mecanismos oportunos, otro asunto igualmente importante para el aseguramiento de la consistencia es la adecuada automatización de las transformaciones de los modelos descriptivos de los sistemas de simulación a los correspondientes modelos de análisis ejecutables. Esta transformación de modelos, descritos en lenguajes diferentes, es una tarea en la que el riesgo de introducir inconsistencias y errores es elevado. Por tanto, la automatización de estas transformaciones no sólo es una solución interesante desde un punto de vista de reducir el tiempo consumido, sino que también es un mecanismo adecuado para limitar o, en el mejor de los casos, eliminar la posibilidad de introducir inconsistencias en el modelo resultante.

2.2. Modelado de artefactos técnicos: funcionalidad y geometría

Vistas algunas generalidades sobre el modelado de sistemas, esta sección se centra en el modelado geométrico de artefactos técnicos. Este tipo de modelado es especialmente relevante en el diseño funcional de cualquier tipo de producto (diseño de ensambles de producto), pero también en el diseño funcional de los amarres (diseño de ensambles de proceso), estableciendo los requisitos geométricos, especificaciones y condiciones de uso (configuraciones, ajustes y/o correcciones en posicionamientos, etc.) de los recursos de fabricación necesarios para satisfacer las especificaciones del producto a fabricar. Desde la perspectiva de la fabricación, la geometría del producto (piezas y ensambles) se obtiene como resultado de una sucesión de procesos (de conformado o ensamblaje), descritos en un plan de procesos que generalmente cuenta con diversas etapas. Por lo tanto, las características geométricas del producto se van generando o modificando a lo largo de estos procesos de transformación (etapas), de manera que el producto va pasando por un conjunto de estados en los que la geometría resultante de una determinada etapa depende de la del anterior y de las interacciones con los recursos de fabricación de dicha etapa.

Para poder describir y analizar este tipo aspectos geométricos e interacciones físicas es necesario disponer de modelos no en tareas de especificación (modelos descriptivos fundamentalmente), sino también en tareas de análisis de la geometría y las tolerancias, y su influencia en determinadas condiciones funcionales, por ejemplo,

mediante el análisis de cadenas funcionales en ensambles. En este sentido, un elemento clave es el ensamble de proceso, es decir, un artefacto compuesto por el producto y diferentes recursos (máquinas, herramientas, utillajes, etc.), cuyo análisis permite determinar las desviaciones producidas en la geometría del producto en una determinada etapa del proceso de fabricación. Sin embargo, además de abordar el análisis a nivel unietapa, también es necesario definir modelos que permitan estudiar cómo los rasgos geométricos y sus desviaciones van creándose o variando a lo largo de los procesos multietapa.

Con el objetivo de acercar al lector a algunos conceptos básicos sobre todas estas cuestiones, esta sección se divide en tres apartados. El primer apartado se centra en resumir algunos conceptos básicos sobre geometría, tanto nominal como desviada, y el toleranciado. Seguidamente, el segundo apartado presenta de forma resumida algunas propuestas que abordan el modelado de artefactos físicos, incluyendo la definición matemática de superficies y zonas de tolerancia. Finalmente, el tercer y último apartado se centra en el análisis de las desviaciones geométricas en el contexto de procesos multietapa.

2.2.1. Geometría nominal y desviada

El principal objetivo de los procesos de fabricación es realizar transformaciones en los materiales de partida para lograr un producto final cuya geometría (real) dé cumplimiento a unas especificaciones definidas en el diseño del producto. Sin embargo, como es bien sabido, los procesos de fabricación no son perfectos, lo que da lugar a la definición de un axioma que establece la naturaleza imprecisa de la fabricación: “todos los procesos de fabricación son inherentemente imprecisos y producen piezas que varían” [53]. Por tanto, para comprobar que las piezas producidas y sus variaciones cumplen las especificaciones establecidas, los productos son sometidos a procesos en los que se miden determinadas características clave, tanto para comparar los resultados de la medición con las especificaciones y determinar su aceptación o rechazo (inspección), como para emplear dichas medidas como referencia para tomar decisiones y calcular acciones correctivas (acciones de control). Sin embargo, en relación a los procesos de medida surge un segundo axioma que se refiere a la incertidumbre: “ninguna medida puede ser absolutamente precisa y en cada medición existe una incertidumbre finita respecto el atributo o valor medido” [53].

Como consecuencia de estos dos axiomas básicos del ámbito de la fabricación, se puede deducir que no es posible producir ni medir productos con las dimensiones teóricas (geometría nominal), por lo que la especificación de un producto debe incluir tanto la definición de su geometría nominal (ideal) como de unos valores que limiten las variaciones o desviaciones aceptables de cada característica geométrica. Estos límites a las variaciones admisibles se conocen comúnmente como tolerancias y los ingenieros deben establecerlas considerando las influencias de dicha variabilidad sobre los requisitos funcionales que debe cumplir el producto. Con esta orientación surgió el dimensionado geométrico y el toleranciado (GD&T) [54], que es una disciplina que permite establecer cuáles son las características dimensionales y geométricas clave que

influyen en la funcionalidad y montaje del producto, así como limitar sus desviaciones máximas admisibles mediante la definición de tolerancias.

El desarrollo de la disciplina GD&T tiene su origen en los inicios del siglo XX, impulsado por la necesidad de diseñar y producir componentes que pudieran ser fácilmente ensamblados e intercambiables. Sin embargo, fue a mediados del siglo XX cuando el GD&T adquirió mayor protagonismo, dando lugar a la definición de diversas normas, tanto en las agencias americanas como en la organización internacional de estandarización (ISO), con el fin de formalizar procedimientos y lenguajes para estandarizar estas tareas. Una de las normas más extendidas y adoptadas es la GPS (Geometrical Product Specification) [55], que establece el lenguaje internacional para expresar tolerancias en la documentación técnica, sirviendo como estándar que integra aspectos tanto de la especificación como de fabricación y la inspección. Por tanto, GPS no solo define una simbología (sintaxis concreta), sino que establece todo un conjunto de conceptos, principios y operaciones vinculadas al toleranciado, aplicables a diversas etapas del ciclo de vida del producto.

En este punto, resulta necesario recordar algunos aspectos fundamentales, tanto de la geometría nominal como la desviada. Por una parte, la geometría nominal se refiere a la forma ideal del artefacto diseñado, representada habitualmente en los modelos de piel de los sistemas CAD. Sin embargo, como ya se ha comentado, los sistemas y procesos de fabricación y de inspección no son perfectos, por lo que no se puede fabricar un artefacto físico que materialice de forma exacta la geometría nominal. Por este motivo, la especificación de un determinado artefacto físico debe incluir la definición de una geometría tolerada, que describe los límites de la variabilidad admisible en la geometría del artefacto real. En muchos casos, estas especificaciones se definen estableciendo relaciones entre las diversas geometrías, limitando por ejemplo la orientación y/o posición de una determinada superficie respecto de otras que se utilizan como referencia. Por tanto, además de modelar los diferentes tipos de geometría, un aspecto fundamental será establecer las relaciones adecuadas que soporten su especificación.

Dejando momentáneamente a un lado la definición a nivel de especificación y abordando la representación de la geometría real, con defectos, es conveniente diferenciar entre los errores de forma frente a los de posición/orientación. Aunque todas las geometrías reales cuentan con desviaciones de los tres tipos, en algunos análisis es frecuente trabajar únicamente con geometrías ideales (sin errores de forma), que son simplificaciones calculadas a partir de la geometría real aplicando ciertos procedimientos de ajuste. A este respecto, GPS permite representar tanto la geometría extraída (definida como un conjunto finito de puntos medidos sobre la superficie real, incluyendo los errores de forma) y la asociada (geometría ideal definida a partir de modelos de superficie no ideales a través de la operación de asociación). Complementariamente, otra clasificación diferencia entre las superficies integrales, que pertenecen al modelo de superficie, frente a los elementos derivados, que son geometrías que no existen físicamente en la pieza, sino que derivan de superficies integrales. Ejemplos típicos de elementos derivados son, por ejemplo, el centro de una esfera, el eje de un cilindro o el plano medio de una ranura.

A modo de ejemplo visual, la Figura 2.6 representa gráficamente algunos de los conceptos comentados, tomando como base la representación de una pieza cilíndrica.

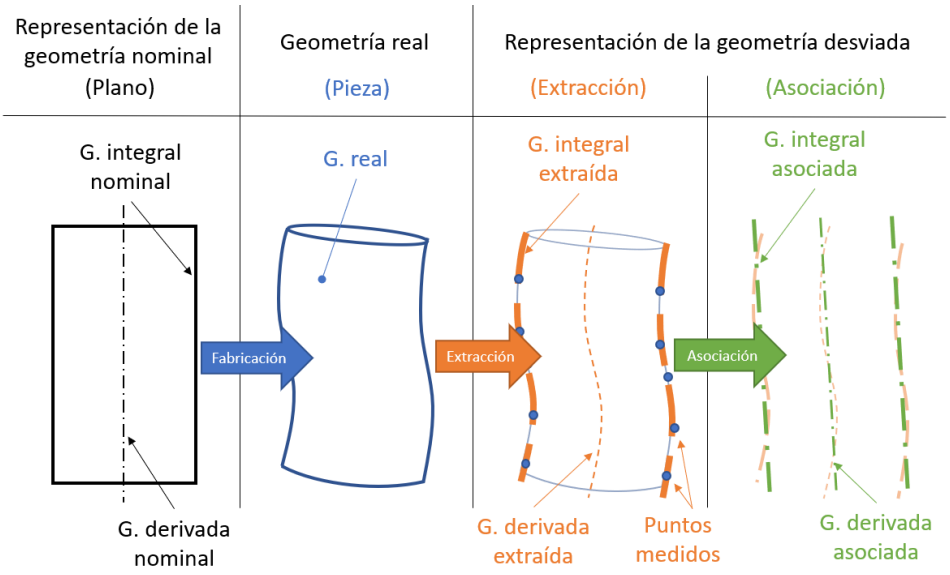


Figura 2.6. Representación de la geometría nominal y desviada. Basado en [16]

Así pues, aclarada parte de la terminología vinculada con el modelado geométrico, es momento de establecer la relación entre la especificación (geometría nominal y tolerada) y la geometría real, que determinará la aceptación o rechazo de la pieza. La geometría tolerada puede expresarse de diferentes maneras, pero la más común es la propuesta en GPS, que se basa en el concepto de "zona de tolerancia". Una zona de tolerancia es un espacio definido para representar los límites de variabilidad admisible para una determinada geometría que se pretende controlar, ya sea una superficie integral de la pieza o un elemento derivado. La zona de tolerancia tiene una determinada forma y tamaño, y generalmente con una posición y orientación definidas respecto de otras superficies de referencia, aunque hay casos en los que algún aspecto puede no estar restringido. En el caso de los procesos de inspección reales, se entiende que la especificación se cumple cuando la geometría controlada del artefacto físico queda contenida en la zona de tolerancia, para lo cual se realizan las mediciones y cálculos oportunos que permitan comprobar el cumplimiento de la especificación. En cambio, en los modelos matemáticos de análisis, como las simulaciones, es frecuente modelar la geometría real definiendo sus desviaciones respecto a la definición nominal, y almacenando dicha información en construcciones como los vectores de movimientos diferenciales (Differential Motion Vector - DMV) o torsores de pequeños desplazamientos (Small Displacement Torsor - SDT). Según esta orientación, las zonas de tolerancia limitan los valores que pueden adoptar estas desviaciones para dar cumplimiento a la especificación analizada, un enfoque que será de gran interés a la hora de desarrollar la propuesta.

En relación a los tipos de tolerancia considerados tradicionalmente en los estándares, la Figura 2.7 muestra una clasificación típica a modo de diagrama, empleando la notación de especialización típica de lenguajes como UML.

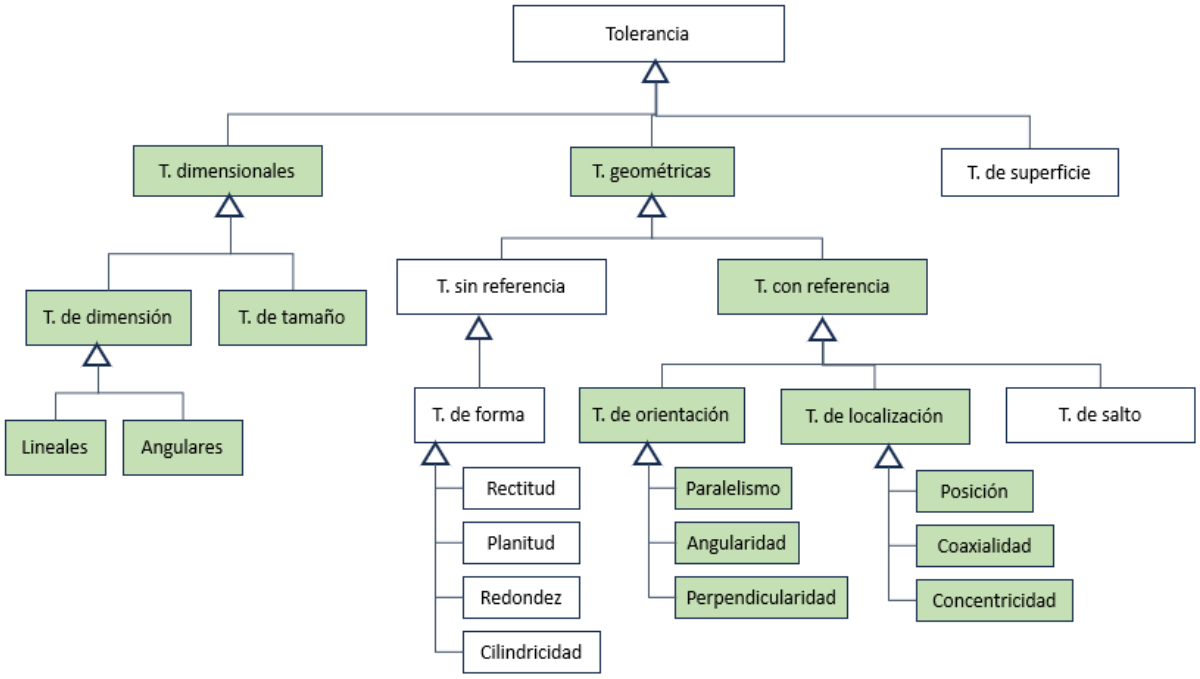


Figura 2.7. Clasificación de tolerancias según GPS.

Como se muestra en la Figura 2.7, las tolerancias se clasifican en tres tipos: tolerancias dimensionales, geométricas y de superficie. Además, las tolerancias geométricas se dividen en tolerancias con y sin referencia, en función de si la construcción de la zona de tolerancia depende de otras geometrías que funcionan con referencia. Por ejemplo, la zona de tolerancia construida para la verificación de una especificación de rectitud, de paralelismo y de posición de un plano es, en los tres casos, una zona de tolerancia plana definida por dos planos paralelos entre sí, separados una distancia definida en el tamaño de la tolerancia. En el caso de la planitud, la especificación se cumple si existe una zona de tolerancia plana del tamaño definido (sin importar su orientación ni posición) que contenga todos los puntos medidos del plano tolerado. En cambio, en el caso del paralelismo entre planos dicha zona de tolerancia ha de ser perfectamente paralela a una referencia definida, que debe ser una geometría de tipo plano. En este caso, se restringe la orientación de la zona de tolerancia, pero queda sin definir su posición. En caso de limitar también la distancia a la que se construye la zona de tolerancia respecto de la referencia se estaría definiendo una tolerancia de posición de un plano respecto de otro plano. La Figura 2.8 representa gráficamente estos tres casos.

Cabe indicar que el desarrollo de esta investigación se centra en el modelado matemático de geometrías y que, por tanto, se limita al tratamiento de geometrías sin errores de forma, que pueden presentar defectos dimensionales y angulares respecto su definición nominal y que pueden expresarse con una ecuación simple. Esto sería equivalente a decir que para representar la geometría real se emplea la geometría asociada, ya sean geometrías integrales o derivadas. A este tipo de geometrías son aplicables tanto las tolerancias dimensionales como las tolerancias geométricas de orientación y localización. Quedan por tanto excluidos los errores de forma y, por tanto, las tolerancias geométricas que los controlan (tolerancias geométricas sin

referencia y de salto). La Figura 2.7 representa visualmente este criterio mediante el coloreado de los casos contemplados en la investigación.

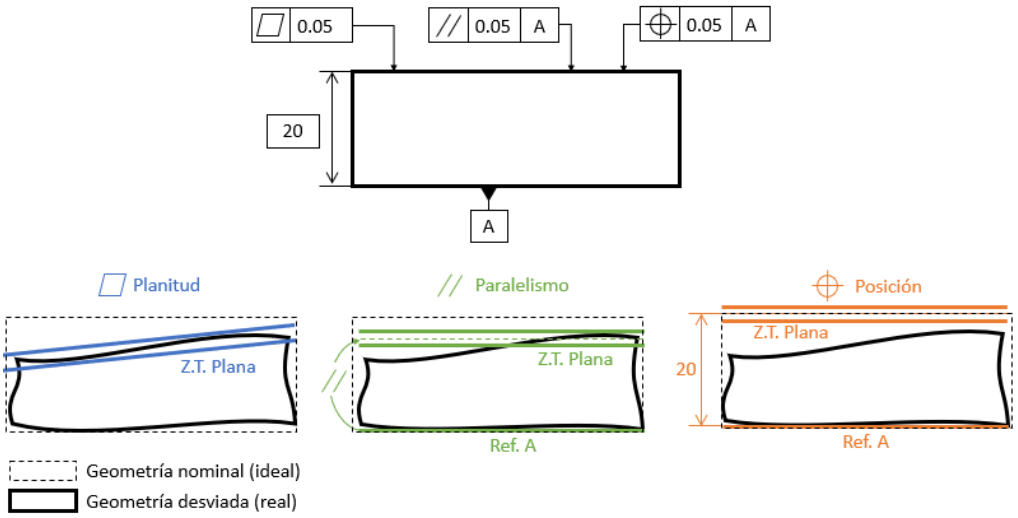


Figura 2.8. Representación de tolerancias de planitud, paralelismo y posición aplicadas sobre un plano.

2.2.2. Modelado de geometría y tolerancias

Más allá de las normas y estándares vinculados al GD&T, en la literatura existen numerosas propuestas relacionadas con el modelado de artefactos físicos que, de una manera u otra, incorporan aspectos y conceptos geométricos y, en ocasiones, también cuestiones sobre toleranciado. Sin embargo, conviene diferenciar entre aquellas propuestas que abordan el modelado de la geometría desde una perspectiva funcional, frente a aquellas otras propuestas orientadas al análisis geométrico con una base matemática. A continuación, se comenta brevemente cada uno de estos enfoques y algunas de las propuestas más extendidas en cada caso.

Las propuestas que se enfocan en el análisis funcional de artefactos físicos dan soporte a la identificación de sus superficies o rasgos geométricos del artefacto, permitiendo además establecer relaciones entre éstos. Por lo general, este tipo de propuestas carecen de una caracterización o definición matemática de las superficies, si bien pueden ser utilizadas para la definición de cadenas funcionales, construidas a partir de las geometrías y relaciones definidas y vinculadas a ciertas funcionalidades del producto. Un ejemplo de este tipo de modelos es el Open Assembly Model (OAM) [56], que establece una serie de conceptos para la definición de artefactos (piezas y ensambles), dedicando especial atención a los conceptos que dan soporte al modelado de ensambles, identificando los features y las asociaciones que permiten establecer relaciones de ensamble. OAM es un modelo basado en el modelo Core Product Model (CPM) [57], en el que se establecen los conceptos básicos sobre la estructura de los artefactos y la vinculación de los aspectos geométricos y de forma con la funcionalidad. Conviene apuntar además que OAM incluye una taxonomía de tolerancias, si bien no llega a definir matemáticamente sus características. Sin embargo, existen algunas propuestas que tratan de extender el modelo OAM para enriquecerlo, como es el caso del Modified OAM [58], en el que se incorpora por ejemplo el concepto de sistema de

coordenadas, aplicable tanto a los artefactos como a cada uno de los rasgos característicos (features) y con una definición matemática básica.

También vinculado al análisis funcional, es interesante mencionar el modelo Contact and Channel Approach (C&C2-A) [59], una propuesta que permite analizar el principio de solución a través del modelado de superficies activas y sus relaciones, tanto entre superficies de una misma pieza (concepto de "Channel and Support Structures" o CSS) como entre superficies de piezas diferentes entre las que existe una interacción (concepto de "Working Surface Pair" o WSP). A pesar de que estos modelos tampoco soportan una definición detallada de la geometría, sí aportan una visión interesante a la hora de identificar las superficies y las relaciones que funcionalmente son relevantes, además de establecer otros conceptos básicos necesarios para el modelado de artefactos.

Sin embargo, atendiendo a los intereses de esta investigación, serán especialmente relevantes los modelos orientados al análisis que sí admiten una definición matemática de la geometría. Estos modelos no solo soportan la representación tridimensional de la geometría, tan necesaria en los entornos CAD, sino que además ofrecen los recursos matemáticos necesarios para desarrollar análisis y simulaciones sobre la misma. Aunque existen múltiples alternativas para definir espacios y sistemas con diferentes características, este apartado se limita al análisis de espacios Euclídeos (de dos o tres dimensiones), en los que se definen uno o más sistemas de referencia. Esos sistemas de referencia son definidos generalmente a partir de un conjunto de vectores ortogonales que se emplean para definir matemáticamente elementos geométricos. Sin embargo, conviene diferenciar entre los sistemas de referencia globales y locales. Los sistemas locales son particulares para cada una de las piezas consideradas y se emplean para definir su geometría en base a su especificación. En cambio, a la hora de modelar ensambles, cada pieza se reposiciona en el espacio, siendo necesario definir un sistema global respecto del cual definir la nueva posición y orientación de cada pieza, o lo que es lo mismo, de su sistema local. En este tipo de modelos matemáticos, las matrices de transformación son un elemento clave para convertir la información expresada respecto de un sistema de referencia y definirla respecto de otro sistema de referencia. Estos principios básicos, junto con la definición matemática de geometrías simples, dan soporte al análisis de procesos de ensamble 2D presentado en el Capítulo 3, en el que se describen los primeros modelos de simulación desarrollados.

Durante la investigación se han explorado diversos modelos centrados en la caracterización matemática de la geometría atendiendo a diversos criterios. Una de estas propuestas es el modelo Technological and Topological Related Surfaces (TTRS) [60], que caracteriza las superficies determinando el tipo de elementos geométricos necesarios para definir matemáticamente una superficie en función de su clase de invarianza, y que además identifica el tipo de relaciones posibles entre las distintas clases de superficies consideradas. A pesar de que originariamente este modelo fue propuesto para modelar superficies y relaciones de una misma pieza, posteriormente se ha aplicado también a las relaciones de ensamble entre superficies de piezas diferentes, dando lugar al concepto de PseudoTTRS. El modelo TTRS ha sido un modelo ampliamente utilizado en el ámbito de la geometría nominal, pero también ha

servido como base para algunas propuestas orientadas al modelado matemático de tolerancias, como es el caso de [14] y [61].

Centrando el discurso en las propuestas orientadas al modelado de información sobre las tolerancias, en [12] se propone una clasificación que diferencia tres tipos de modelos: modelos de presentación, modelos de interpretación y modelos de representación. Los modelos de presentación son aquellos que solo los humanos pueden leer y entender en base a un estándar, como es el caso de GPS. Establecen generalmente una notación gráfica interpretable por expertos, pero que no puede ser entendida por ordenadores, lo cual imposibilita su uso en aplicaciones de toleranciado (Computer-Aided Tolerancing - CAT), en las que se requiere de una semántica explícita e interpretable por los ordenadores. Por su parte, los modelos de interpretación definen expresiones matemáticas con las que establecer el significado los modelos de presentación de una forma rigurosa y sin ambigüedades pero que no pueden ser leídas ni entendidas por ordenadores de forma directa. Es el caso, por ejemplo, de los modelos paramétricos, modelos de superficies variacionales, de grados de libertad o los conocidos como T-maps [12]. Por último, los modelos de representación incorporan la capacidad de ser implementados computacionalmente, como es el caso del modelo EXPRESS [62], lo cual permite el desarrollo de técnicas y herramientas asistidas por ordenador para la representación y análisis de desviaciones geométricas [63]. Por tanto, serán los modelos de representación el principal foco de esta investigación, si bien este tipo de modelos se pueden implementar en diversos lenguajes y con diversos enfoques. Por ejemplo, en [64] se presenta una revisión en la que se identifican hasta 10 alternativas, incluyendo los modelos TTRS y los modelos UML, entre otros.

Además de los modelos de alto nivel que establecen los conceptos para este tipo de análisis, existen otros modelos matemáticos y métodos de resolución que dan soporte al diseño de tolerancias, tanto de composición como de síntesis o distribución de tolerancias, como el peor caso (modelo determinista), la raíz de la suma de cuadrados (modelo estadístico) o las simulaciones Monte Carlo. Estos modelos establecen los mecanismos y expresiones matemáticas que permiten resolver cadenas funcionales predefinidas, pero carecen de una semántica definida explícitamente e interpretable por los ordenadores, lo que los acerca a los anteriormente citados modelos de interpretación.

Por último, cabe destacar que, aunque los modelos comentados en este apartado permiten modelar la geometría de cualquier tipo de artefacto físico, se intuye una tendencia natural a pensar en primera instancia en el modelado de productos. Sin embargo, las características geométricas del producto (incluidas sus desviaciones) son el resultado de la interacción física con otros sistemas, como son los recursos de fabricación (máquinas, herramientas, utillajes, etc.), en determinadas etapas en las que interseccionan sus respectivos ciclos de vida. Por tanto, en esta investigación será especialmente relevante la definición de ensambles de proceso, un término empleado para referirse al artefacto que representa un amarre, compuesto por el producto y los recursos, estableciendo entre ellos relaciones. Este ensamble de proceso dará soporte al análisis de las desviaciones producidas en las operaciones de fabricación desarrolladas en un mismo amarre. Sin embargo, la mayoría de los productos objeto

de análisis no son fabricados en una única etapa o un único amarre. Por este motivo, además de analizar matemáticamente las desviaciones propias de un ensamble de proceso, también es necesario modelar y analizar la propagación de dichas desviaciones a lo largo de las diversas etapas de un proceso de fabricación, una cuestión que será tratada con mayor detalle en el siguiente apartado.

2.2.3. Desviaciones geométricas en procesos multietapa

Como ya se ha comentado anteriormente, el producto se obtiene como resultado de una sucesión de procesos (de conformado o ensamblaje) que generan o modifican su geometría, de manera que el producto va pasando por un conjunto de estados en los que la geometría resultante de una determinada etapa depende de la del anterior y de las interacciones con los recursos de fabricación de dicha etapa. El análisis de un determinado ensamble de proceso permite desarrollar el análisis de una o más operaciones de fabricación siempre que se den en un único amarre, es decir, localizando el producto de manera única y analizando su interacción con una o más herramientas. Sin embargo, en procesos de fabricación multietapa resulta fundamental definir un tipo de análisis que permita considerar las desviaciones introducidas en cada etapa y cómo éstas se van acumulando y propagando a lo largo del proceso de fabricación.

En este sentido, una de las soluciones más extendidas es la adopción del llamado Modelo de Espacio de Estados, comúnmente utilizado en el campo de los sistemas de control [65]. El modelo de espacio de estados es un modelo matemático de un sistema físico especificado como un conjunto de entradas, salidas y variables relacionadas por ecuaciones diferenciales de primer orden (sin incluir segundas derivadas). Estas variables, llamadas variables de estado, evolucionan con el tiempo, de una manera que depende de los valores que tenía en un instante determinado (anterior) y de los valores impuestos por factores externos. Los valores de las variables de salida dependen de los valores de las variables de estado. Cabe mencionar que este modelo da soporte tanto a análisis en los que se hace un tratamiento continuo del tiempo, como aquellos en los que se discretiza en una serie de estados. Además, los términos (matrices generalmente) que modifican las variables de estado pueden ser constantes (invariantes) o depender del instante temporal considerado. En el caso de la simulación de los sistemas de fabricación que se pretenden analizar, se trata de un sistema discreto (formado por k etapas) y variante, ya que las modificaciones que se realizan sobre las variables de estado (que representan el producto) son diferentes en cada etapa. Así, la expresión general del modelo de espacio de estados para sistemas discretos y variantes en el tiempo es la siguiente.

$$\begin{aligned}\mathbf{x}(k+1) &= \mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u}(k)\\ \mathbf{y}(k) &= \mathbf{C}(k)\mathbf{x}(k) + \mathbf{D}(k)\mathbf{u}(k)\end{aligned} \tag{1}$$

Basado en el modelo de espacio de estados, una de las propuestas más extendidas para el tratamiento de las desviaciones en sistemas de fabricación multietapa es el modelo Stream of Variation (SoV), propuesto en [66] y aplicado [67] para procesos de ensamblaje 2D de múltiples etapas (procesos de ensamblaje de chapa) y más tarde

extendido para procesos de ensamblaje y mecanizado 3D de múltiples etapas [20] [21]. El modelo SoV se ha aplicado con éxito en una gran cantidad de aplicaciones, especialmente para el aseguramiento de la calidad del producto [68], la planificación de procesos [19], el diagnóstico de fallos [69], y las aplicaciones para la toma de decisiones y la optimización para la planificación de la inspección [70] [71], control de calidad y compensación de errores con el uso de datos de taller en tiempo real [72] [73]. Debido a su consolidación en este tipo de análisis, esta investigación adopta la técnica SoV en diversas etapas del desarrollo de la propuesta, motivo por el cual a continuación se presentan de forma resumida sus principales fundamentos. El modelo SoV es una técnica capaz de estimar la propagación del error a lo largo de un sistema de fabricación multietapa modelando una serie de relaciones cinemáticas entre utillajes, superficies de referencia del producto utilizadas para ubicarlo en el utillaje, y la desviación propia de la operación de fabricación (soldadura, mecanizado, etc.). Adoptando el modelo de espacio de estados para modelos discretos y variantes, el modelo SoV [66] se expresa como:

$$\boldsymbol{X_k = A_k \cdot X_{k-1} + B_k \cdot U_k + W_k,} \tag{2}$$

donde X_k es el vector que representa las desviaciones geométricas de calidad del producto después de la etapa de fabricación k; X_{k-1} representa las mismas desviaciones pero después de la etapa k-1, que se convierte en la entrada de la etapa k; U_k representa las desviaciones propias del recurso de fabricación (desviaciones de las herramientas, de los ejes móviles, etc.); las matrices A_k y B_k se obtienen mediante análisis cinemáticos y representan, respectivamente, el efecto de reorientación del producto entrante cuando se amarra en la etapa k, y los efectos de las desviaciones de los recursos sobre las desviaciones de calidad del producto resultantes en la etapa k; y W_k representa los errores de modelado, ya que las matrices A_k y B_k se obtienen mediante análisis cinemáticos sometidos a un cierto grado de linealización.

En relación a las desviaciones de calidad geométrica del producto, conviene indicar que el término X_k puede contener datos diferentes en función del análisis propuesto. Por ejemplo, en el análisis de procesos de ensamble 2D desarrollado en el Capítulo 3, este vector define la desviación del sistema de coordenadas local de cada una de las piezas que conforman el ensamble producto, ya que las piezas se suponen rígidas. Sin embargo, en análisis de procesos de mecanizado de piezas, el término X_k debe almacenar la desviación de cada una de las superficies analizadas que, de forma general, pertenecerán a una misma pieza. En los dos análisis mencionados las desviaciones se modelan mediante uno o más vectores de movimiento diferencial (Differential Motion Vector - DMV) en la forma [d, θ], donde d y θ son respectivamente la desviación de posición y orientación respecto de la definición nominal. Obviamente, el tamaño de este vector dependerá de las dimensiones consideradas en el análisis, considerando dos valores de desviación posicional y un tercero de orientación para el caso 2D, mientras que en análisis 3D se consideran tres desviaciones posicionales y tres de orientación.

Además de considerar las desviaciones introducidas en las distintas etapas de procesado, la técnica SoV también considera etapas de inspección a lo largo del proceso

para el control de calidad. Suponiendo una etapa de inspección después de la etapa k, los resultados de la inspección se describen como:

$$\boldsymbol{Y_k = C_k \cdot X_k + V_k}, \tag{3}$$

donde Y_k es el vector que representa la desviación de las características inspeccionadas después de la etapa k y, por lo tanto, es función de la desviación actual del producto (X_k). En esta expresión intervienen la matriz C_k, similar a la matriz A_k de la primera expresión para considerar el efecto del amarre del producto en el equipo de inspección, y el vector V_k, que representa el error de medición supuesto, calculado generalmente en relación a la incertidumbre del equipo de medida.

Con el fin de ejemplificar la aplicación de esta técnica, a continuación, se presenta un breve ejemplo centrado en el caso de un proceso de ensamble 2D, publicado en [30] y representado gráficamente en la Figura 2.9. Este proceso está compuesto por N etapas en las que la ubicación de cada pieza a ensamblar viene determinada por dos localizadores que limitan, respectivamente, dos y cuatro grados de libertad. Como se puede observar, una desviación en un localizador de la etapa k produce un error en el ensamble resultante de dicha etapa. Dicho ensamble entra posteriormente en la etapa k+1 y se produce un error de posicionamiento del ensamble entrante, provocando un error en el ensamble resultante de la etapa k+1 incluso sin considerar errores en los nuevos localizadores, un factor adicional que introduce desviaciones tanto en el subensamble como en la nueva pieza a incorporar. Finalmente, en la etapa de inspección, se miden las características del ensamble final y se introduce un error asociado a la medición, en función de la incertidumbre de los equipos de medida supuestos.

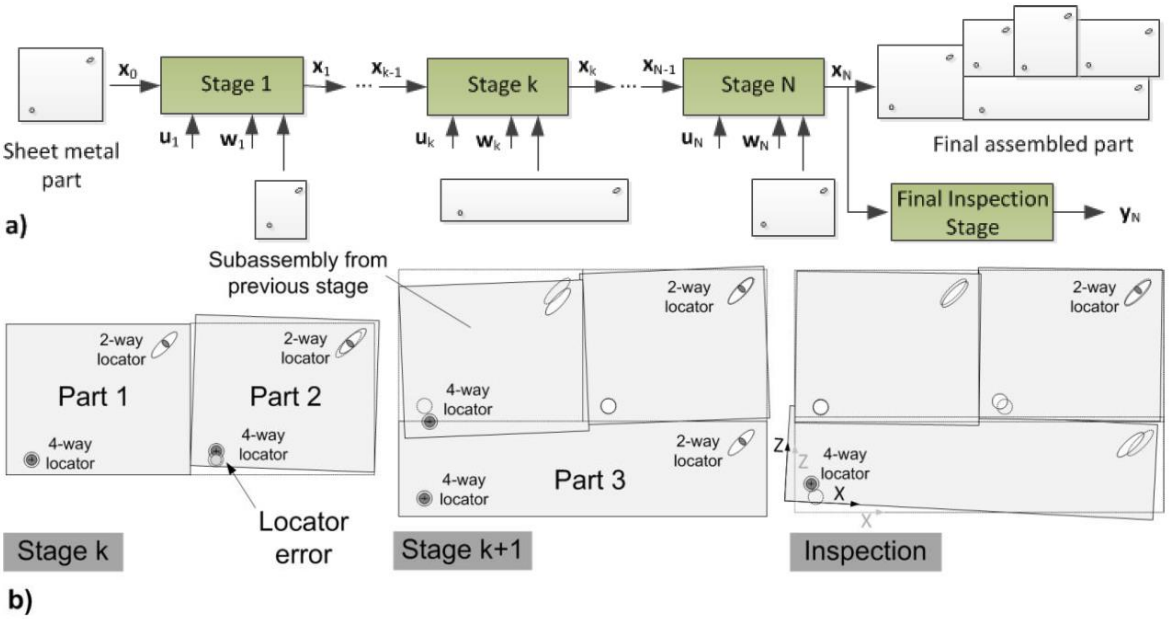


Figura 2.9. Ejemplo de la propagación de errores en un proceso de ensamble multietapa. a) Sistema de ensamble con N etapas. b) Ilustración de la propagación de los errores en un sistema de ensamble 2D [30]

2.3. Diseño Integrado de Producto, Proceso y Recurso

El enfoque que guía esta investigación es el que en la literatura se conoce como Diseño Integrado de Producto, Proceso y Recurso (Sistema de Fabricación), que desde la perspectiva MBSE contempla tanto el desarrollo de modelos descriptivos como de modelos de simulación ejecutables que representen las características del supersistema resultante de la integración de los tres subsistemas, en adelante denominado "sistema PPR". Este enfoque, compatible con la ingeniería de sistemas, facilita el desarrollo de diferentes modelos de diseño y análisis creados en diversas etapas de sus respectivos ciclos de vida, estableciendo interacciones entre ellos. Como en cualquier desarrollo de modelos sobre un sistema, el modelador del sistema PPR debe tener un conocimiento amplio y profundo del mismo, a partir del cual sintetizar o extraer los aspectos clave a modelar. Con este fin, esta sección recoge una breve síntesis sobre algunos aspectos clave del sistema PPR.

En primer lugar, y con el fin de clarificar cada uno de los tres subsistemas, a continuación, se presentan algunas de sus principales características:

- *Producto*. Según *[74]*, es la cosa o sustancia producida por un proceso natural o artificial. Centrados en el contexto de la fabricación mecánica, un producto es un artefacto físico producido a partir de unos materiales de partida que sufren una serie de transformaciones a lo largo del proceso de fabricación. Así pues, este artefacto cambia de características geométricas a lo largo del proceso, pasando por diversos estados. Al finalizar su fabricación (o en etapas intermedias de inspección) cada artefacto producido debe ser conforme a unos requerimientos geométricos descritos en la especificación del producto. Por tanto, en la tarea de modelado será necesario diferenciar entre el producto especificado (conjunto de información necesaria para la fabricación de un tipo de producto) y los productos fabricados o realizados (datos sobre la materialización de un producto, con características concretas), si bien existe una clara relación entre ellos en el sentido de establecer unos requerimientos y obtener unos resultados conformes (o no) con dichos requerimientos.

- *Proceso*. Conjunto estructurado y parcialmente ordenado de actividades, involucrando varias entidades de la empresa y que está diseñado y organizado para un propósito, es decir, que puede ser ejecutado para conseguir un resultado final deseado en busca de un objetivo dado *[75]*. En el ámbito de la fabricación, este objetivo es generalmente obtener un producto completo, o bien una propiedad o aspecto de un producto. Por tanto, se puede interpretar como el comportamiento del sistema en su conjunto, descrito explícitamente o emergiendo del comportamiento del conjunto de recursos y de las relaciones entre ellos. De nuevo, el modelado del proceso admite diversos niveles de agregación, por lo que también será necesario establecer criterios sobre hasta qué nivel de detalle interesa describir o modelar el proceso.

- *Recurso*. A pesar de que existen numerosas definiciones sobre este término según el contexto y la orientación adoptada, una de las propuestas más alineada con el enfoque adoptado es la propuesta de *[76]*, que define el recurso como

> cualquier dispositivo, herramienta y medio, excepto material en bruto y componentes del producto final, necesario para producir bienes o servicios en una empresa. En el ámbito de esta investigación, la tarea de modelado se centrará en el equipamiento y dispositivos electromecánicos con capacidades transformadores (modifican las características físicas del producto), si bien también se considerarán recursos con capacidades logísticas, como transportes y almacenes, ya que influyen en el flujo de materiales y la productividad del proceso. Conviene también tener en cuenta que los recursos pueden referirse a diversos niveles de agregación, por lo que a la hora de modelar estos elementos será necesario representar esta estructura jerárquica y establecer un nivel atómico, que no pueda o no interese ser descompuesto.

En el ámbito de la fabricación, el recurso de mayor nivel de agregación suele ser el sistema de fabricación en su conjunto. En este punto, resulta interesante hacer una distinción entre sistema de producción y sistema de fabricación. Según los autores del estándar MANDATE [77], los sistemas de producción son generalmente grandes sistemas donde las principales entradas son requisitos de producción, conceptos de producción, parámetros de los sistemas, materias primas y componentes, y otras características de gestión de la producción, mientras que los resultados son fundamentalmente productos finales, incluyendo la calidad de esos productos y cualquier otro tipo de información relacionada con el producto. Por su parte, un sistema de fabricación es una parte del sistema de producción centrado en la disposición y operación de elementos (máquinas, herramientas, material, personas e información) para producir un producto físico, informativo o de servicio con valor añadido, cuyo éxito y coste se caracterizan por parámetros medibles establecidos en el diseño del sistema [78]. De forma más resumida, el sistema de fabricación se refiere a la definición de los recursos empleados en la fabricación de productos, es decir, en la transformación de determinados insumos materiales hasta obtener productos finales que cumplan las especificaciones de diseño, mientras que el sistema de producción es una construcción más general, que engloba a todo el negocio dedicado a la producción de productos, incluyendo la planificación, gestión de materiales, aspectos operativos, etc.

Desde el punto de vista de los intereses de esta investigación y de los análisis que se pretenden desarrollar, los modelos de simulación centrados en el análisis de la productividad deberán trasladar aspectos propios del sistema de producción en general, incluyendo cuestiones que afecten al flujo de materiales, la planificación y programación de la producción, los tipos o familias de productos procesados, las hojas de ruta de cada producto, etc., aunque también se reflejan aspectos del sistema de fabricación como la cantidad o tipos de recursos considerados. Sin embargo, en los modelos de simulación extendidos que incorporan el análisis de las desviaciones geométricas tienen un mayor protagonismo aspectos propios del sistema de fabricación, modelando de forma más detallada cada recurso de fabricación y, de forma más específica, sus características geométricas. Aclarada esta diferenciación, a partir de este momento se empleará únicamente el término “sistema de fabricación” para enfatizar la necesidad de modelar los diversos recursos del sistema (estaciones de trabajo, equipos, herramientas, etc.), si bien el modelado de estos sistemas también debe incluir aspectos relativos a la producción.

En relación al modelado y la gestión de datos generados a lo largo del ciclo de vida de los sistemas de fabricación, el análisis presentado en [79] diferencia dos paradigmas principales: la gestión del ciclo de vida del producto (Product Lifecycle Management - PLM) y el modelado integrado de productos, procesos y recursos (Product-Process-Resource model - PPR). Estos dos paradigmas son complementarios, si bien algunos autores consideran el segundo (PPR) una versión más neutral del primero [79]. El PLM suele estar orientado a la creación de herramientas y plataformas que permitan gestionar el conjunto de modelos generados a lo largo del ciclo de vida del producto (aunque también es aplicable a otros sistemas como sistema de fabricación), mientras el PPR está más enfocado en la integración de los tres elementos. Esto es debido a que el producto no puede entenderse como un elemento aislado y autodefinido, sino que depende en gran medida de otros elementos propios del sistema de fabricación y del proceso.

Siguiendo esta orientación, el National Institute of Standards and Technology (NIST) tomó el concepto de ciclo de vida y lo aplicó no sólo al producto, sino también al sistema de producción y al propio suministro de materiales a la hora de establecer el modelo de Ecosistema de Fabricación Inteligente (Smart Manufacturing Ecosystem) [80]. Este modelo representa la confluencia de tres ejes principales que corresponden a los ciclos de vida de estos tres elementos principales, incluyendo además la conocida como pirámide de fabricación, que representa los diversos niveles de automatización que típicamente pueden darse en estos sistemas. Como se muestra en la Figura 2.10, estos ciclos de vida no solo interaccionan en determinadas etapas en que confluyen los ciclos (por ejemplo, interacciones físicas durante la fabricación del producto), sino que también se consideran interacciones a través de flujos de información y datos entre etapas de los diversos ciclos considerados, haciendo necesaria la integración de los modelos y aplicaciones involucradas en las diversas etapas y ciclos de vida.

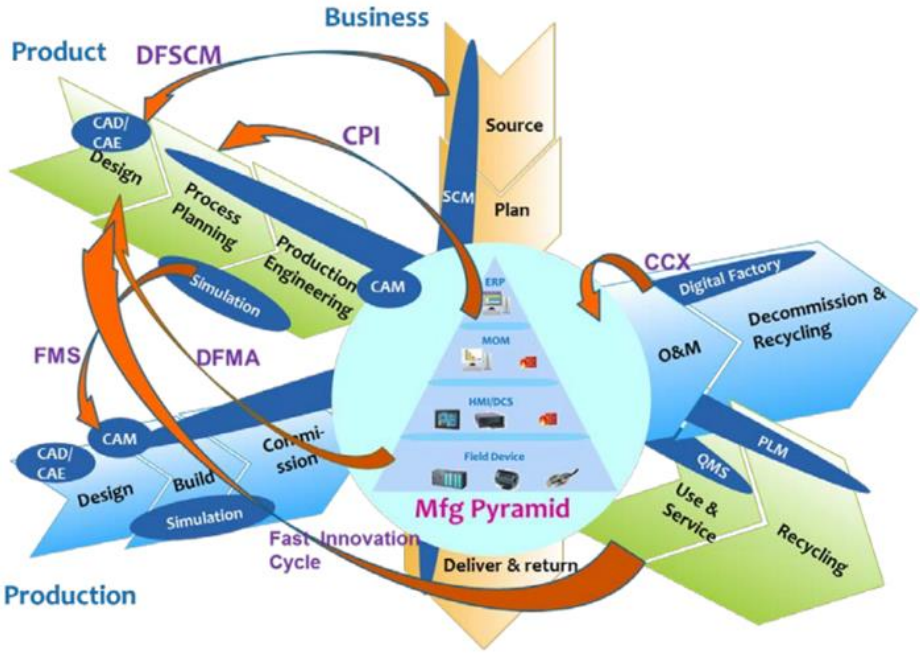


Figura 2.10. Smart Manufacturing Ecosystem desarrollado por el NIST [80]

Se trata de una representación que permite visualizar el foco de esta investigación, que está en las etapas de diseño y construcción del sistema de fabricación, en las que, como se indica en la Figura 2.10, se pueden llevar a cabo simulaciones orientadas generalmente a validar el diseño o la configuración del sistema antes de la etapa de puesta a punto (*commissionning*). Además, como se representa en la figura, existe una estrecha relación entre estas etapas y la planificación del proceso de fabricación de un producto, etapa en la que también destaca el uso de simulaciones para validar las soluciones adoptadas. El Ecosistema de Fabricación Inteligente del NIST sirve para ejemplificar el concepto de sistema PPR, formado por diversos subsistemas, que tienen su propia definición, comportamiento y evolución, pero que interactúan y están conectados entre sí en determinadas etapas.

El modelado integrado PPR tiene su origen a finales del siglo XX, siendo [81] uno de los primeros trabajos en los que se identifican estos tres dominios principales y sus relaciones. A partir de esta concepción, muchos otros autores han seguido este planteamiento, dando lugar a numerosos modelos y estándares. En [82] se presenta una sintetizada pero completa revisión de algunos de estos trabajos, incluyendo estándares como MANDATE [77], o estudios vinculados con diferentes aspectos de la fabricación, como la co-evolución de los sistemas [83], la sostenibilidad [84] o la relación entre requisitos de los producto y las capacidades de los recursos [85], entre muchos otros. Entre estas propuestas, cabe destacar el trabajo presentado en [86], en el que se aborda la consistencia entre los modelos manejados en este enfoque. Tal y como se representa en la Figura 2.11, las relaciones entre los tres subsistemas pueden estar vinculada con expresiones para asegurar la consistencia entre ellos.

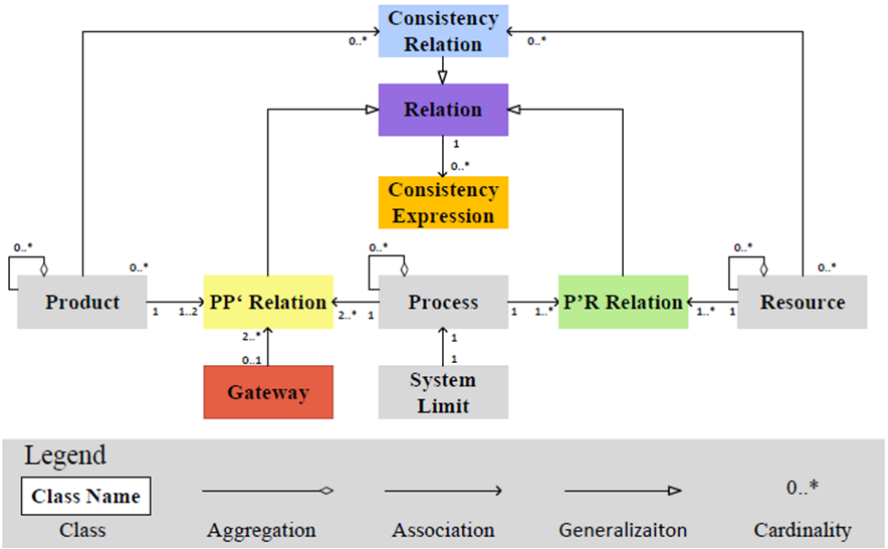


Figura 2.11. Metamodelo del lenguaje de modelado PPR presentado en [86]

Por lo tanto, un aspecto clave para la adopción de este enfoque integrado es el aseguramiento de la consistencia entre los modelos manejados en los diferentes subsistemas, fundamental a la hora de establecer las interacciones entre los diversos subsistemas. Esta orientación también es adoptada en algunas metodologías que emplean la representación en V para el desarrollo integrado del producto y su sistema de fabricación, tal y como se propone en [45], trabajo del cual se extrae la Figura 2.12.

En esta figura, se representa la sucesión en paralelo de tareas de diseño del producto y del sistema de fabricación, que se solapan en la etapa final de ejecución (intersección de los dos ciclos de vida).

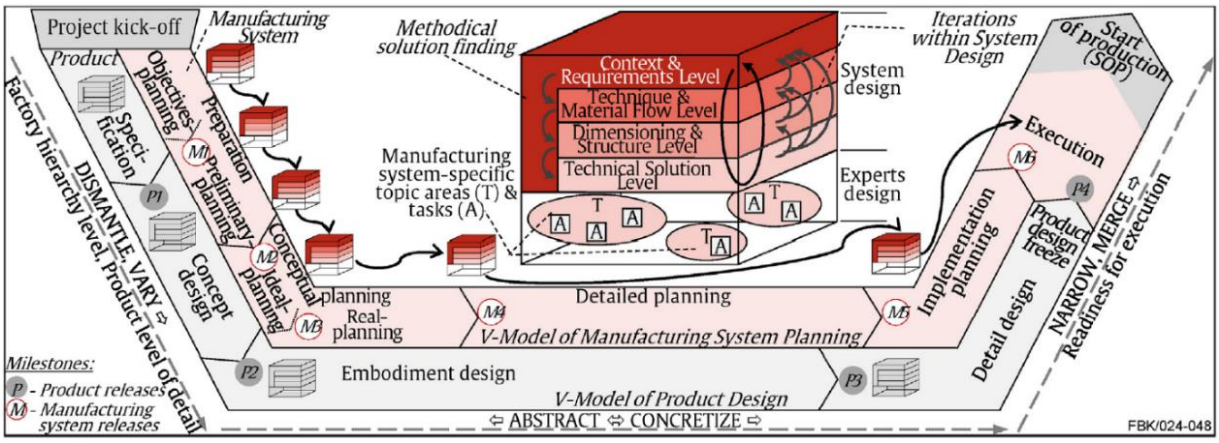


Figura 2.12. Representación en V del diseño integrado del producto y el sistema de fabricación [45]

Vistas algunas generalidades del sistema PPR y enfatizando la necesidad de gestionar adecuadamente los modelos asegurando la consistencia entre ellos, en la sección 5.1 se aborda un análisis más detallado de las características estructurales y/o comportamentales de estos tres subsistemas, especialmente aquellas que influyen en el modelado para el análisis de la productividad y las desviaciones geométricas y que, por consiguiente, serán tomadas en consideración a la hora de desarrollar los lenguajes propuestos en esta investigación para el aseguramiento de la consistencia.

2.4. Evolución de los sistemas de fabricación

A lo largo del último siglo y especialmente en las dos décadas del presente, los sistemas de fabricación han sufrido importantes cambios, impulsados por la irrupción de nuevas tecnologías y por cambios en el mercado y la sociedad, lo que ha supuesto el surgimiento de nuevos paradigmas de fabricación y de ciertos paradigmas operativos que les dan soporte a los grandes cambios que los primeros propugnan. Con el fin de acercar al lector a esta evolución de los sistemas de fabricación, en esta sección se repasan brevemente estos paradigmas, prestando especial atención al paradigma operativo de la fabricación reconfigurable y a sus implicaciones, tanto en el diseño y puesta a punto de los recursos de fabricación como, de forma especial, en el diseño y validación de estrategias de control.

2.4.1. Paradigmas de fabricación

En las últimas décadas, los importantes cambios sociales, la globalización y los rápidos avances en la tecnología han sido factores clave en el establecimiento de nuevos paradigmas de fabricación, que han sido impulsados por nuevas tecnologías y enfoques con el objetivo de satisfacer las exigencias del mercado. Este punto presenta de forma resumida algunos de estos paradigmas, basándose en algunos de los múltiples trabajos de revisión del estado del arte sobre esta cuestión, como [87] o [88].

Desde los inicios del siglo XX, la producción en masa fue el paradigma tradicionalmente adoptado en los grandes sistemas de fabricación, apostando por el diseño de líneas concebidas para soportar la producción en cadena de productos específicos, reduciendo costes e incrementando la calidad y el volumen de producción. Décadas más tarde, en torno a la década de 1980, un nuevo paradigma, la producción flexible, se impuso para dar respuesta a la demanda de una mayor variedad de los productos y, en consecuencia, un menor volumen en la producción de cada tipo. Esta tendencia a incrementar la variedad de productos fue tomando mayor importancia hasta que en los inicios del siglo XXI surgió un nuevo paradigma, la personalización en masa.

De forma más reciente, nuevas tendencias apuntan hacia un nuevo paradigma orientado a la sostenibilidad y alineado con los principios de la denominada cuarta revolución industrial, impulsada por importantes avances tecnológicos, especialmente en las tecnologías de la información y la comunicación. Alineadas con este cambio de paradigma, iniciativas como Smart Factory [1], Industrie 4.0 [2] o Factory of the Future [3] apuestan por una industria más inteligente, sensorizada y sostenible, haciendo uso de los grandes avances tecnológicos, como por ejemplo, los sistemas ciber-físicos, la computación en la nube, el internet de las cosas, o la inteligencia artificial, entre muchos otros. En línea con lo que proponen estas iniciativas, es evidente que las plantas de fabricación de nueva generación deberán ser cada vez más inteligentes, lo cual requiere de un ecosistema interconectado entre todos los individuos y objetos participantes en los nuevos modelos de negocio, haciendo que toda la información del proceso esté disponible en tiempo real. Si nos centramos en propuestas relacionadas con el modelado de sistemas, planteamientos como el gemelo o la sombra digital permiten capturar dicha información y llevar a cabo análisis y estudios para la toma de decisiones en tiempo real. En definitiva, la Industria 4.0 tiene varios beneficios importantes, incluida la integración de operaciones en tiempo real, la reducción de costes, la eficiencia energética, la sostenibilidad, una mayor flexibilidad y productividad, y un mayor retorno de la inversión con el consecuente incremento del rendimiento [89].

Pero con la idea de profundizar en la evolución de los sistemas de fabricación, más allá de presentar una visión general centrada en los grandes paradigmas es importante centrarse en los paradigmas operacionales, es decir, aquellas prácticas que las empresas industriales han adoptado y adaptado con el fin de adaptarse a las nuevas situaciones y mejorar su competitividad. Por ejemplo, la automatización, tanto fija como programable, fueron paradigmas operacionales adoptados para mejorar la productividad. En relación con la mejora de las capacidades de personalización se pueden citar, por ejemplo, los paradigmas operacionales de fabricación Lean o la automatización flexible. Ejemplos de paradigmas que han surgido con el objetivo de mejorar la calidad son el Zero Defect Manufacturing o el Production Quality, este último combinando aspectos de calidad y productividad. Otros que también son relevantes están vinculados con la agilidad, en la que se enmarcan los paradigmas de la fabricación holónica o la fabricación reconfigurable. Una vez que se han presentado los paradigmas más representativos, a continuación, se comentan algunas

características de las tecnologías que les dan soporte, en especial, los sistemas flexibles y los sistemas reconfigurables, identificando sus diferencias.

Las tecnologías que posibilitan la producción flexible son las que caracterizan a los sistemas de fabricación y de gestión de la producción (planificación, programación, etc.) diseñados para una familia de productos, es decir, un conjunto de diversos tipos de productos que comparten ciertas características clave, que posibilitan que sean fabricados empleando recursos comunes. Esto se traduce en unas arquitecturas con unos recursos hardware fijos y unos recursos software que, a pesar de ser también fijos, son programables para adaptarse a cada tipo de producto dentro de la familia considerada. Sin embargo, este enfoque limita las capacidades en algunos aspectos, como la introducción de mejoras, la personalización de los productos o la respuesta a cambios en el mercado.

Alternativamente, los sistemas de fabricación reconfigurables son sistemas estructuralmente modulares, tanto a nivel de hardware como software, lo que permite cambiar la configuración de los sistemas como estrategia para dar respuesta a los cambios en la demanda del mercado. Esto permite convertir de forma rápida y eficiente el sistema para fabricar nuevos productos, integrar nuevas tecnologías que vayan surgiendo y ajustar las capacidades en función de los cambios en el mercado y los productos. En líneas generales, los sistemas reconfigurables poseen las siguientes propiedades:

- *Escalabilidad*. Diseño de la capacidad del sistema de fabricación para adaptarse a la demanda futura del mercado de manera eficiente a nivel de costes.
- *Convertibilidad*. Diseño del sistema de fabricación para la adaptación a los nuevos productos demandados por los clientes.
- *Capacidad de diagnosis*. Diseño de inspecciones de calidad del producto, embebidas de forma óptima en el sistema de fabricación.
- *Personalización*. Diseño del sistema de fabricación en torno a una familia de productos.
- *Maximización de la productividad*. Incremento de la productividad a través de la reconfiguración de operaciones y reasignación de tareas en diversas máquinas o recursos.
- *Mantenimiento efectivo*. Maximización de forma conjunta de la fiabilidad de los recursos y del rendimiento del sistema a través de una adecuada programación de las tareas de mantenimiento.

Como se puede apreciar, las cuatro primeras explotan las características de modularidad e integrabilidad, mientras que los dos últimos están más vinculados con la operativa orientada a mejorar la productividad y la fiabilidad.

Vistas algunas características básicas de los dos tipos de sistemas de fabricación que son objeto de nuestro interés, es importante destacar que tanto la flexibilidad como la capacidad de reconfiguración de los sistemas requieren del diseño de variantes del sistema, incluyendo diversas configuraciones alternativas, ya sean estructuras físicas

diferentes (por ejemplo, el montaje de un determinado utillaje o herramienta), o de naturaleza soft (cargar un determinado código de control numérico, por ejemplo). Por tanto, este tipo de recursos incluyen tareas previas al procesamiento del producto, orientadas a establecer la configuración apropiada o ajustar los equipos. A la hora de gestionar estas variantes y las tareas de configuración y puesta a punto, un elemento clave son los recursos de control, encargados de monitorizar el estado del sistema y tomar las decisiones oportunas. Con el fin de profundizar en esta cuestión se dedica el próximo apartado a tratar en mayor detalle algunos aspectos generales sobre los sistemas de control y algunas estrategias y lógicas de control que serán de interés para la investigación.

Por último, cabe destacar que la complejidad de los sistemas de fabricación, incluidos sus sistemas de control, hacen necesario el desarrollo de modelos de simulación que permitan analizar las soluciones adoptadas durante el diseño, configuración y puesta a punto de estos sistemas, una tarea que se aborda como principal objetivo en esta investigación.

2.4.2. Control: monitorización, diagnosis y toma de decisiones

Como se ha comentado en apartados anteriores, esta investigación está alineada con los principios del paradigma Production Quality, que combina las disciplinas de producción, calidad y mantenimiento, con el objetivo principal de mantener bajo control y mejorar con el tiempo la tasa de producción y el nivel de cumplimiento de las piezas con un desperdicio mínimo de recursos y materiales [8]. Este paradigma está alineado con el enfoque de fabricación sin defectos (Zero Defect Manufacturing - ZDM), cuyo objetivo es la eliminación de defectos en los productos fabricados [7], pero este objetivo es más difícil de lograr cuando los sistemas de fabricación se vuelven más complejos, como en el caso de sistemas multietapa, en los que los defectos de calidad se propagan a lo largo de las diferentes etapas.

Para abordar este y otros retos, un elemento fundamental de los sistemas de fabricación son los sistemas de control que, además de monitorizar el estado del sistema (captura y análisis de datos) incluyen la toma de decisiones para controlar determinados automatismos y llevar a cabo estrategias de control activo. Con el objetivo de profundizar en aquellas estrategias de control activo dirigidas a reducir o evitar defectos de calidad y con ello a mejorar la productividad, en este apartado se comentan algunas características básicas de los sistemas de control y se describen ciertas estrategias de control que se alinean con el paradigma de Production Quality y que se van a explorar en la investigación.

Así pues, los indicadores que van a guiar el diseño del sistema son dos, la productividad y la calidad geométrica. Por una parte, el control y la mejora de la productividad están fundamentalmente vinculadas con los flujos de producción y logística (a partir de ahora flujos logísticos) de los materiales, y se centran en mejorar la eficiencia aprovechando al máximo los recursos disponibles para lograr producir el máximo número de productos en el menor tiempo posible, gestionando de forma adecuada los problemas ocasionados por bloqueos, reconfiguraciones de máquinas, etc. En este sentido, se debe destacar que el control del flujo logístico y la gestión de los problemas

antes mencionados es más complejo en sistemas no lineales en los que existen rutas alternativas, o en sistemas en los que se procesan productos diferentes con recursos comunes, entre otros casos. Por otra parte, por lo que respecta a la calidad geométrica de los productos finales y a su control y mejora, algunos sistemas utilizan los resultados de la medición de determinadas características de los productos para tomar decisiones correctivas, por ejemplo, modificando la configuración de los recursos (ajuste de utillajes, cambios de herramienta, etc.) o incluso tomando decisiones sobre la ruta seguida para procesar con un nivel de calidad diferente en función de la pieza procesada.

Como se puede apreciar, ambos objetivos están íntimamente relacionados, de manera que las decisiones tomadas en favor de la mejora de un criterio pueden influir negativamente en el otro. Por ejemplo, el ajuste de las condiciones de trabajo para cada pieza procesada puede mejorar de forma considerable la calidad geométrica, pero incrementa el tiempo invertido y reduce la productividad. En cambio, la ausencia de inspecciones o de ajustes de máquina debe incrementar el número de unidades producidas por unidad de tiempo, pero muy probablemente afecta a una peor calidad en el producto, hasta el punto de aumentar el número de productos defectuosos y reducir la productividad real.

Ahora, antes de centrar el discurso en las estrategias que son de interés para la investigación, conviene recordar qué es un sistema de control en el contexto de los sistemas de fabricación, las partes que lo constituyen y los tipos de lógicas de control. En general, un sistema de control es aquel que, mediante la comprobación del funcionamiento de otros sistemas, es capaz de administrarlos, ordenarlos y dirigirlos a fin de obtener los resultados ideales y reducir las probabilidades de fallo, al tiempo que sirven para aumentar la seguridad, rentabilidad y fiabilidad de los procesos. En el caso de los sistemas de fabricación, el sistema de control es una parte fundamental que monitoriza el estado de otros subsistemas (recursos de diversos tipos), recopilando datos sobre el funcionamiento de éstos y tomando decisiones para modificar y ajustar aspectos variables con los que mejorar los resultados obtenidos y acercarlos a unos criterios determinados. Para ello, por lo general, los sistemas de control están dotados de mecanismos de monitorización (captura de datos), diagnóstico (análisis del estado, identificando problemas y sus causas) y toma de decisiones (medidas correctivas para resolver los problemas). Por otra parte, por lo que respecta a los flujos de datos que manejan, el control puede ser de lazo abierto, normalmente con una lógica FeedForward, o de lazo cerrado. Finalmente, por lo que respecta a las características estructurales del sistema de control (arquitecturas), se pueden diferenciar dos tipos de control: Control centralizado y Control distribuido.

Centrándonos ahora en la selección de la estrategia de control y en su diseño, cabe reseñar que ésta debe buscar el cumplimiento (optimización) equilibrado de toda una serie de indicadores clave de rendimiento (KPI), en el caso que nos ocupa relacionados con la productividad (por ejemplo, la tasa de producción), la calidad (por ejemplo, la tasa de defectos) o con ambas (por ejemplo, la eficacia global del equipo). Para ello, una herramienta que puede ser de gran ayuda es la simulación.

La simulación es una herramienta muy potente con la que replicar el comportamiento del sistema real de fabricación y poner a prueba los efectos de las decisiones de control. La realización de diversos experimentos de simulación permite comparar diferentes estrategias con diferentes parámetros de diseño (tamaño o frecuencia del muestreo, número de etapas de inspección, localización de las mismas etc.) teniendo en cuenta los valores obtenidos para los KPI's establecidos en nuestro cuadro de mando. Pero, como las simulaciones deben emular el comportamiento del sistema referente, otro aspecto importante en el diseño del sistema de simulación tiene que ver con los flujos de datos que emulan los flujos (de materiales, datos, señales, etc.) del referente. En esta investigación, las simulaciones a desarrollar van a manejar dos flujos de datos:

- *Flujo logístico*, que emula el flujo de materiales a través del sistema de fabricación. En este flujo los datos que se transfieren de unos bloques (etapas del proceso) a otros son los que permiten monitorizar el intercambio de los materiales emulados. Este flujo, junto con el comportamiento discreto de los diversos bloques considerados, permite simular la dinámica (eventos discretos) del sistema, pudiéndose obtener tras los experimentos de simulación diferentes medidas de rendimiento, como la relativa a los tiempos (tiempos medios de procesado, de bloqueo, de transporte, etc.), la carga de trabajo, las piezas procesadas en los diversos subsistemas (productividad, ocupación media de los almacenes, etc) o el porcentaje de piezas defectuosas, por poner algunos ejemplos. Esta información permite evaluar el impacto de las estrategias de control sobre la calidad de producción, a través del cálculo de los KPI's correspondientes.

- *Flujo de desviaciones*, que emula la transmisión de las desviaciones geométricas del producto a través del proceso multietapa. Los datos de este flujo, que se transfieren de unos bloques a otros, y la formulación matemática que permite calcular el valor de las desviaciones en cada etapa (comportamiento emulado), permiten simular la propagación y acumulación de las desviaciones. Un aspecto a tener en cuenta, es que el proceso simulado puede contener etapas de medición, en las que el efecto de las incertidumbres de medida, que afectará a las desviaciones de las etapas de fabricación. Esto permite que se contemplen los efectos del plan de medición en la evaluación de las estrategias de control orientadas a la mejora de la calidad. Este plan puede contemplar más o menos etapas de inspección, si se van a medir todos los productos o un muestreo con ciertos parámetros (frecuencia, tamaño muestral, etc.), unas decisiones que pueden influir notablemente en los tiempos invertidos y que puede ser evaluada mediante las simulaciones.

Llegados a este punto, es conveniente indicar que las experiencias (casos) que se van a realizar en esta investigación se centran en abordar simulaciones de procesos de fabricación lineales, con hojas de ruta (procesado y medición) completamente definidas y sin considerar alternativas. Además, las simulaciones desarrolladas no abordan la toma de decisiones relativas a la modificación de la configuración del sistema de fabricación para mejorar la productividad, pero esto no implica que los sistemas de simulación desarrollados no sean capaces de soportar este tipo de análisis.

Otro aspecto a tener en cuenta, por lo que respecta al control basado en el flujo de desviaciones, es que este no se limita a la monitorización, porque ha de soportar la

toma de decisiones correctivas relacionadas con el ajuste de los amarres, calculando la posición óptima de los localizadores para minimizar las desviaciones del producto final. Esto se consigue aplicando dos lógicas de control:

- *FeedForward*. Implementa la técnica propuesta en *[71]*, en la que se utilizan los datos de procesos de medida en etapas tempranas de la fabricación para calcular la posición óptima de los localizadores en etapas posteriores, minimizando las desviaciones en el producto final. Este cálculo y las respectivas tareas de ajuste de los localizadores se ejecutan para cada producto individual.

- *FeedForward&Backward*. Ampliación de la lógica anterior, calculando la posición óptima de los localizadores para todas las etapas de fabricación. Además del cálculo para las etapas posteriores, esta segunda propuesta emplea los datos medidos (generalmente calculando la media de un determinado muestreo) para calcular correcciones en etapas anteriores que reduzcan también el valor de las desviaciones. En este caso, un factor muy influyente será la frecuencia y el tamaño de la muestra empleada, ya que estos parámetros deben definirse considerando las capacidades de los recursos transformadores y los almacenes previos, de manera que el muestreo sea representativo, midiendo piezas que han sido procesadas completamente con las nuevas condiciones adoptadas debido a la última decisión correctiva del control.

Para un mayor detalle de estas propuestas se recomienda consultar la publicación [30], en el que además se muestra un caso de estudio en el que ambas lógicas de control son aplicadas y comparadas.

Finalmente, conviene indicar que las experiencias (casos) que se van a realizar en esta investigación: a) se centran en abordar simulaciones de procesos de fabricación lineales, con hojas de ruta (procesado y medición) completamente definidas y sin considerar alternativas, y b) no abordan la toma de decisiones relativas a la modificación de la configuración del sistema de fabricación para mejorar la productividad. Esto no implica que los sistemas de simulación desarrollados no sean capaces de soportar este tipo de análisis.

3. Estudio preliminar de simulaciones multidominio

Tras una primera revisión bibliográfica, centrada en el estudio de modelos de simulación para el análisis de sistemas de fabricación en etapas tempranas de su diseño, se plantea el desarrollo de un modelo para simular un sistema de ensamble multietapa (MAS). Este primer modelo de simulación multidominio pretende integrar tanto el modelado del comportamiento de eventos discretos propio del MAS orientado al análisis de la productividad, como el modelado de las desviaciones geométricas y el análisis de su propagación a lo largo de las diversas etapas en base a los principios del Stream of Variation (SoV), descrito en el Capítulo 2. En el ámbito particular de la fabricación, se han realizado numerosos esfuerzos enfocados principalmente al análisis de producción y mantenimiento con simulaciones de eventos discretos, pero se ha detectado una falta de investigación orientada a la gestión de variaciones geométricas, especialmente en modelar y simular los efectos de las variaciones geométricas para el aseguramiento de la calidad, la productividad y sus interrelaciones, lo que motivó esta propuesta.

Con este objetivo, se planteó la necesidad de explorar las posibilidades que ofrece el lenguaje Modelica para la creación de estos modelos de simulación multidominio, ya que se perfilaba como uno de los principales candidatos para dar soporte a la metodología que se pretende desarrollar en la investigación. Así mismo, con esta experiencia también se exploraron algunos entornos de modelado, y en particular el uso de OpenModelica. Este acercamiento práctico tanto al lenguaje como a uno de sus entornos de modelado permite hacer balance de su potencial, ventajas e inconvenientes desde el punto de vista de los objetivos planteados en esta investigación.

Sin embargo, antes de abordar el modelado con Modelica, se planteó un análisis del sistema referente (MAS), así como de los principales conceptos manejados en el propio modelo de simulación. En estas primeras etapas del diseño y desarrollo del sistema de simulación se consideró interesante hacer uso de modelos conceptuales que permitieran describir de una manera gráfica y formal las bases conceptuales tanto del sistema MAS como del sistema de simulación a desarrollar. De hecho, el propio enfoque MBSE promueve la construcción de modelos de simulación ejecutables a partir de modelos descriptivos, poniendo de relieve la necesidad de automatizar las transformaciones entre lenguajes de ambos tipos. En este sentido, la experiencia sirvió para tener un acercamiento al modelado con SysML, un lenguaje muy extendido en el modelado de sistemas, así como explorar el manejo de la herramienta de modelado

Papyrus, uno de los entornos libres más extendidos para el modelado con UML y SysML.

Con el fin de resumir esta experiencia, el capítulo se estructura en diversas secciones para tratar de forma detallada las siguientes cuestiones: descripción de los lenguajes y entornos utilizados; modelado del prototipo en sus diferentes etapas; y caso de estudio desarrollado para validar la propuesta. Finalmente, también se incluye una discusión con la valoración de esta experiencia y su relación con algunos trabajos previos.

Cabe comentar que gran parte de los contenidos que se presentan a continuación fueron publicados en [30], si bien algunas secciones han sido adaptadas (ampliadas o compactadas) con el fin de integrarlas en el formato de este documento y cohesionar el discurso.

3.1. Lenguajes y herramientas empleadas

Desde finales del siglo pasado, las metodologías de diseño y modelado de sistemas orientados a objetos han ido ganando popularidad en el campo del diseño y análisis de sistemas, debido a que se acercan conceptualmente al enfoque de sistemas principalmente a través del principio de encapsulación y la definición de interfaces para la interacción entre subsistemas. Junto con estas metodologías, numerosos lenguajes han sido propuestos adoptando esta orientación, tanto en el ámbito del modelado conceptual como el de análisis. Cabe destacar que el hecho de compartir ciertas bases conceptuales facilita enormemente la interacción/transformación entre modelos implementados en lenguajes diferentes, promoviendo su uso combinado en determinados contextos. Siguiendo estos principios, en este apartado se presentan los dos lenguajes que han sido utilizados tanto en el diseño como en la implementación del modelo de simulación multidominio propuesto. Por una parte, SysML es el lenguaje adoptado para el modelado descriptivo del sistema de simulación, mientras que por otra se ha utilizado Modelica para implementar los modelos de simulación ejecutables. A continuación, se dedica un apartado para tratar en detalle cada uno de ellos.

3.1.1. System Modeling Language (SysML)

En el modelado de sistemas, y asumiendo la adopción de los principios de diseño y análisis orientado a objetos, uno de los lenguajes más extendidos y adoptados en la literatura es Systems Modeling Language (SysML) [90]. En este apartado se presenta de forma muy resumida algunas de las principales características de este lenguaje con el fin de facilitar la interpretación de los modelos en los que se haga uso de este lenguaje a lo largo del documento. Sin embargo, para una más detallada comprensión de SysML se recomienda consultar su especificación [90] o publicaciones como [91].

SysML es un lenguaje descriptivo para el modelado de sistemas en general que permite el establecimiento de comunicaciones formales y no-ambiguas entre ingenieros, diseñadores y analistas de sistemas, soportando la capacidad de comunicación incluso entre personas que no sean necesariamente especialistas. Una de sus principales ventajas es la capacidad de representar gráficamente vistas del modelo a través de diagramas, lo que facilita la comunicación y la interpretación de los modelos por parte

de los usuarios. SysML también admite la definición en formato texto de algunas características del sistema, fundamentalmente comportamientos implementados como expresiones opacas, si bien estas expresiones deben estar escritas en lenguaje natural o codificadas en otros lenguajes, como, por ejemplo, Java, C++, Modelica, etc. Además, conviene comentar que los modelos SysML suelen ser codificados en texto mediante lenguajes como XML [92] para su intercambio, si bien estos modelos textuales no almacenan aspectos gráficos sobre los diagramas, sino únicamente las construcciones (entidades y relaciones) que conforman el modelo.

En la literatura es común definir SysML como un perfil (con ciertas particularidades) que extiende Unified Modeling Language (UML) [93] para acercar dicho lenguaje a la ingeniería de sistemas y capturar con suficiente detalle los conceptos más significativos de este campo. Además, como perfil de UML, SysML también cuenta con el mecanismo de creación de perfiles, el cual puede ser utilizado con el fin de crear lenguajes de modelado específicos de dominio (Domain-Specific Modeling Language – DSML), es decir, lenguajes construidos a partir de los anteriores pero que soportan los conceptos y la semántica propia de dominios más específicos, una herramienta que será de especial interés a lo largo de esta investigación. Cabe indicar que, por su parte, UML está basado en un meta-metamodelo llamado Meta Object Facility (MOF) [94], que se sitúa en el nivel M3 de la arquitectura mostrada en la Figura 3.1. En esta figura se representa una arquitectura similar a la presentada en la Figura 2.2, pero esta vez situando los lenguajes y metamodelos mencionados, y con algunas modificaciones correspondientes a algunas particularidades de SysML que se comentan a continuación.

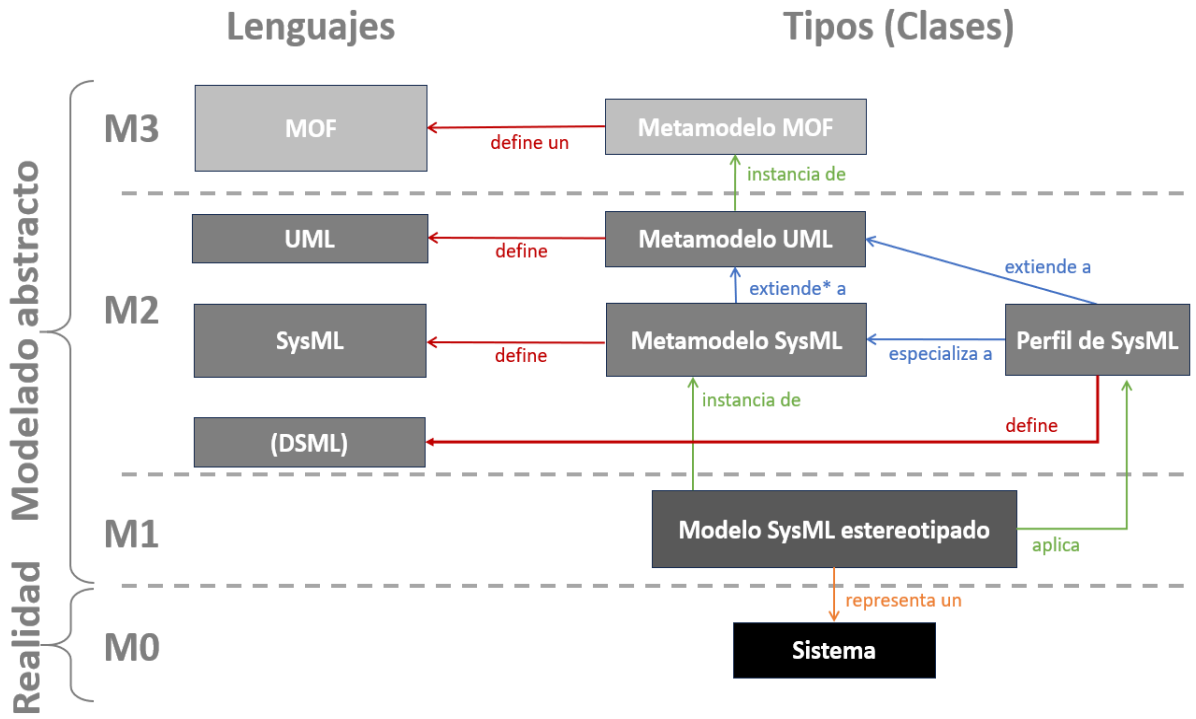


Figura 3.1. UML y SysML en la estructura de niveles de modelado.

En primer lugar, es importante tener en cuenta que SysML no es un perfil que extienda todo UML, sino que es un lenguaje definido a partir de una versión reducida de UML

conocida como UML4SysML, un subconjunto en el que se adoptan gran parte de los conceptos de UML, dejando al margen otros. Este solapamiento entre ambos lenguajes se representa gráficamente en la Figura 3.2, mostrando todo UML en color rojo y destacando un área en color morado para representar UML4SysML, que a su vez forma parte del área que representa SysML (azul). Así mismo, la Figura 3.2 también representa dos mecanismos de creación de DSML, tanto la definición de perfiles cuyos estereotipos extienden UML (naranja), como perfiles de SysML (verde) cuyos estereotipos pueden especializar estereotipos de SysML o bien extender metaclases de UML4SysML.

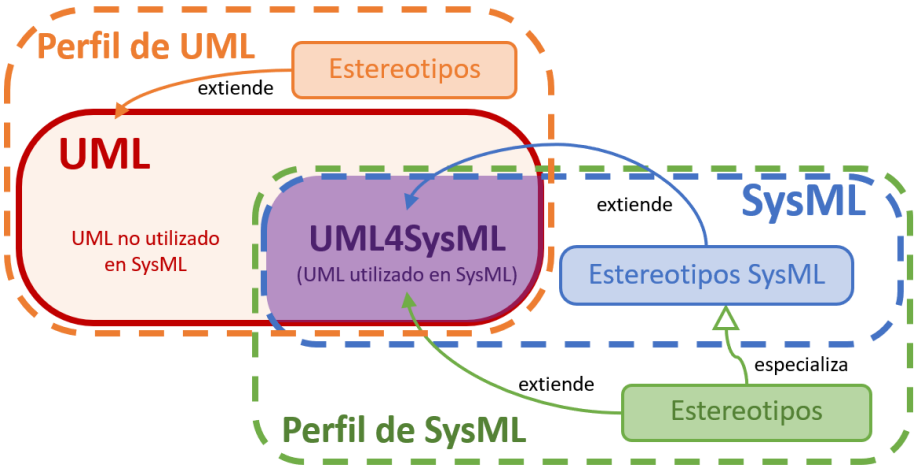


Figura 3.2. Relaciones entre UML, SysML y otros perfiles.

Es precisamente este mecanismo de creación de perfiles una de las capacidades de SysML más interesantes para esta investigación. En estos perfiles, además de extender el lenguaje para incorporar nuevos estereotipos, también se incluye la definición de reglas, generalmente implementadas con el lenguaje Object Constraint Language (OCL) [95]. Estas expresiones OCL limitan la sintaxis de los elementos del modelo en los que se aplica un determinado estereotipo con la pretensión de capturar aspectos semánticos propios del concepto al que representa dicho estereotipo. Así pues, estas reglas OCL definidas a nivel de perfil (M2) se deben validar en los modelos de usuario (M1) donde se aplican estereotipos. El proceso de validación del modelo comprueba si las reglas OCL se cumplen, unas consultas sin efectos secundarios en el modelo, es decir, sin modificarlo o alterarlo, pero permitiendo identificar inconsistencias en el modelo.

Como se ha comentado anteriormente, la representación gráfica de vistas de los modelos es una de las principales ventajas de SysML, que ofrece una amplia variedad de diagramas, algunos de ellos similares a los disponibles en UML y otros propios de SysML, que permite a los usuarios interpretar el modelo (o partes del mismo) de manera sencilla y formal. Es conveniente destacar que el conjunto de diagramas definidos en un modelo no son el modelo en sí. Un diagrama es únicamente una representación gráfica de vista (una parte) del modelo desde un punto de vista o unos intereses particulares. La Figura 3.3 muestra un esquema de los diagramas considerados en SysML. Para facilitar la interpretación de las figuras, en este documento se adopta notación propia de lenguajes como UML y el propio SysML, en

los que se emplea la relación con un triángulo en su extremo para representar la relación de especialización/generalización, mientras que el formato de letra cursiva se emplea para las entidades abstractas (no existen objetos de ese tipo, ya que solo se pueden instanciar sus especializaciones).

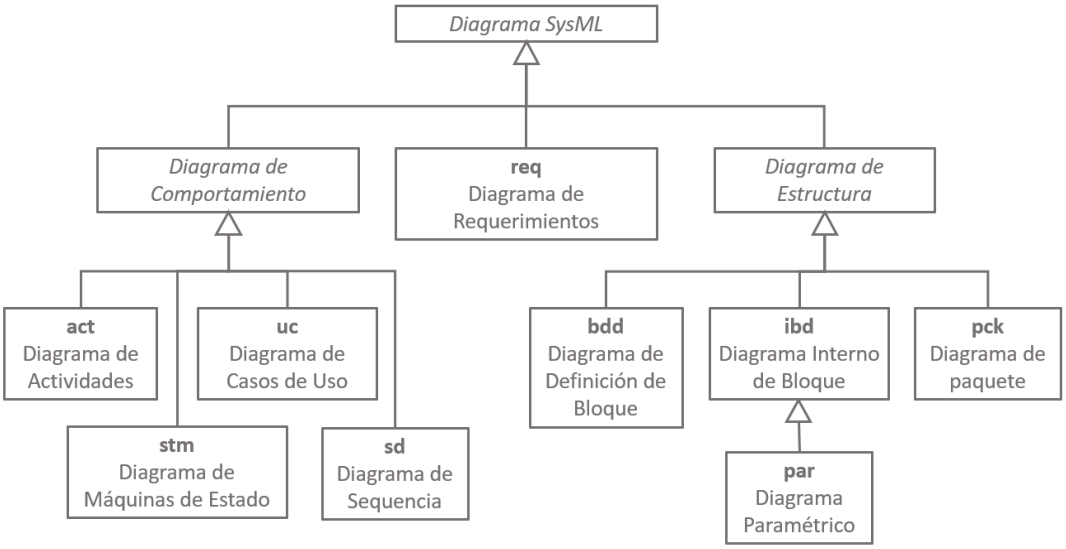


Figura 3.3. Diagramas de SysML.

A efectos de los intereses y el uso que se hace de SysML en esta investigación, serán especialmente relevantes diagramas estructurales como los diagramas de definición de bloques (Block Definition Diagram - BDD), los diagramas internos de bloque (Internal Block Diagram - IBD) y los diagramas paramétricos (Parametric Diagram - PAR). Los BDD son los diagramas más comunes, y se emplean para mostrar los elementos de definición (bloques, bloques restricción, interfaces, etc.), visualizando algunas de sus características y las relaciones entre ellos, ya sean generalizaciones, dependencias o asociaciones (incluidas composiciones, agregaciones y bloques asociativos). Por su parte, los IBD se emplean para representar aspectos estructurales internos de determinados bloques, mostrando algunas de sus propiedades (partes, referencias y puertos fundamentalmente) y las conexiones entre ellas. Una especialización particular de los IBD son los PAR, un tipo de diagrama particular de SysML que permite mostrar información sobre las restricciones del sistema, particularmente mostrando las relaciones entre los parámetros de las restricciones definidas y propiedades valor del bloque contexto o de alguna de sus propiedades. En cuanto al modelado de comportamiento, será de especial importancia el uso de diagramas de máquinas de estado (State Machine Diagrams - STM), pero esta cuestión será tratada con mayor detalle más adelante.

En cuanto a los principales conceptos manejados en SysML, el Bloque (Block) es la unidad estructural básica que representa cualquier tipo de entidad del sistema modelado o de su entorno. Conviene aclarar que el Bloque es un elemento de definición de tipos (similar al concepto de Clase en UML), y no de entidades (objetos) particulares. Más allá de tener un nombre que lo identifique, el Bloque puede tener una serie de características (features) tanto de tipo estructural (partes, referencias, propiedades valor, restricciones o puertos) como comportamental (operaciones y

recepciones). Estas características son, en su mayoría, usos de otros elementos del modelo, es decir, que son concreciones de un tipo que es definido en otro elemento del modelo (bloque, tipo de valor, etc.). Por ejemplo, las propiedades de tipo parte (part) o puerto (Port) son usos de otros bloques, mientras que las propiedades valor (ValueProperty) son usos de tipos de valor (ValueType).

Conviene hacer mención especial a las propiedades de tipo Puerto, que son elementos estructurales que un bloque ofrece para la conexión con otros elementos de su entorno. Los puertos se conectan entre sí mediante Conectores en un IBD, y estos conectores pueden representar una igualdad total entre puertos (BindingConnectors) o tener un *tipo* particular descrito en una Asociación (o bloque asociativo) del modelo.

SysML también permite definir restricciones (*Constraints*), que en ocasiones se implementan como bloques restricción (*ConstraintBlocks*) con el fin de definir un elemento reutilizable en el modelo. Estas restricciones habitualmente contienen relaciones matemáticas entre parámetros, cuyo valor se asigna mediante la conexión con propiedades valor en un diagrama paramétrico.

Con el fin de ofrecer un acercamiento más práctico a estos conceptos, la Figura 3.4 muestra algunos diagramas del modelado básico de un coche. Por ejemplo, para aclarar la diferencia entre *tipo* y *uso* se toma como ejemplo el modelado la sentencia: "un coche tiene cuatro ruedas". En el modelo propuesto, los bloques Coche y Rueda recogen la definición de qué es un coche y qué una rueda respectivamente. Entre ambos elementos existen relaciones de composición para definir que un Coche está compuesto por cuatro ruedas, es decir, cuatro usos del bloque Rueda. Estas cuatro relaciones de composición, representadas en el BDD de la Figura 3.4 (A), vienen definidas cada una por un *rol,* que es el nombre que identifica a cada uso de rueda: delantera derecha (dd); delantera izquierda (di); trasera derecha (td) y trasera izquierda (ti). Además, la multiplicidad 1 indica que únicamente se utiliza una rueda con cada rol. En este ejemplo, se emplea la representación explícita de las relaciones de composición para mostrar la estructura del bloque Coche, aunque las características (features) también se pueden mostrar alternativamente en compartimentos adicionales, como se ejemplifica con el atributo "diam" (propiedad valor de tipo real) de la Rueda.

Siguiendo con la interpretación de este ejemplo, se puede ver que cada Rueda está caracterizada por una propiedad valor de tipo real llamada "dim" que define el diámetro de la rueda. Además, otra propiedad de tipo Puerto representa la llanta, modelada de este modo por ser una parte que servirá como interfaz para la interacción con el entorno del bloque (rueda), en este caso para representar la conexión mecánica con otros componentes del coche. En este sentido, se define un bloque asociativo (asociación estereotipada como bloque) que relaciona dos ruedas y que representa al eje mecánico que las conecta. Estos aspectos pueden verse representados en el IBD de la Figura 3.4, que muestra la estructura interna del Coche y que, además de cuatro propiedades (de tipo Rueda), también muestra dos conectores (de tipo Eje), cada uno de los cuales relaciona dos de dichas propiedades a través de sus puertos (de tipo Llanta). Cabe indicar que, más allá del uso que se da en este ejemplo, los puertos y conectores de SysML también permiten modelar el flujo de información (datos),

materia o energía. Finalmente, se ha definido además un bloque restricción (Igualdad) que establece la igualdad entre dos parámetros de tipo real. El bloque asociativo Eje hace uso de Igualdad para establecer la condición de que el diámetro de las dos ruedas que relaciona son iguales. Esta relación entre los parámetros de la restricción y las propiedades valor correspondientes se muestra en el PAR del bloque Eje, representado en la Figura 3.4.

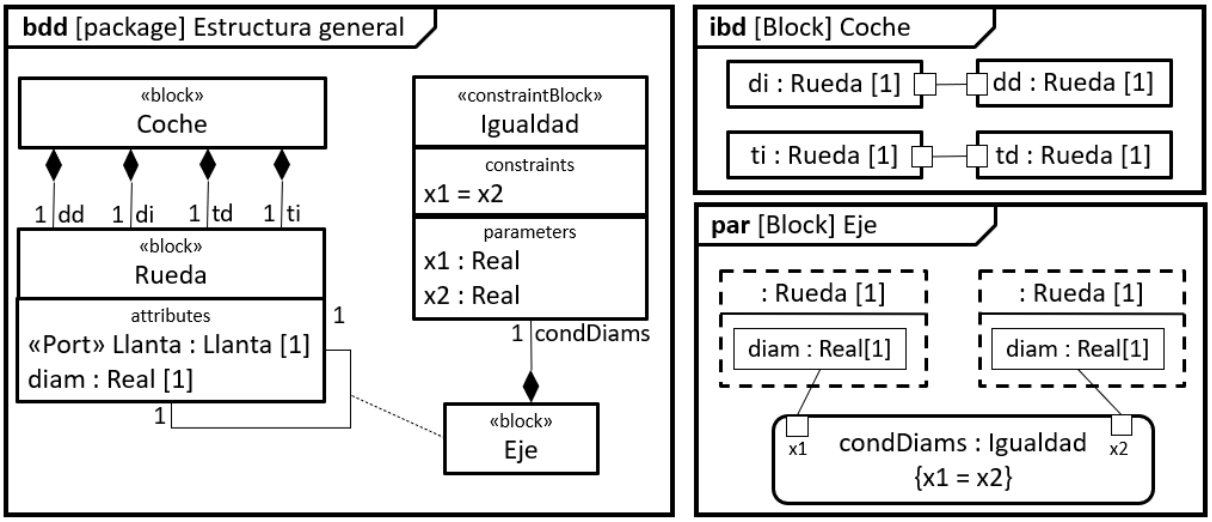


Figura 3.4. Diagramas estructurales del modelo ejemplo Coche.

Más allá de las características estructurales, SysML también permite definir comportamiento dinámico a través de tres tipos de diagrama y sus respectivas construcciones: diagramas de actividades, de secuencia y de máquinas de estado. Cada una de estas alternativas ofrece una forma de modelar comportamiento haciendo especial énfasis en ciertas características del comportamiento (temporalidad, flujos, eventos y transiciones, etc.). A efectos de interpretar adecuadamente el uso de SysML que se da en esta investigación, conviene destacar los diagramas de máquinas de estado, ejemplificados en la Figura 3.8. En este tipo de diagrama se representan un conjunto de estados, las posibles transiciones entre ellos, y los disparadores (*triggers*) y guardas (*guards*) que condicionan la ejecución de dichas transiciones. Esta forma de modelar el comportamiento está muy alineada con el formalismo de simulación de eventos discretos (Discrete EVent System - DEVS), tan característico de los sistemas de fabricación y sus respectivos modelos de simulación.

Profundizando un poco en el alcance o las capacidades de representación de los modelos, cabe indicar que algunos lenguajes de modelado (como eCore o OWL) presentan una clara diferenciación entre los modelos de clases (tipos) y los modelos de instancias (individuos/objetos), tal y como se representa en la Figura 2.2. Sin embargo, en el caso de modelos definidos en SysML, el modelado de clases y objetos no se implementa en modelos diferenciados, sino que un mismo modelo puede incluir entidades que representen tipos (Blocks en SysML) y otras entidades que representan instancias concretas (InstanceSpecifications en SysML), conviviendo ambas construcciones en un mismo modelo. De esta manera, un modelo SysML puede estar representando un tipo de sistema, o bien representar un sistema en concreto (objeto).

Esta falta de concreción ha sido causa de cierta incertidumbre en las versiones más extendidas de SysML (SysML 1.4 y anteriores). Recientemente, la especificación de SysML 2 [96] busca dar una solución más específica a esta cuestión mediante la entidad Individual Element, si bien se trata de una versión del Instance Specification para representar un objeto identificable. Es necesario aclarar que SysML V2 no ha sido adoptado en la investigación por su reciente publicación y la falta de implementación en las herramientas de modelado actuales.

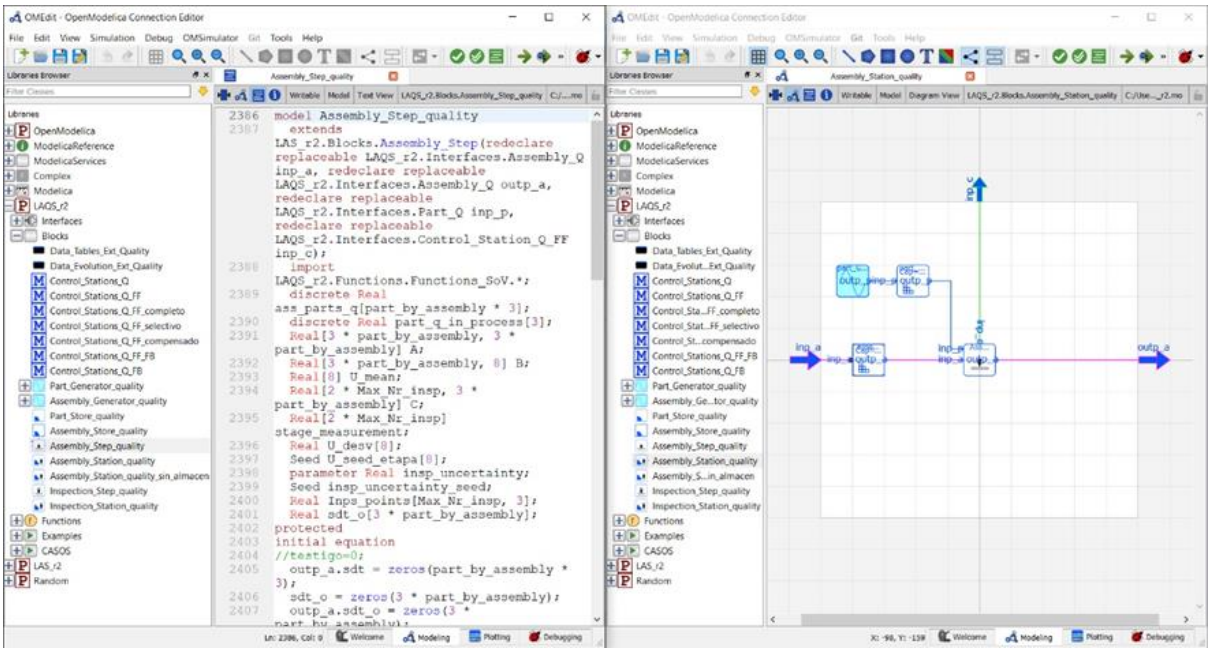
3.1.2. Modelica

Durante la última década del siglo XX, una primera generación de lenguajes con orientación a objetos para el modelado matemático de sistemas de simulación fue desarrollado, incluyendo ObjectMath, Dymola, Omola, NMF o gPROMS, entre otros [44]. Sin embargo, el desarrollo de diversos lenguajes incompatibles puso de manifiesto la necesidad de estandarización. Ante esta situación, la Asociación Modelica [97] impulsó iniciativas que inicialmente estuvieron centradas en el desarrollo del lenguaje Modelica [98] y de librerías Modelica que facilitaran el modelado y simulación de sistemas complejos tradicionales como los mecatrónicos, y que más recientemente han dado soporte al diseño de sistemas ciber-físicos. A continuación, se presenta un breve resumen de las bases de Modelica, aunque para un mayor detalle se recomienda consultar su especificación [98] o guías más prácticas como [99] o [100].

Modelica es un lenguaje que permite la definición de modelos de manera declarativa, modular y jerárquica, con capacidades multidominio [44]. Estas características lo convierten en una buena solución para modelar sistemas complejos que requieren combinar elementos de diferentes dominios, por ejemplo, componentes mecánicos, eléctricos, electrónicos u orientados a procesos, entre otros. Modelica es un lenguaje fundamentalmente textual, aunque cuenta con ciertas capacidades de representación gráfica a través de la definición de anotaciones en el modelo. Entre las principales ventajas que ofrece Modelica frente a otros lenguajes de modelado de sistemas de simulación, el autor de [44] destaca su orientación a objetos y el modelado acausal, entre muchas otras. Por una parte, la orientación a objetos facilita la creación de componentes, así como la estructuración, reuso y evolución de modelos complejos. Por otro lado, el modelado acausal permite la definición de ecuaciones (igualdades que se cumplen en los dos sentidos), lo cual permite adaptar el sistema de ecuaciones al flujo de datos en el contexto en el que son utilizadas. Esta característica permite crear modelos más flexibles que se adapten a diferentes flujos de la información, frente a aquellos lenguajes limitados a establecer relaciones causales entre unas entradas y salidas claramente definidas, como es el caso de Simulink, por ejemplo.

En cuanto a los principales conceptos manejados en Modelica, cabe indicar que no existe un metamodelo explícito que especifique el lenguaje, si bien en la literatura existen diversos trabajos que proponen metamodelos de Modelica como [101], principalmente orientados a dar soporte a transformaciones entre modelos descritos en diferentes lenguajes. Algunas de las construcciones más características son: las clases (o especializaciones como modelo o bloque) que dan soporte a los modelos de simulación; las funciones que se emplean para definir algoritmos reutilizables; los

paquetes que permiten estructurar las librerías; o los modelos parciales que se emplean como base para ser extendidos y definir modelos más complejos. Además, Modelica permite trabajar con un conjunto de ecuaciones diferenciales, algebraicas y discretas. Existen un conjunto de restricciones que únicamente aseguran que un modelo está completamente definido, así como mecanismos de verificación que aseguran que el modelo puede ser simulado con el conjunto de relaciones matemáticas y las condiciones iniciales definidas. Sin embargo, la completitud y la adecuación para la ejecución no aseguran ni que el modelo sea válido (consistente con aquello que se desea simular, cumpliendo los requerimientos) ni la obtención de resultados tras la ejecución del experimento.



(a) (b)

Figura 3.5. Capturas del entorno de simulación OpenModelica: código texto (a) y vista gráfica (b).

En lo referente a los entornos de modelado y simulación con Modelica, cabe indicar que existen numerosas herramientas comerciales que soportan este lenguaje. Algunas de las más conocidas son Dymola (Dassault Systèmes) [102], Simplorer (ANSYS) [103], solidThinking Activate (Altair) [104], Wolfram SystemModeler (Wolfram) [105] o Simcenter Amesim (Siemens) [106]. Sin embargo, en esta investigación se ha optado por seleccionar como herramienta de trabajo el principal entorno de simulación Modelica de software libre, llamado OpenModelica [107]. Esta aplicación es un entorno de código abierto que da soporte al modelado, compilación y simulación de Modelica, destinado tanto a la investigación y la docencia como a su uso a nivel industrial. Este entorno permite la creación y edición de modelos de simulación a nivel gráfico y en formato texto, su compilación y ejecución, y la visualización de resultados de los experimentos. Además, en las versiones más recientes de OpenModelica no solo soporta lenguaje Modelica (incluidas ecuaciones, algoritmos, manejo de eventos, funciones y paquetes), sino que también se incluye un complemento Modelica Eclipse para desarrolladores avanzados, explorando la integración Modelica-UML en Eclipse como ejemplo de las actividades de desarrollo en curso, una cuestión de gran interés para esta investigación [108]. La Figura 3.5 muestra dos capturas de pantalla de

OpenModelica, ejemplificando tanto la vista durante la edición del modelo textual (a) como la vista gráfica del modelo (b).

3.2. Modelado del prototipo

Para abordar este trabajo, inicialmente se realizó un estudio del sistema referente, que en este caso es un sistema de ensamble multietapa (MAS), identificando sus principales características y su estructura funcional y lógica. En líneas generales, la investigación pone el foco en el análisis de Líneas de Montaje Flexibles y Modulares [109], unos sistemas de ensamblaje que se caracterizan por una gran flexibilidad, reconfigurabilidad, reactividad e inteligencia, entre otros muchos principios de diseño propios de los sistemas de fabricación avanzada. Entre estos principios destaca la necesidad de una integración flexible de los lazos de control de calidad, lo cual motivó la construcción del modelo de simulación que permitiera validar estrategias de control enfocadas a prevenir la generación y propagación de defectos geométricos.

Sin embargo, en esta etapa de investigación, la construcción del prototipo se limita a tratar el caso de un MAS para el montaje de un solo producto y con un proceso multi-estación lineal, sin rutas alternativas. La línea de montaje consta de diversas estaciones de montaje que pueden tener capacidades de medición en proceso, y una estación de inspección para la medición de los montajes finales con el fin de establecer su aceptación o rechazo en función de las especificaciones definidas. Cabe indicar que en este primer trabajo no se modelizan los procesos de transporte entre estaciones, pero sí se modelizan almacenes (buffers) que permiten regular el flujo de materiales. Dos componentes fluyen hasta la máquina, en la que se posicionan mediante un juego de localizadores que pueden ajustarse manual o automáticamente, y se procede a su ensamble, que para el supuesto se considera de tipo permanente (por ejemplo, una soldadura).

En relación al sistema de control incluido en el MAS, el sistema está dotado de una lógica de control tanto para la logística (flujo de materiales y órdenes de producción) como para el aseguramiento y mejora de la calidad geométrica. Aunque el sistema de simulación debe dar soporte al análisis de sistemas complejos, para el caso lineal considerado el control logístico apenas tiene influencia, por lo que la simulación de los elementos de control se centra en el tratamiento de la calidad geométrica. En este sentido, tanto los resultados de la inspección como de las mediciones en las estaciones de ensamblaje se utilizan para calcular, de acuerdo con diferentes algoritmos de optimización, la corrección de la posición de los localizadores para mejorar el índice de calidad para los ensambles finales. Para más información, se sugiere consultar [30].

A partir de estas bases, se establecieron unos requisitos para el sistema de simulación, que debe ser modular, flexible y escalable, aprovechando la capacidad de Modelica para construir modelos ejecutables a partir de librerías de bloques reutilizables. Estos requisitos están orientados a posibilitar una futura extensión del modelo, enriqueciendo la simulación para incorporar los modelos de comportamiento de un sistema de menor escala, como sistemas multi-eje, sistemas de configuración de localizadores con ajuste automático, etc.

Para dar respuesta a estos requisitos, se desarrollaron diversos modelos descriptivos SysML a través de los cuales definir las principales características estructurales y comportamentales tanto del sistema de simulación como de los experimentos. Posteriormente, este principio de solución se implementó en Modelica para obtener un sistema de simulación ejecutable, desarrollando diversas librerías y casos de estudio, ejecutando experimentos que permitieran validar el modelo propuesto. La presente sección recoge los principales resultados referentes a dichas tareas de modelado del sistema de simulación, dejando el caso de estudio para ser tratado en una sección independiente (sección 3.3).

3.2.1. Diseño del sistema de simulación: modelos descriptivos

El siguiente paso es establecer el metamodelo para el sistema de simulación de una línea de montaje. Este metamodelo debe definir la estructura estática de bloques y el comportamiento de dichos bloques, los cuales servirán como elementos constructivos para la creación del correspondiente modelo de simulación ejecutable para una línea de ensamblaje específica.

Así pues, se define el metamodelo que describe los conceptos vinculados con el modelado de sistemas de simulación para el análisis de un MAS genérico, cuya estructura es representada mediante un BDD en la Figura 3.6. En este diagrama se representa el sistema de simulación del MAS (MAS_simulation) y todos los componentes que lo forman, con los que mantiene relaciones de composición. Conviene indicar que en el nombre de muchos bloques se añade el sufijo "_sim" para poner énfasis en que estos bloques representan entidades del sistema de simulación, y no del sistema referente. Caso aparte es la relación de agregación que apunta al bloque MAS, bloque que sí representa el modelo del sistema referente. Así pues, el MAS_simulation está formado por diversos elementos que representan la simulación de la llegada de órdenes y materiales (OrderArrival_sim), diversas (1..*) estaciones de trabajo de ensamble (AssemWS_sim), una estación de inspección final (InspecWS_sim), el final del flujo de materiales (OrderSink_sim) y la unidad de control central (Control_sim). El diagrama también permite mostrar la descomposición de los bloques AssemWS_sim y InspecSW_sim, dando soporte a la simulación tanto de la llegada y almacenamiento de materiales como de la máquina que desempeña el proceso principal de cada estación (ensamble o inspección).

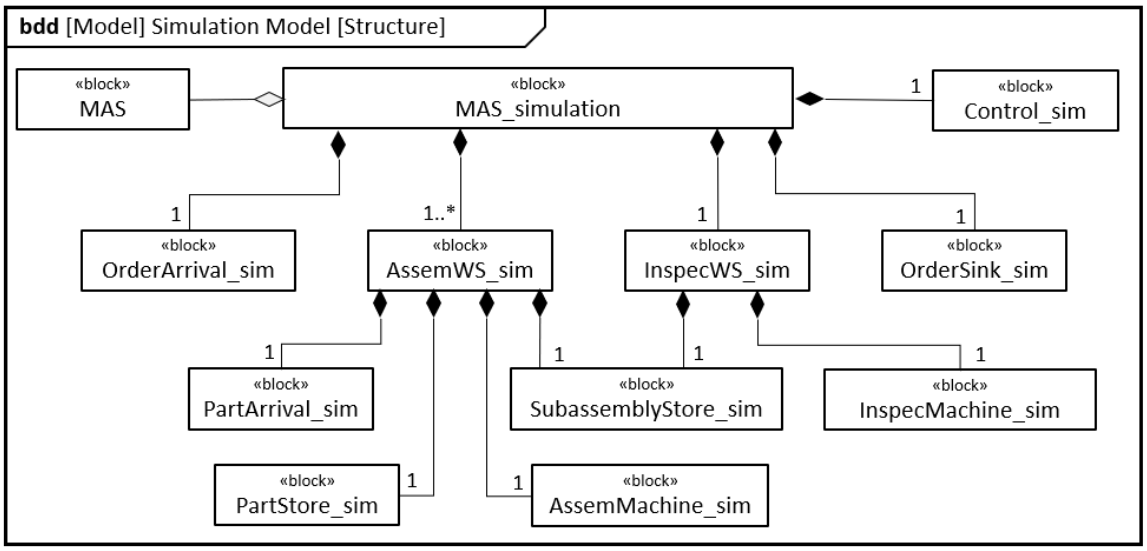


Figura 3.6. BDD del metamodelo de sistemas de simulación [30]

A partir de los conceptos manejados en el metamodelo (nivel M2), es momento de centrar el discurso en el desarrollo de modelos a nivel M1, tanto librerías de construcciones reutilizables como modelos de simulaciones concreta, construidos mayoritariamente a partir del uso de elementos de las librerías propuestas.

En primer lugar, se planteó el diseño de una librería con los elementos necesarios para la construcción de modelos de simulación. Sin embargo, con el fin de dar respuesta a los requisitos de modularidad y flexibilidad, finalmente se decidió desarrollar dos librerías diferenciadas:

- *Basic Library for MAS Simulation (BL4MAS_Sim)*. Esta librería contiene un conjunto de construcciones que permiten definir un sistema de simulación de MAS, centrado en la simulación del flujo logístico y el comportamiento discreto del sistema según el formalismo DEVS.

- *Extended Library for MAS and Variation Flow Simulation (EL4MAS&VF_Sim)*. Los elementos definidos en esta librería extienden/especializan los anteriores para, además de la simulación DEVS, incorporar las estructuras y comportamientos propios de la variabilidad geométrica y su propagación según la técnica Stream of Variation (SoV).

La estructuración de las librerías en una básica y una extendida facilita la posibilidad de que, en trabajos futuros, se desarrollen librerías alternativas basadas en BL4MAS_Sim y que incluyan otros análisis, ya sea para simular la propagación de la variabilidad según otras técnicas, o para considerar otro tipo de análisis, como consumos eléctricos, emisiones, etc.

Definidas las librerías, el siguiente paso es el diseño de modelos de simulación concretos, conformes al metamodelo planteado y a las librerías propuestas (fundamentalmente EL4MAS&VF_Sim). Con el fin de ejemplificar esta tarea, la Figura 3.7 muestra un IBD para el supuesto de un sistema de simulación que analiza un MAS compuesto por tres etapas de ensamble y una etapa de inspección final. Además, se

muestran tanto los componentes de inicio y final del flujo logístico como la unidad de control que monitoriza todos los procesos. Cabe resaltar que todos los tipos a los que apuntan estas propiedades son de la librería extendida EL4MAS&VF_Sim, unos bloques identificados con el sufijo "_VF" (Variation Flow) para indicar que incluyen el modelado de las desviaciones geométricas y su propagación.

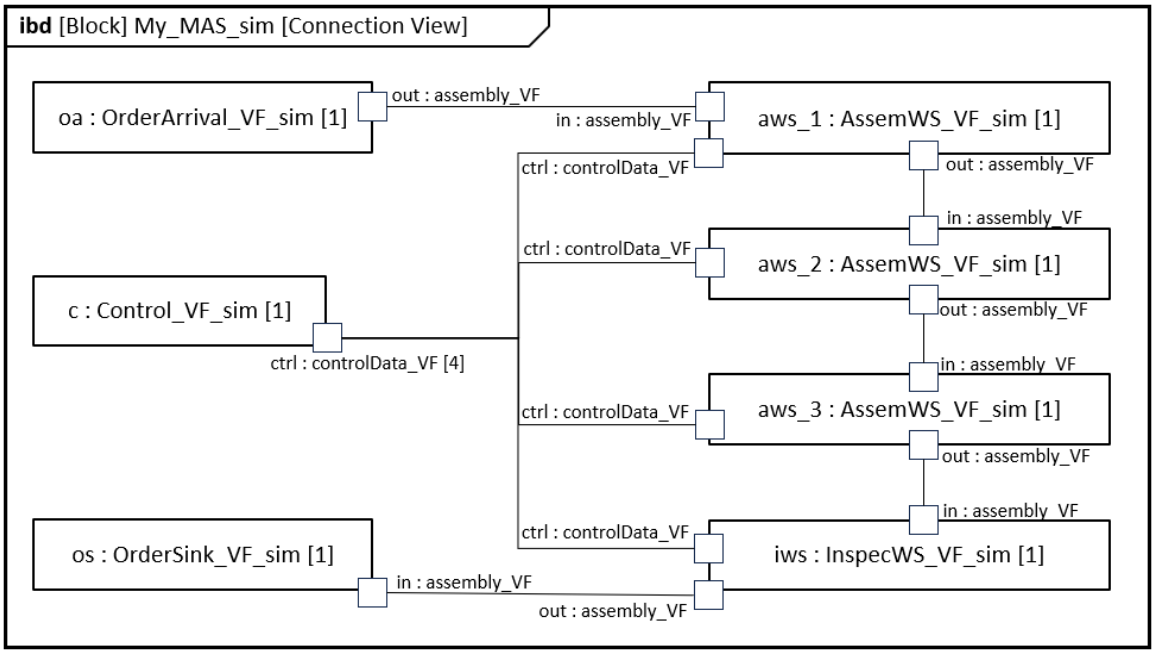


Figura 3.7. IBD de un sistema de simulación de un MAS con cuatro estaciones de trabajo [30]

Más allá del modelado estructural, otro aspecto fundamental del modelo del sistema de simulación es su comportamiento. Dado que el comportamiento de los componentes del sistema de simulación es de naturaleza discreta, se modelan utilizando el formalismo DEVS [110], en el que se distinguen dos tipos de elementos: entidades con un comportamiento indivisible (DEVS atómico), y entidades con comportamientos complejos (DEVS acoplado). Estos principios, trasladados al modelado con SysML, se traducen en definir un comportamiento propio (una máquina de estados como Classifier Behavior) en los bloques con un comportamiento atómico. Por su parte, los elementos más complejos (como el sistema de fabricación en su conjunto) presentan un comportamiento emergente, equivalente a un DEVS acoplado definido a partir de los comportamientos atómicos y las interacciones entre éstos, ya que los eventos de entrada y salida manejados en los DEVS atómicos están definidos por señales o datos que fluyen a través de los conectores que relacionan las diversas partes del sistema.

Con el fin de ejemplificar este comportamiento, la Figura 3.8 muestra un diagrama de máquina de estados de SysML, en concreto la máquina de estados definida para el bloque Assembly_Process, que es uno de los elementos reutilizables disponibles en la librería BL4MAS_Sim. Como se puede observar, esta máquina de estados incluye, además del nodo de inicio, los estados Idle (compuesto por otros estados), Setup, Assembling, Repairing y Blocked. El diagrama también muestra las transiciones entre estados, indicando los eventos que actúan como disparadores (*triggers*) de la

transición. Por ejemplo, la transición entre Assembling y Repairing se desencadena por el evento "failure_event", que indica que se ha producido un fallo en la estación de ensamblado y que se debe detener el trabajo para proceder a la reparación del equipo. Además, en ocasiones también es necesario definir guardas, que son condiciones que deben cumplirse para que una determinada transición se ejecute, más allá de recibir el triger correspondiente. Por ejemplo, al terminar el trabajo de ensamblado se produce un evento (time_event). Sin embargo, la transición hacia el siguiente estado está condicionada en función de si la siguiente etapa está disponible para recibir material (guard=[next_ready==true]) o si está ocupada (guard=[next_ready==false]). En caso de cumplirse la primera condición, el flujo de materiales prosigue hacia la siguiente etapa, quedando la actual disponible (estado Idle), mientras que si está ocupada se produce un bloqueo (estado Blocked) hasta que se produce el evento que marca la disponibilidad de la siguiente etapa.

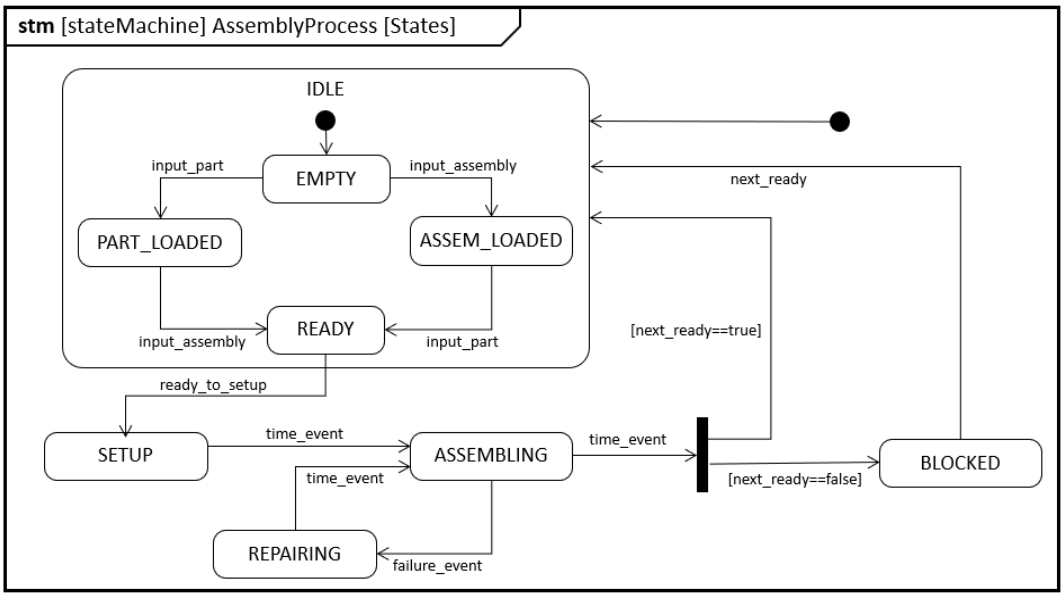


Figura 3.8. Diagrama de máquina de estados del bloque AssemblyProcess [30]

Una vez modelado el sistema de simulación, la siguiente tarea es el modelado de las simulaciones a ejecutar, es decir, de los experimentos que se pretenden llevar a cabo. Para ello, no solo es necesario disponer del sistema de simulación apropiado para el análisis de un MAS concreto y, por tanto, con información de diseño sobre el sistema referente, sino que además es fundamental establecer un conjunto de datos que definen el escenario de simulación. En este sentido, es necesario distinguir entre los datos estáticos que solo cambian en los diferentes experimentos (por ejemplo, la capacidad máxima de un almacén) y aquellos datos que pueden sufrir cambios dinámicos (gobernados por un comportamiento definido) durante cada experimento, por ejemplo, el desgaste de los localizadores en una etapa. Adoptando la terminología matemática, en este documento se emplea el término "parámetro" para referirse a los datos estáticos, mientras que los segundos se denominan "variables", cuyo valor inicial puede estar definido a través de un parámetro. De este modo, para la completa definición de los experimentos es necesario definir valores concretos para los parámetros considerados en el modelo.

3.2.2. Implementación del sistema de simulación: modelos ejecutables

Esta sección contiene una descripción sintetizada de la implementación del modelo en lenguaje Modelica para obtener modelos de simulación ejecutables. Como se ha comentado anteriormente, durante el diseño del sistema de simulación se decidió crear dos librerías, una para soportar la simulación del flujo logístico (BL4MAS_Sim) y una extensión de ésta para incluir la simulación del flujo de características de calidad (EL4MAS&VF_Sim). A continuación, se dedica un apartado a comentar las particularidades de cada uno de estas librerías, aunque para los modelos propuestos en esta investigación serán empleados principalmente las construcciones extendidas que soportan la simulación de la variabilidad geométrica y su propagación.

A. Librería BL4MAS_Sim (simulación DEVS del MAS)

La librería BL4MAS_Sim se implementa en Modelica como un paquete que contiene las construcciones (clases, conectores, funciones, etc.) necesarias para simular el flujo de materiales y órdenes de producción a lo largo de las etapas tanto de ensamblaje como de inspección, soportando diferentes tareas. A continuación, se nombran algunas de las principales clases definidas y las funciones que soportan.

- *Simulación_Escenario*. Contiene los parámetros iniciales que definen el escenario de simulación, agrupados en:
    - *Assembly_Plan*. Recogen la secuencia de las etapas de montaje y/o inspección. Para cada etapa se indica: la estación en la que se lleva a cabo, el subconjunto y parte incorporada, así como los parámetros para simular el tiempo del proceso a través de una distribución normal (media y desviación estándar).
    - *Assembly_Identification*. Identifican cada uno de los tipos de subensamble utilizados en las distintas etapas, indicando los tipos de pieza incluidos.
    - *Otros parámetros*. Contienen información sobre la llegada de órdenes, llegada de piezas, tiempo de configuración, tasa de fallo, capacidad de almacenamiento, etc.

- *Scenario_Dynamics*. Se utiliza para modelar el comportamiento dinámico de aquellos datos que presentan una variación temporal, ya sea mediante funciones continuas o cambios discretos. Por ejemplo, para considerar un aumento gradual en el tiempo de procesamiento, o un cambio puntual (repentino) en la distribución del tiempo de llegada de las piezas debido al cambio de proveedor.

- *Assembly_Station e Inspection_Station*. Son clases que representan una estación de trabajo de ensamblaje e inspección, respectivamente. Son bloques complejos con un comportamiento DEVS acoplado, compuestos de bloques individuales con comportamiento DEVS atómico. En la Figura 3.9 se muestra la estructura interna de una Assembly_Station, compuesta principalmente por las clases que se describen a continuación:

- *Part_Generator / Assembly_Generator*. Simulan el proceso de llegada de piezas y ensambles, respectivamente. Part_Generator está representado en color azul en la Figura 3.9.
- *Part_Store / Assembly_Store*. Simulan el almacenamiento de piezas o ensambles a la entrada para una WorkstationProcess. Estos almacenes utilizan una estrategia FIFO, pero pueden incluir otras estrategias. Están representados en rojo y amarillo, respectivamente, en la Figura 3.9.
- *Assembly_Step / Inspection_Step*. Simulan el propio proceso caracterizado por un formalismo DEVS atómico. Assembly_Step está representado en color verde en la Figura 3.9.

- *Monitor&Control*. Esta clase recopila datos de cada WorkstationProcess, lo que le permite ejecutar un análisis estadístico para estimar los parámetros de rendimiento de la productividad. Además, esta clase se puede especializar para llevar a cabo acciones de control.

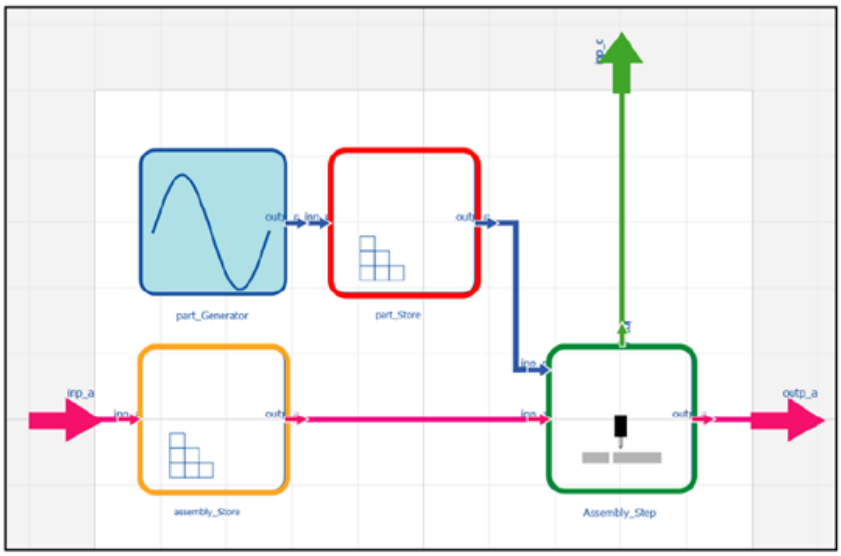


Figura 3.9. Estructura del bloque AssemblyStation. Interfaz gráfica de OpenModelica [30]

La Figura 3.9 también permite mostrar gráficamente un ejemplo de los tres tipos de conectores considerados en el paquete LFS, dando soporte a diferentes flujos de datos entre bloques. Estos conectores son:

- *Assembly_connection*. Contiene los datos relacionados con el flujo de ensamblajes, incluyendo el orden de montaje, las piezas contenidas y los tipos de estas piezas, además de variables booleanas para indicar eventos externos que afectan a los bloques por los que fluyen los ensambles. Este tipo de conector está representado por una línea rosa en la Figura 3.9.

- *Part_connection*. Contiene los datos relacionados con el flujo de piezas que se agregan en cada etapa del ensamblaje, incluyendo la identificación de la pieza y su tipo. Este conector está representado por una línea azul en la Figura 3.9.

- *Control_connection*. Contiene los datos utilizados para monitorizar y controlar la estación de trabajo. Este flujo de datos es bidireccional, de la estación al

control (monitorización) y viceversa (acciones de control). Este conector está representado por una línea verde en la Figura 3.9.

B. Librería EL4MAS&VF_Sim (Simulación DEVS+Variation Flow)

Como se mencionó anteriormente, la librería EL4MAS&VF_Sim se implementa en Modelica como un paquete que contiene construcciones que extienden/especializan los elementos definidos en BL4MAS_Sim para incorporar la simulación del flujo de características de calidad (limitado a un estudio 2D), y de forma más concreta utilizando la técnica SoV que ha sido descrita en el apartado 2.2.3 de este documento. Para ello, estas construcciones extendidas incorporan estructuras y mecanismos relacionados con el modelado de las variaciones geométricas y la simulación de su propagación a través de las etapas del MAS.

Por una parte, los conectores definidos en BL4MAS_Sim se amplían en EL4MAS&VF_Sim para incorporar datos adicionales como el vector de movimiento de desviación (Deviation Motion Vector - DMV) de la técnica SoV, que contiene los errores de traslación y orientación, y se transmite junto con el resto de información de la pieza o ensamblaje. El conector Control_connection también se extiende para incluir variables relacionadas con las características medidas en cada etapa de inspección, así como información sobre las posiciones óptimas de los localizadores calculadas de acuerdo con diferentes lógicas de control.

Por su parte, las clases definidas en BL4MAS_Sim también se extienden en EL4MAS&VF_Sim incorporando datos que se describen brevemente a continuación:

- *VFS_Simulation_Scenario y VFS_Scenario_Dynamics*. Incorporan datos adicionales del plan de proceso (como el esquema de localización para cada etapa), datos geométricos sobre las geometrías nominales, la posición nominal de los localizadores y sus respectivas variaciones (modeladas por una distribución normal con media y desviación estándar), o incertidumbres de medición de cada etapa con un proceso de medición, entre otros muchos datos.

- *VFS_Assembly_Station y VFS_Inspection_Station*. Incorporan propiedades y funciones necesarias para calcular, según la técnica SoV, las transformaciones del DMV en cada estación. En concreto, las estaciones de ensamblaje toman en consideración las desviaciones de las piezas/ensambles entrantes y las desviaciones introducidas en cada etapa como consecuencia de los errores en los localizadores. Por su parte, para las etapas de inspección o de ensamble con medida, SoV también incluye el cálculo de desviaciones en las características de calidad medidas debido a las incertidumbres de los equipos empleados.

- *VFS_Basic_Monitor&Control*. Incorpora la monitorización del flujo de características de calidad y mecanismos para soportar diferentes bucles de control de calidad. En concreto, para este prototipo se han desarrollado dos especializaciones para dar soporte a dos lógicas de control alternativas: FeedForward y FeedForward&Backward, comentadas en el punto 2.3.3 de este documento.

3.3. Caso de estudio

Este caso de estudio se desarrolla para validar la propuesta (tanto el metamodelo como las librerías desarrolladas) mediante la implementación de un modelo de simulación ejecutable, ejemplificando además el potencial de uso de este tipo de modelos de simulación con el análisis comparativo de diversas alternativas de control y sus efectos desde las perspectivas de calidad geométrica y productividad. Conviene mencionar que este caso fue publicado en [30], y que en este documento se recoge una síntesis del modelo y los resultados obtenidos, invitando al lector a consultar la fuente original para un mayor detalle.

El caso presentado a continuación se basa en el caso de estudio de [71], un proceso de ensamblaje de un producto 2D compuesto por cuatro piezas, el cual se ensambla en un proceso de tres etapas y se inspecciona al final del proceso, tal y como se ilustra en la Figura 3.10.

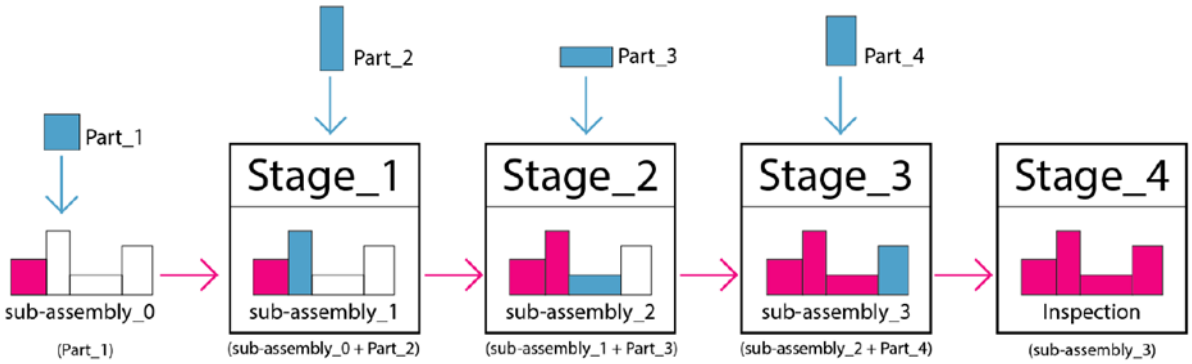


Figura 3.10. Esquema del proceso analizado en el caso de estudio [30].

Para analizar este MAS, se consideran tres escenarios diferentes relativos al control de la calidad geométrica del producto. Así, en uno de los escenarios no se aplica ningún tipo de acción correctiva, mientras que en los dos restantes se aplican respectivamente las lógicas de control FeedForward y FeedForward&Backward, representadas gráficamente en la Figura 3.11 para el proceso considerado.

Además de los escenarios descritos, se plantean dos supuestos alternativos en los que emplear actuadores manuales o automáticos en los localizadores, modificando los tiempos de ajuste y, por tanto, afectando a la productividad del sistema. Por tanto, combinando las condiciones de trabajo, se consideran en total cinco casos alternativos, que son:

- Caso 1: Simulación sin acciones correctivas.
- Caso 2: Simulación del escenario 1 empleando actuadores manuales.
- Caso 3: Simulación del escenario 1 empleando actuadores automáticos.
- Caso 4: Simulación del escenario 2 empleando actuadores manuales.
- Caso 5: Simulación del escenario 2 empleando actuadores automáticos.

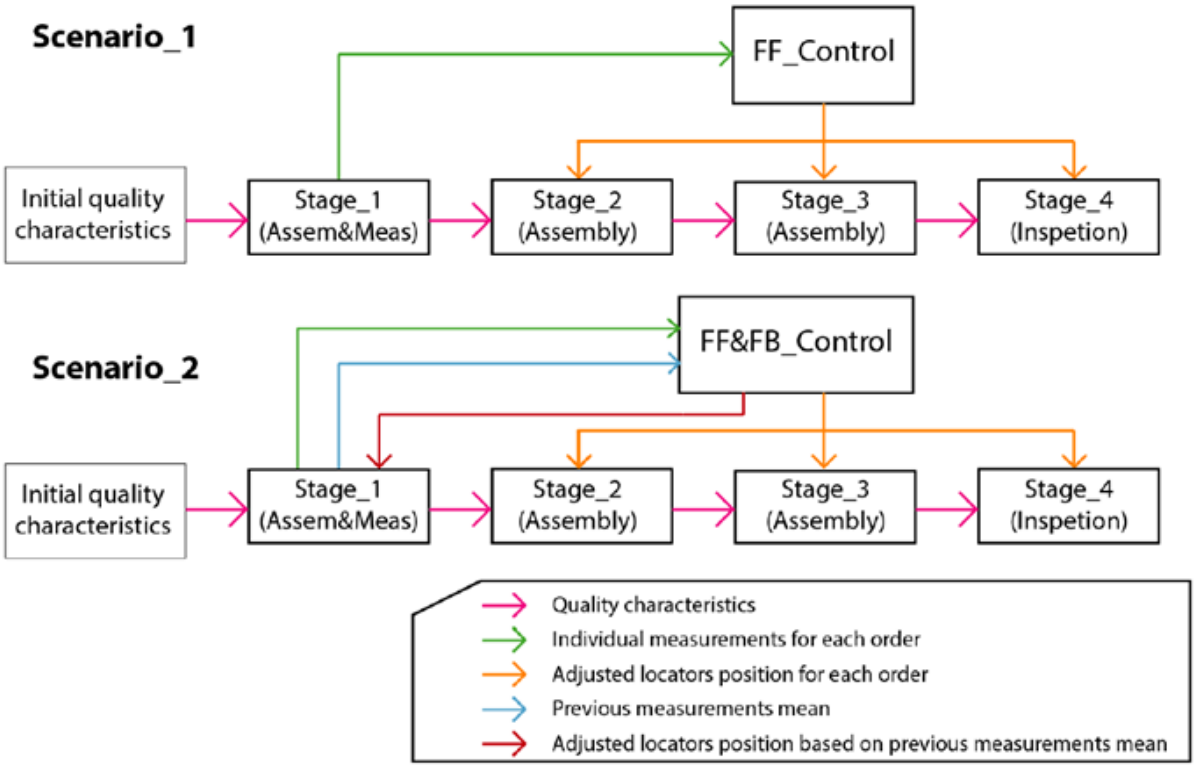
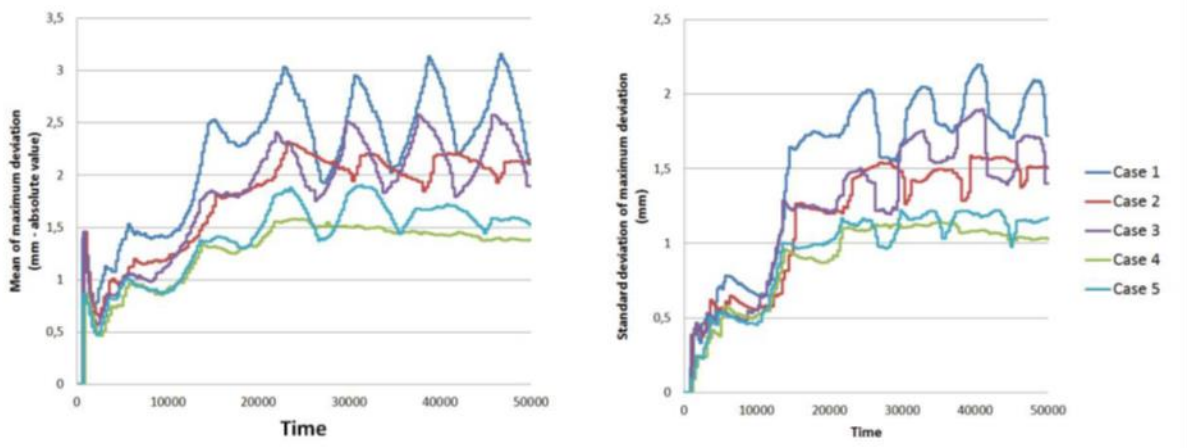


Figura 3.11. Esquema de monitorización y control en los escenarios 1 y 2 [30].

Tanto los principales parámetros considerados como el análisis detallado de los resultados obtenidos en la simulación están disponible en [30], recuperando a continuación únicamente algunas de las principales conclusiones.



(a) (b)

Figura 3.12. Análisis estadístico de las desviaciones máximas detectadas en la estación final para cada uno de los casos considerados. A) Media; B) Desviación típica.

En relación a la calidad geométrica, la Figura 3.12 muestra gráficamente el análisis estadístico de la máxima desviación detectada en la inspección final, representando tanto su valor promedio (a) como su desviación típica (b). A la vista de estos resultados, los casos en los que se ha aplicado un control (casos 2-5) mejoran notablemente los resultados tanto en media como en desviación típica, siendo especialmente destacable

el caso 5, en el que se obtienen los valores mínimos y con un comportamiento más estable.

En relación a la productividad, como era de esperar, la introducción de operaciones adicionales de medición y de procesos de ajuste de los localizadores provoca una reducción de las unidades producidas en el mismo tiempo simulado, como se muestra en la fila "Prod./h" de la Tabla 3.1. Sin embargo, si combinamos estos datos con los resultados de calidad se puede comprobar que el número total de productos terminados que cumplen con las especificaciones es notablemente mayor en el Caso 3 (Control Feed-Forward con actuadores automáticos), tal y como se indica en la fila "P.T. OK" de la Tabla 3.1. Sin embargo, sería interesante desarrollar otro tipo de análisis como el estudio de costes para cruzarlo con los resultados obtenidos y poder realizar un análisis multidominio más completo.

Tabla 3.1. Resultados de productividad. Indicadores empleados: productos terminados (P.T.); *throughput* o tasa de productos terminados por hora (Prod./h); productos terminados que cumplen especificaciones (P.T. OK), productos terminados que cumplen especificaciones

	Caso 1	**Caso 2**	**Caso 3**	**Caso 4**	**Caso 5**
P.T. (uds.)	**214**	164	203	153	180
Prod./h	**15.41**	11.81	14.61	11.01	12.96
P.T. OK (uds.)	155	132	**164**	136	156
Prod.OK/h	11.17	9.51	**11.81**	9.80	11.24

3.4. Conclusiones de la experiencia

Esta experiencia en el dominio del modelado y la simulación de sistemas de fabricación, y particularmente en el uso de Modelica, ha permitido detectar una serie de fortalezas y debilidades, tanto de los modelos desarrollados como de la metodología adoptada, que se describen a continuación.

Por una parte, se ha podido comprobar la ventaja que ofrece el uso de lenguajes orientados a objetos, como Modelica. El encapsulamiento de elementos y los mecanismos de especialización y redefinición de propiedades de estos lenguajes ofrecen al modelador una gran versatilidad a la hora de construir elementos reutilizables, un aspecto clave en la definición de librerías que faciliten el modelado al usuario final. Además, como es obvio y remarca la literatura, la integración con lenguajes descriptivos con esta orientación a objetos se verá, en principio, facilitada. Adicionalmente, la experiencia ha puesto de manifiesto grandes similitudes entre los conceptos básicos de Modelica y SysML, lo cual supone una prometedora ventaja a la hora de explorar la integración o la transformación de modelos de un lenguaje a otro.

Además, los resultados de los experimentos realizados [30] y [26], presentados parcialmente en este capítulo, confirman la elección del lenguaje Modelica como una

buena alternativa para el modelado de simulaciones multidominio y, en concreto, para las simulaciones planteadas, en las que se integra el análisis de la calidad y la productividad de sistemas de fabricación. Por lo que respecta al entorno de trabajo utilizado en la experiencia, cabe indicar que, OpenModelica ha demostrado ser válida para este tipo de tareas. Si añadimos que se trata de una aplicación de software libre, la utilización de la misma para la implementación de los desarrollos resultantes de la investigación está completamente justificada.

Sin embargo, no todos los aspectos relacionados con el uso del lenguaje y la herramienta de modelado han sido positivos. La complejidad de los modelos desarrollados y los limitados mecanismos de comprobación disponibles en estos entornos obligan a que el usuario deba mantener una gran atención durante el modelado, porque el riesgo de introducir fallos e inconsistencias es elevado. Además, el tiempo requerido para la identificación y corrección de errores también es considerable, y depende mucho de la destreza o nivel de conocimiento del usuario con el lenguaje, el entorno y el propio dominio bajo análisis.

Estas dificultades se deben en gran medida a que en estos entornos no se comprueba la consistencia en base a semántica específica del dominio, puesto que la validación del modelo se limita a comprobar la sintaxis propia de Modelica, careciendo de mecanismos que permitan la inclusión de otro tipo de consideraciones sintácticas o semánticas vinculadas con el dominio de interés. Esto implica un gran riesgo, porque se pueden definir modelos que, aun siendo compilables y ejecutables, carezcan de sentido desde el punto de vista del análisis. Tras haberse constatado la importancia de esta problemática, se propone investigar todos aquellos enfoques (mecanismos) de modelado que permitan asegurar la consistencia de los modelos de simulación, con el objetivo de automatizar, en lo posible, comprobaciones que incluyan semánticas propias del dominio bajo análisis.

En relación al tratamiento de las desviaciones y su propagación, cabe indicar que la adopción del Stream of Variation ha permitido resolver los casos propuestos de una forma eficiente, dando soporte a los cálculos con un reducido coste computacional. Sin embargo, su adopción tiene ciertas limitaciones que se resumen a continuación. Por una parte, puesto que está basado en el espacio de estados, SoV permite resolver sistemas lineales de forma muy compacta mediante el uso de ecuaciones matriciales de primer orden. Sin embargo, en el caso de abordar problemas con relaciones no lineales entre las variables es necesaria una linealización del problema, realizando una serie de simplificaciones o aproximaciones que dan lugar a un error de cálculo.

Por otra parte, la adopción de SoV implica seleccionar el mínimo número de variables que permitan definir el estado, que en el caso del análisis de las desviaciones geométricas son parámetros que permiten caracterizar la posición y orientación de la pieza (en el caso 2D, la posición en X e Y y el giro en Z). Atendiendo a esta circunstancia, surge el problema de que estos parámetros no tienen por qué coincidir con los que se suelen utilizar para establecer un determinado esquema de acotado en el que, además, tienen un papel fundamental las geometrías tomadas como origen (datums) para el establecimiento de las especificaciones GD&T (cotas). Por este motivo, la forma en la

que se relacionan los resultados del análisis basado en SoV con las especificaciones GD&T de las piezas resulta muy compleja. Por ejemplo, si nos fijamos solo en una de las piezas consideradas en el ensamble (Part_1), los parámetros geométricos en base a los que se establecería la formulación SoV (X, Y y γ) serían los que se muestran en la Figura 3.13, que puede que no se correspondan con las especificaciones del acotado, p.e. funcional, de la pieza. Por lo tanto, fuera cuál fuera el acotado establecido, en la formulación SoV siempre se utilizarían los mismos parámetros, perdiéndose el significado que se atribuye y corresponde a cada acotado. Para la pieza Part_1, atendiendo a como la pieza participa en la funcionalidad del ensamble y del producto en que se inserta, se podrían haber considerado diferentes acotados, como los mostrados en la Figura 3.14, en los que se acota la ranura respecto de las referencias A y C (a) o respecto de las referencias A y B (b).

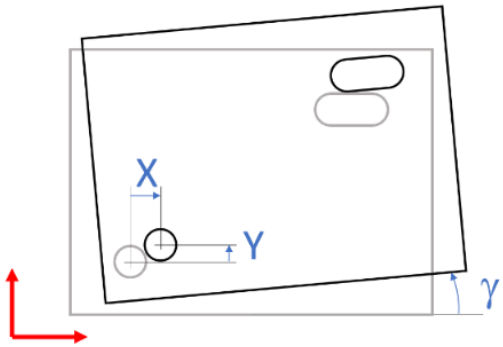


Figura 3.13. Parámetros de la pieza Part_1 en la formulación SoV.

Con el fin de superar estas limitaciones, se explora el uso del modelo TTRS y la adopción de sus conceptos para el modelado de la geometría, tanto nominal como desviada. Como se tratará en los siguientes capítulos, la compatibilidad de TTRS con los estándares GD&T permite abordar la especificación de las piezas, trasladando sin ambigüedades el esquema de acotado a una formulación matemática adecuada para la simulación. Además, TTRS también se emplea para establecer las relaciones de ensamble entre geometrías de piezas diferentes, permitiendo abordar análisis de ensambles sin adoptar ninguna hipótesis o simplificación.

Finalmente, también se ha podido demostrar que tanto el lenguaje Modelica como la forma en que se han construido los modelos propuestos se adecuan a los requisitos del análisis planteado, permitiendo la comparación de diferentes estrategias de control estudiadas, constatándose que estas influyen de forma diferente en la productividad del sistema y en la calidad geométrica del producto. Por lo que respecta a la forma/estructura de los modelos, cabe indicar que están formados por unos bloques básicos que simulan los recursos y que están conectados a través de puertos para soportar el intercambio de los datos, tanto del flujo logístico como del de desviaciones. Otro bloque fundamental es el que emula el control y ejerce las funciones de monitorización y toma de decisiones, que se conecta con el resto de recursos emulados para obtener datos de su comportamiento y proporcionar valores vinculados con las acciones correctivas establecidas según la estrategia considerada en cada experimento. Puesto que el resultado ha sido satisfactorio, se decide adoptar esta estructura en la construcción de modelos de simulación, por lo que para el resto de la investigación el

foco se va centrar en la mejora de los aspectos relacionados con consistencia y el modelado TTRS anteriormente comentados.

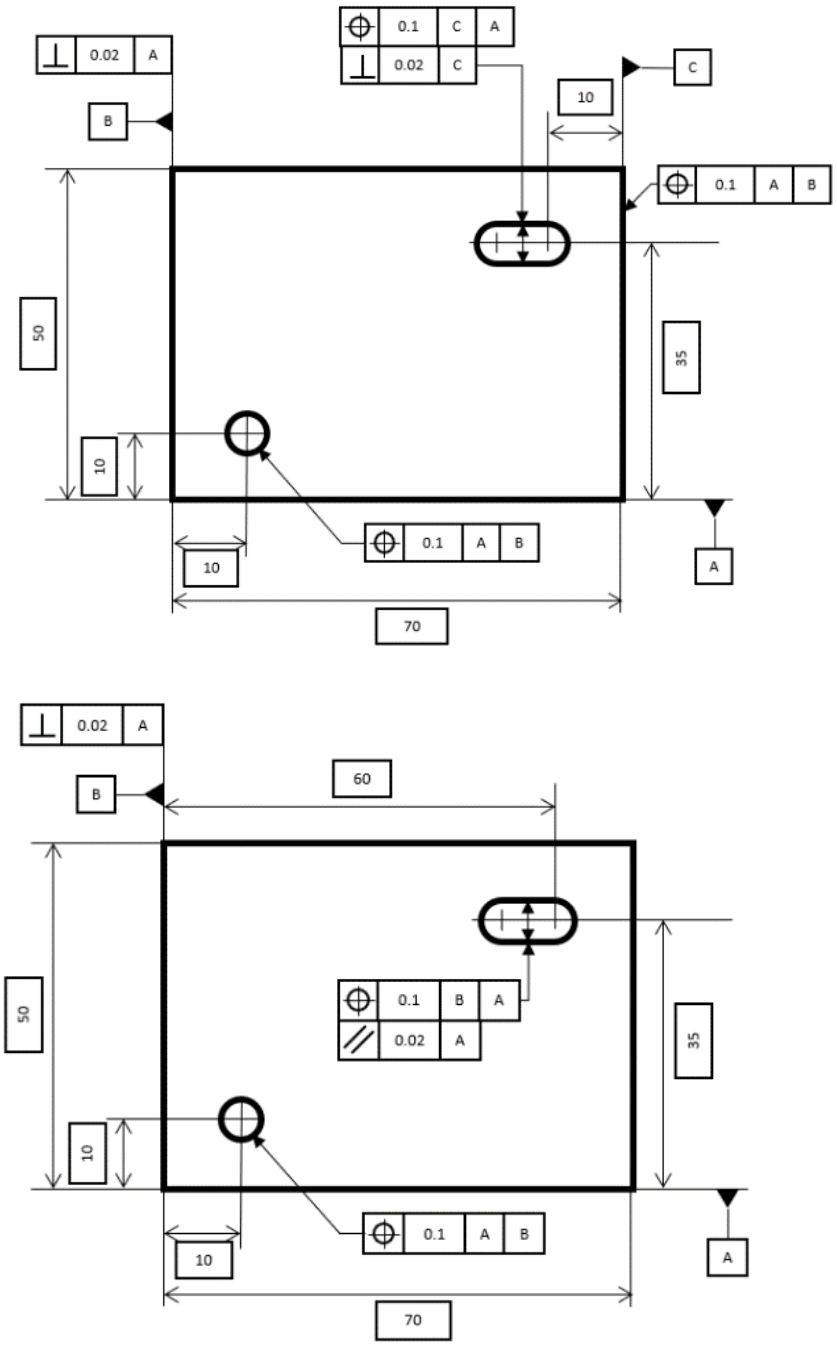


Figura 3.14. Acotados diferentes para la pieza Part_1.

4. Metodología para el modelado consistente de simulaciones

De forma general, una metodología es entendida como la definición de un procedimiento en el que se hace uso de un conjunto de lenguajes y herramientas (entornos, recursos, etc.) para resolver un problema, idealmente derivado de un marco teórico y desde un punto de vista en particular. En la literatura existen un amplio número de propuestas metodológicas alineadas con el enfoque MBSE. Cuenta de ello se da en [40] o [111], trabajos en los que se analizan algunas de las propuestas metodológicas más extendidas, como Harmony for Systems Engineering (IBM), Object Oriented Systems Engineering Method (INCOSE) o Model-Based Systems Engineering Methodology (Vitech), entre muchas otras.

En el caso de esta investigación, el problema planteado se centra en el desarrollo consistente de modelos de simulación ejecutables, y en particular de modelos que permitan simultáneamente la simulación basada en eventos discretos de procesos de fabricación multietapa y el análisis de las desviaciones geométricas en el producto, incluyendo su propagación a lo largo de las diversas etapas de fabricación consideradas. Así pues, se pretende definir una metodología que permita asegurar la correcta construcción de estos modelos de simulación tomando en consideración la semántica propia de los dominios involucrados.

Teniendo en cuenta el desafío planteado, las siguientes secciones se dedican a la descripción detallada del marco de la propuesta, los requisitos considerados y a la descripción de la metodología, haciendo especial énfasis en el procedimiento establecido.

4.1. Marco de la propuesta

El problema anteriormente enunciado se enmarca en un contexto ingenieril centrado en el diseño de sistemas de simulación, en especial, aquellos empleados para la validación de soluciones en etapas tempranas del diseño de sistemas de fabricación. En este contexto, y adoptando un enfoque MBSE, se propone el uso de un modelo central que soporte toda la información necesaria para el diseño de un determinado producto, un enfoque que ofrece múltiples ventajas, como: a) provee las bases para mejorar el entendimiento y comunicación en contextos de modelado multiparadigma y multidominio; b) facilita la gestión de la consistencia en el conjunto heterogéneo de modelos manejados a lo largo del proceso de diseño de cualquier producto; y c) permite

capturar la información más importante del sistema en un único modelo, a partir del que desarrollar otros modelos de análisis.

En este contexto, SysML ha sido un lenguaje particularmente relevante, no sólo como lenguaje para describir un sistema con un alto nivel de abstracción y desde diferentes puntos de vista, sino también como lenguaje para conectar modelos heterogéneos, un mecanismo de integración que frecuentemente se apoya en la construcción de un modelo SysML pivote, como se propone en [112]. Por ello, la base de esta propuesta es la definición y uso de un modelo central SysML, que en esta investigación es construido por el usuario, si bien en un contexto más amplio se podría considerar la posibilidad de construirlo a partir de otros modelos CAx empleados para definir el diseño conceptual del producto/sistema de fabricación y, a medida que se detalle dicha solución, enriquecer y ampliar el modelo central, asegurando la integración de toda la información disponible. Con el fin de facilitar la interpretación del marco, la Figura 4.1 muestra gráficamente y de forma esquemática los principales elementos considerados, tanto los principales entornos y modelos considerados, como los flujos de información y transformaciones entre modelos.

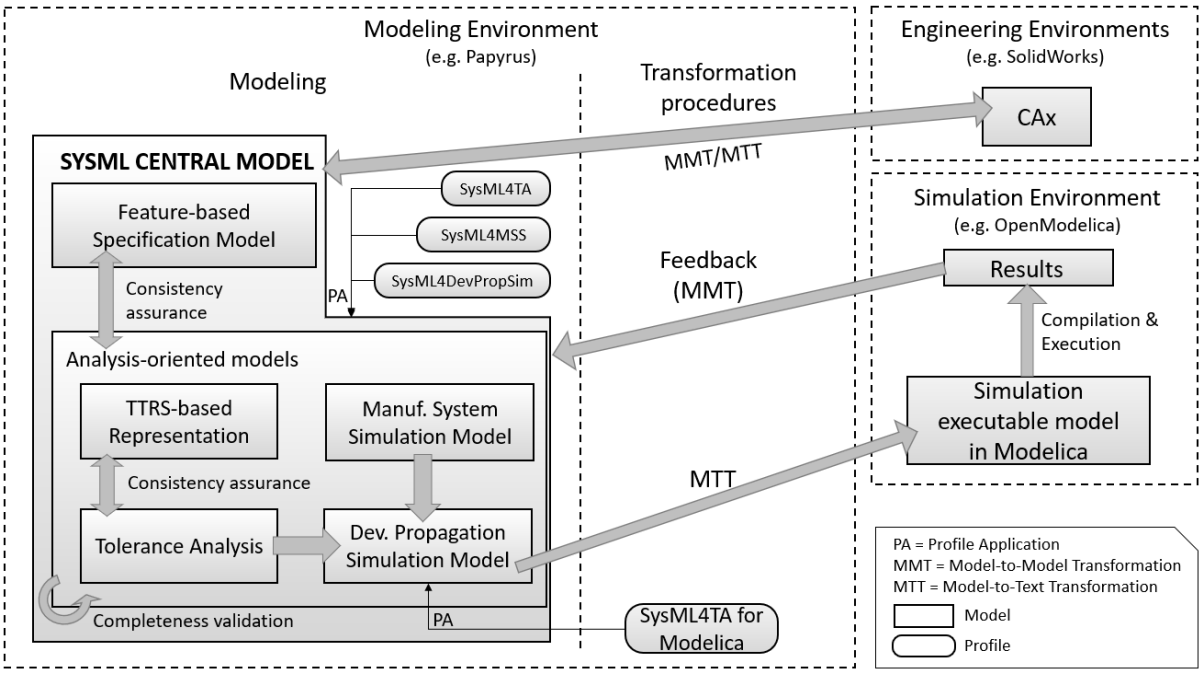


Figura 4.1. Marco de la propuesta.

Tal y como se representa en la Figura 4.1, el modelo central SysML incluye tanto modelos de especificación del sistema referente (sistema de fabricación, productos considerados, etc.) como otros modelos SysML, por ejemplo, del sistema referente orientados al análisis, o de sistemas de simulación. Entre todos estos modelos o vistas del modelo deben existir mecanismos para asegurar la consistencia de la información y la completitud de los modelos, que en el caso de esta propuesta estará soportado por la aplicación de perfiles SysML específicos (representados con forma ovalada en la figura). Una vez definido el modelo del sistema de simulación en SysML y asegurada su consistencia, se plantea la ejecución de una transformación automática del modelo

SysML a modelo Modelica (Model-to-Text Transformation – MTT). Para ello, se desarrolla un perfil adicional al modelo SysML del sistema de simulación que contiene los conceptos propios de Modelica, y cuya aplicación sobre el modelo SysML facilita dichas transformaciones. Una vez creado el modelo de simulación ejecutable de Modelica, éste se compila y ejecuta para obtener los resultados correspondientes. Cabe comentar que, en ocasiones, es interesante almacenar en el propio modelo central SysML dichos resultados, de manera que se plantea también la transformación de los resultados obtenidos para incorporarlos al modelo central.

Así pues, en el marco planteado el uso de perfiles SysML cobra un papel especialmente relevante, explorando su aplicación con dos objetivos principales:

- *Garantizar la consistencia*. La creación de perfiles como lenguajes de modelado específicos de dominio permite disponer de una serie de conceptos que capturan la semántica de dominio específica a través de reglas debidamente implementadas, como se tratará en el Capítulo 5. La aplicación de estos perfiles permite realizar las comprobaciones que confirman la consistencia del modelo o permiten detectar errores (inconsistencias) en su construcción.
- *Facilitar las transformaciones MMT*. La creación de perfiles también permite implementar en entidades SysML conceptos propios de otros lenguajes y metamodelos, estableciendo unas relaciones que permiten facilitar la automatización de las transformaciones MMT y MTT, como se verá en el Capítulo 7.

4.2. Requisitos de la metodología

Para el establecimiento de los requisitos del sistema de simulación es fundamental tener en cuenta a todas las partes interesadas, ya que resulta fundamental el trabajo colaborativo entre especialistas de diversas disciplinas, como en el diseño de cualquier sistema complejo. En el caso de las simulaciones planteadas en esta investigación, se considera a todos los participantes en el proceso de diseño y desarrollo de sistemas de fabricación físico, básicamente ingenieros/as de sistemas, de fabricación/ producción, de automatización, de calidad, etc.

Según [113], cualquier metodología de diseño de un artefacto técnico, como es el caso de un sistema de simulación, además de establecer el proceso también debe incidir en la forma en que el sistema es descrito, es decir el modelado. Por este motivo, resulta muy importante que la metodología propuesta favorezca la comunicación y colaboración interdisciplinar, un aspecto básico tanto a la hora de establecer los procedimientos y lenguajes de modelado como el flujo de actividades a desarrollar y su coordinación.

Por lo que respecta al modelado, es importante valorar la oportunidad de definir modelos descriptivos como apoyo a la definición de modelos de simulación ejecutables, explorando las necesidades de integración de estos y, en concreto, las posibilidades de transformación de modelos descriptivos a ejecutables y viceversa. Otros aspectos a valorar son el tipo de vistas que debe cubrir (requerimientos, estructuras,

comportamiento, etc..), la expresividad y grado de formalidad del lenguaje/s, y especialmente, los tipos de comportamiento (híbrido, multidominio, etc.) que se pueden describir. Por lo que respecta al enfoque de modelado, cabe indicar que los lenguajes de modelado orientados a objetos ofrecen una clara ventaja por lo que respecta al desarrollo, mantenimiento y reutilización de los modelos.

Teniendo en consideración todo lo comentado hasta el momento, los requisitos fundamentales adoptados para el desarrollo de la metodología se pueden concretar en su capacidad para:

- Facilitar el trabajo colaborativo y multidisciplinar de todos los interesados/as en el diseño, verificación y validación del sistema físico y del sistema de simulación correspondiente, garantizando la continuidad digital (transformaciones entre modelos, etc) desde la especificación del diseño del sistema referente hasta la verificación y validación del modelo de simulación ejecutable.

- Soportar el diseño de sistemas de simulación modulares, en base a su "componentización", utilizando el mecanismo de instanciación de componentes de librerías. Se trata de una propiedad que, como se ha comentado con anterioridad, posee la orientación a objetos.

- Abarcar todo el proceso de definición y desarrollo del sistema de simulación, desde el establecimiento de los requisitos hasta la definición de modelos de simulación ejecutables, incluyendo los procesos de soporte del mismo (trazabilidad, etc.).

- Ser implementable en entornos/plataformas para el modelado de sistemas, aprovechando las facilidades que éstas ofrecen para la verificación e integración de modelos.

- Modelar diferentes puntos de vista (dominios y perspectivas) y diferentes dinámicas de sistema (de eventos discretos, de dinámica continua e híbridos).

A partir de los requerimientos establecidos y de las conclusiones que emergen de las experiencias previas (Capítulo 3), el siguiente apartado presenta los aspectos clave de la metodología propuesta.

4.3. Metodología propuesta

Definido el problema y los requerimientos, es momento de concretar los elementos que conforman la metodología desarrollada, incluyendo la elección de los lenguajes y herramientas, así como la descripción del procedimiento propuesto. A modo de presentación inicial, la Figura 4.2 representa de forma gráfica estos elementos característicos de la metodología propuesta.

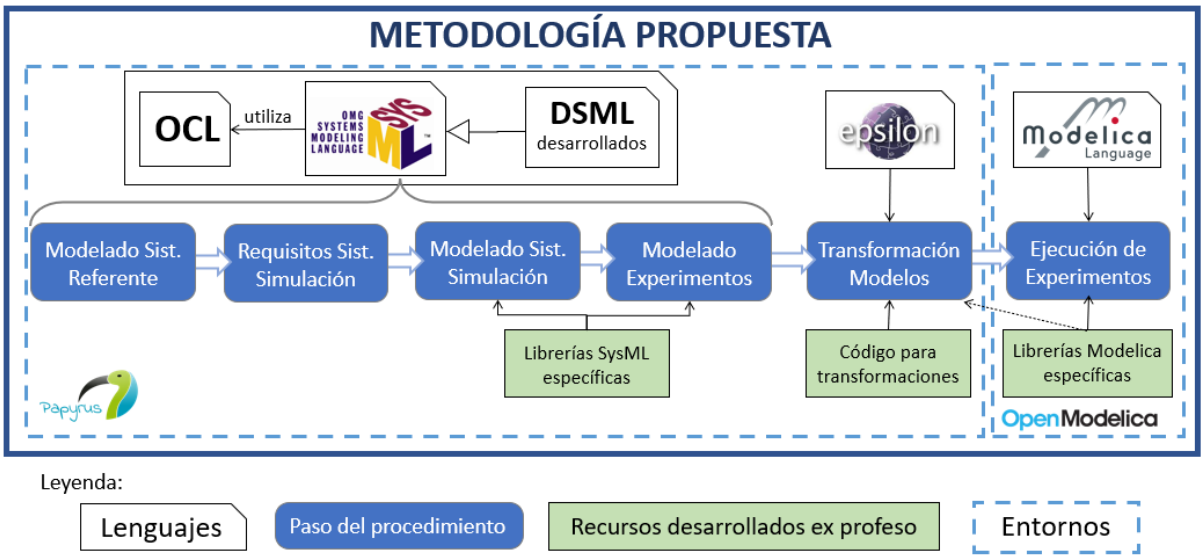


Figura 4.2. Elementos de la metodología propuesta.

Uno de los puntos clave en la definición de la metodología es la selección de unos lenguajes adecuados en función de las necesidades consideradas. En el caso de esta propuesta, la literatura analizada y las conclusiones extraídas de las experiencias de modelado previas (Capítulo 3) han corroborado las capacidades de los lenguajes empleados: SysML y Modelica. Por un lado, Modelica es un lenguaje consolidado en el modelado de simulaciones ejecutables y multidominio y capaz de soportar los análisis propuestos, como se ha podido comprobar en capítulos anteriores. Por su parte, la literatura analizada no solo identifica SysML como un lenguaje creado específicamente para el modelado de sistemas (incluyendo reglas OCL que limitan su sintaxis), sino que también se presenta como un interesante medio para crear lenguajes específicos de dominio (DSML) a través del mecanismo de creación de perfiles. Así pues, además de los mencionados lenguajes (Modelica y SyML, incluyendo el uso de reglas OCL), la propuesta metodológica también incluye DSML desarrollados ex profeso para soportar la semántica específica de los dominios de interés, incluyendo estereotipos que incorporan restricciones definidas en OCL para capturar dicha semántica mediante restricciones de naturaleza sintáctica. Por otra parte, también se propone el uso de Epsilon como lenguaje para la implementación de los códigos que permitan automatizar las transformaciones entre modelos. Cabe indicar que estos lenguajes y modelos creados ex profeso, junto con la creación de librerías, son el resultado de un conjunto de tareas de modelado a nivel de desarrollador y que, por tanto, quedan fuera del procedimiento definido en la metodología, que se establece a nivel de usuario. Los resultados de estas tareas de desarrollo (DMSL, librerías y transformaciones) serán presentados en detalle en los Capítulos 5, 6 y 7.

En lo referente a los entornos de modelado, en la experiencia previa descrita en el Capítulo 3 se ha demostrado la validez del software libre OpenModelica, por lo que se confirma su uso para el modelado, compilación y ejecución de los experimentos. En cuanto al modelado con SysML, cabe recordar que inicialmente se empleó como lenguaje gráfico con el que representar aspectos conceptuales a través de diagramas, sin necesidad de emplear un entorno de modelado específico. Sin embargo, en el estado

actual de la propuesta no solo se utilizan los diagramas como medio de representación, sino que se propone desarrollar modelos formales en los que almacenar información y verificar la consistencia de los mismos mediante la definición de reglas OCL. Por tanto, aunque la metodología propuesta pretende ser de aplicación independientemente del software adoptado, ha sido necesario seleccionar un entorno de modelado SysML para el desarrollo de la propuesta y de los demostradores que confirmen su viabilidad. En este caso, se ha optado por el software libre Papyrus. A pesar de que existen numerosas herramientas comerciales, como Cameo [114], MagicDraw [115], Microsoft Visio [116] o Visual Paradigm for UML [117], entre muchos otras, Papyrus es una de las opciones más extendidas en el mundo académico, tanto con fines de enseñanza como de investigación. Se trata de una herramienta de código abierto para edición y transformación de modelos de diferentes lenguajes (incluido SysML), integrada en Eclipse Modeling Framework. Además, permite definir y aplicar perfiles de una manera muy rica y eficiente, y ofrece una vista de validación específica que permite evaluar expresiones OCL para facilitar la gestión de la consistencia de modelos.

Finalmente, queda abordar la definición del procedimiento propuesto, una cuestión que por su envergadura será tratada de forma independiente en el siguiente apartado.

4.3.1. Procedimiento

En base a los requisitos comentados anteriormente se ha desarrollado un procedimiento formado por seis pasos, que van desde la especificación del sistema referente mediante modelos SysML hasta la obtención de modelos de simulación ejecutables que permitan verificar y validar las soluciones adoptadas. Aunque el procedimiento a grandes rasgos es general y, por lo tanto, aplicable al diseño de cualquier sistema de simulación, en la descripción detallada de cada paso se incluyen aspectos específicos de esta propuesta, orientada a simular sistemas de fabricación y, en particular, la propagación de desviaciones geométricas en sistemas multietapa. Además, el procedimiento propuesto no es estrictamente secuencial, pues contempla la posibilidad de solape de los pasos y de flujos de realimentación motivados por la modificación, mejora y/o ampliación de los modelos. A modo de síntesis, la Figura 4.3 presenta gráficamente estos seis pasos y la dinámica del proceso, indicando también el tipo de artefactos (modelos, diagramas, etc.) generados en cada uno de los pasos, si bien estos aspectos serán tratados en detalle en los siguientes apartados.

A. Modelado del sistema referente

En este primer paso se realiza el estudio de la estructura funcional y lógica del sistema a simular (sistema referente), en nuestro caso un sistema PPR que integra información de productos, procesos y recursos de fabricación. Este sistema debe ser analizado para conocer los elementos estructurales, funcionales y su comportamiento, así como las relaciones establecidas entre ellos. El estudio del sistema referente se consolida con el modelado del mismo en SysML, incluyendo vistas estructurales y comportamentales como diagramas de casos de uso (uc), diagramas de máquinas de estado (stm) o diagramas de definición de bloques (bdd), entre otros. En muchos casos, es necesario ampliar y/o modificar parte del modelo para soportar la información adecuada y

necesaria para el desarrollo del sistema de simulación desarrollado en etapas posteriores.

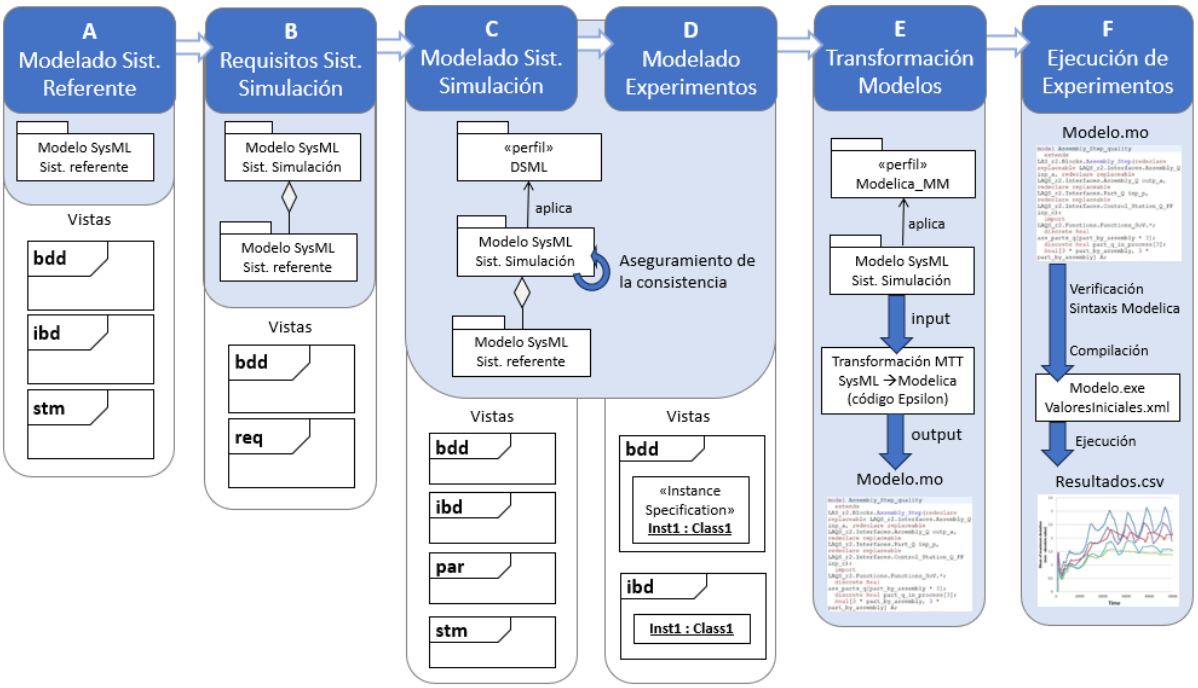


Figura 4.3. Pasos del procedimiento propuesto

Algunas construcciones de estos modelos (fundamentalmente bloques) aparecerán como referencias (relaciones de agregación) en el modelo del sistema de simulación, permitiendo así el acceso a información (datos) almacenada en el modelo del sistema referente y que sea relevante en el análisis desarrollado y el cumplimiento a los requerimientos establecidos para el sistema de simulación.

B. Análisis previo del sistema de simulación

El segundo paso se centra en el análisis previo del sistema de simulación, incluyendo el establecimiento de los requisitos. En función del tipo de sistema que se quiere simular y de las metas y medidas de efectividad establecidas, el diseñador/a puede definir varios conceptos alternativos que sustenten el diseño del sistema de simulación (p. e. los aspectos a integrar -logístico/producción, calidad del producto, salud de los equipos, etc.-, el paradigma de simulación, etc.) y posteriormente los requisitos para cada uno de ellos. En este paso, en el modelo del sistema de simulación son definidos principalmente diagramas de requisitos (req), aunque éstos se apoyan en diagramas de definición de bloques (bdd) muy generales en los que identificar la estructura básica del sistema, así como otros diagramas (casos de uso, actividades, ...). En el contexto de esta investigación, los requerimientos estarán vinculados fundamentalmente con el modelado de las desviaciones geométricas del sistema de fabricación y sus efectos en la calidad del producto, incluyendo el análisis de la propagación de los errores a lo largo de las etapas del proceso de fabricación considerado.

C. Modelado del sistema de simulación

A partir de los requisitos establecidos en el paso anterior, se selecciona una solución viable para la estructura de objetos y para el comportamiento del sistema de simulación. Las descripciones de estructura y comportamiento se descomponen al nivel de abstracción requerido y adecuado a las metas y métricas establecidas. En el contexto de esta investigación, los modelos de simulación desarrollados adoptan una estructura lo más cercana posible a la propia estructura del sistema referente, para facilitar tanto las comprobaciones de consistencia como las transformaciones a modelos ejecutables. Por su parte, el modelado del comportamiento también intenta asemejarse todo lo posible al del sistema referente, que en el caso de los sistemas de simulación se basa en eventos discretos, y que adicionalmente se puede combinar con otras dinámicas continuas, dando lugar a simulaciones híbridas.

En este paso, el modelo del sistema de simulación se amplía creando Diagramas de definición de bloques (bdd) más detallados, Diagramas de bloques internos (ibd), y Diagramas paramétricos (par), mientras que el comportamiento viene descrito fundamentalmente por Diagramas de máquinas de estados (stm). Además, es frecuente que el modelo del sistema de simulación tenga una relación de agregación con el modelo del sistema referente para tener acceso y utilizar cierta información (datos, características) que asegure la consistencia.

Cabe mencionar que, en el caso particular de esta propuesta, estos modelos deben estar construidos por elementos de las librerías desarrolladas (descritas en el Capítulo 6) y que, en caso de que el diseñador cree nuevas construcciones, éstas se deben estereotipar con conceptos propios de los perfiles (DSML) propuestos para garantizar la consistencia de los modelos definidos.

D. Modelado de los experimentos

En este paso, el modelo del sistema de simulación se concreta definiendo valores particulares para los parámetros de la simulación, pudiendo desarrollar diferentes versiones del modelo que den soporte a experimentos alternativos. Esta asignación de valores se puede realizar definiendo instancias (Instance Specifications) en el propio modelo de simulación, o bien vinculando determinadas propiedades valor con instancias del modelo del sistema referente. El objetivo es definir los parámetros necesarios para que el modelo pueda ser resuelto al transformarlo a un modelo ejecutable.

E. Transformación de SysML a Modelica

Para obtener un modelo de simulación ejecutable, todas las descripciones del sistema previas deben implementarse utilizando un lenguaje con esta capacidad, como es Modelica. Si se han completado las etapas anteriores y el modelo SysML tiene el detalle suficiente, se propone automatizar esta transformación del modelo SysML a modelo textual Modelica (fichero de extensión .mo) mediante la ejecución de una aplicación basada en la especificación del estándar Query/View/Transformation Language (QVT) [118]. Conviene indicar que, para el correcto uso de esta aplicación de transformación

desarrollada, es necesario aplicar previamente un perfil específico (MMT2Modelica) que enriquece el modelo SysML, identificando conceptos propios de Modelica e incluso estableciendo determinadas asignaciones o relaciones con construcciones disponibles en la librería Modelica. El detalle de estas transformaciones será tratado en el Capítulo 7.

F. Ejecución de las simulaciones Modelica

Con el modelo Modelica ya definido, el último paso es compilar y ejecutar los experimentos. En esta etapa, los entornos como OpenModelica ejecutan en primer lugar una verificación del modelo para asegurar que está bien construido (en base a la sintaxis Modelica) y que cuenta con los datos suficientes para resolver el sistema de ecuaciones planteado. Si este proceso de verificación resulta exitoso, se compila el modelo (obteniendo un fichero .exe y el almacenamiento de valores iniciales en un fichero .xml) y ejecuta la simulación, mostrando los resultados del análisis (almacenados en un fichero .csv). De forma complementaria, en ocasiones resulta interesante incorporar parte de estos resultados al modelo del sistema referente, aunque esta cuestión queda fuera del alcance de la propuesta.

5. Especificación de los lenguajes de dominio

El presente capítulo recoge la definición de los tres lenguajes de dominio propuestos, orientados a asegurar la consistencia de los modelos en función de la semántica específica de los dominios de interés. Para ello, se dedica una primera sección a presentar y justificar el desarrollo de estos lenguajes, descritos en detalle a lo largo de las tres secciones restantes.

5.1. Presentación de los lenguajes

Durante el diseño y desarrollo de sistemas, y especialmente en tareas orientadas a verificar y validar las soluciones adoptadas, es muy común desarrollar modelos matemáticos que permitan emular y analizar ciertas características y/o comportamientos del sistema referente. En el caso del análisis de sistemas simples, en ocasiones es posible desarrollar modelos matemáticos que permitan obtener una solución analítica. Sin embargo, a medida que los sistemas son más complejos, resulta difícil obtener dicha solución analítica y es necesario recurrir a la simulación, desarrollando sistemas software que intentan reproducir con la mayor fiabilidad posible el comportamiento real del sistema referente, especialmente en todo aquello que es de interés para el análisis propuesto. Para ello, y aunque existen herramientas específicas para la simulación de determinados dominios (incluido el análisis de sistemas de fabricación), una buena práctica es adoptar herramientas y lenguajes genéricos que permitan desarrollar sistemas de simulación multidominio, integrando en un único modelo diversos aspectos del sistema a analizar.

Sin embargo, en relación a la verificación y validación de los sistemas de simulación, conviene tener presente que los lenguajes genéricos (Modelica, por ejemplo) se limitan a verificar sintácticamente los modelos, comprobando que el conjunto de variables y relaciones matemáticas es suficiente para alcanzar una solución. Estos lenguajes carecen de mecanismos que permitan extender el lenguaje e incorporar la semántica específica de los dominios analizados, que permitirían comprobar la consistencia de los modelos en base a dicha semántica específica. Ante esta limitación, se propone diseñar los sistemas de simulación mediante modelos descriptivos empleando un lenguaje específico (perfil SysML) que asegure la consistencia. Posteriormente, estos modelos descriptivos serán enriquecidos con la aplicación de otro perfil SysML para facilitar su transformación a modelos ejecutables de forma automática, aunque esta cuestión será tratada en el Capítulo 7.

Con esta orientación, este capítulo presenta tres lenguajes de modelado específicos de dominio (DSML) implementados como perfiles SysML. La especificación de cada lenguaje incluye la definición de un metamodelo, en el que se establece la sintaxis abstracta (conceptos, propiedades y relaciones) y aspectos de la semántica estática (reglas que capturan el significado de los conceptos manejados). En la posterior implementación como perfil SysML cada concepto del lenguaje se define como estereotipo, incluyendo las reglas OCL que dan soporte a la semántica estática. En relación a la sintaxis concreta, estos perfiles heredan las características del lenguaje base (SysML), y no se aporta ningún cambio en este sentido en la especificación de los nuevos lenguajes. Finalmente, cabe indicar que la semántica dinámica será tratada en el Capítulo 7, dedicado a las transformaciones de modelos y, por tanto, al mapeo entre conceptos de diferentes lenguajes, en concreto entre los perfiles SysML y el lenguaje Modelica.

Como ya se ha indicado, esta investigación se centra en el modelado de sistemas de simulación para el análisis de sistemas de fabricación multietapa, poniendo el foco en el análisis de la calidad geométrica (emulación de desviaciones y cumplimiento de especificaciones geométricas) y la productividad (simulación de eventos discretos y cálculo de índices de productividad, bloqueos, etc.). Con este objetivo, inicialmente se exploraron las capacidades de Modelica y se adoptó la técnica Stream of Variation para desarrollar una propuesta inicial descrita en el Capítulo 3. Sin embargo, la experiencia con este tipo de formulación matemática puso de manifiesto dos importantes limitaciones de este enfoque: a) las desviaciones en una etapa se deben expresar como una relación estrictamente lineal de las desviaciones resultantes de la etapa anterior y las desviaciones introducidas por el utillaje; y b) se pierde la relación entre las desviaciones consideradas y las especificaciones geométricas establecidas para las piezas y los utillajes. Por este motivo, se decidió explorar otras soluciones alternativas para el modelado matemático de superficies y sus relaciones. En concreto, se consideró especialmente interesante el modelo Technologically and Topologically Related Surfaces (TTRS) [60] cuyos conceptos dan soporte al modelado geométrico de artefactos, lo cual no solo es aplicable al producto, sino también a otros artefactos más complejos, como es el caso del ensamble de proceso, necesario para el tipo de análisis que se pretenden desarrollar.

Además de buscar una alternativa para el modelado geométrico en el que se soporta el análisis de la calidad, el modelado del propio sistema de simulación también ha sido objeto de estudio, analizando el tipo de elementos, datos y relaciones necesarias en los sistemas de simulación que se pretenden desarrollar. Así, se han abordado aspectos característicos de los sistemas de simulación que permiten analizar sistemas con comportamiento discreto (como es el caso de los sistemas de fabricación por lotes), pero también aspectos estructurales y comportamentales de los sistemas de fabricación (el sistema referente considerado) que influyen en el modelado de los sistemas de simulación.

Atendiendo a estas dos principales líneas de investigación (modelado geométrico de artefactos y modelado de sistemas de simulación), se han desarrollado tres lenguajes

orientados al aseguramiento de la consistencia de los modelos en base a la semántica de estos dominios. Concretamente, los lenguajes desarrollados/propuestos son:

- *SysML for Manufacturing System Simulation (SysML4MSSim)*. Lenguaje para el modelado de sistemas de simulación para el análisis de la productividad de los sistemas de fabricación discreta (por lotes), caracterizados por un comportamiento discreto que puede ser descrito mediante el formalismo *Discrete EVent System specification* (DEVS).

- *SysML for Tolerance Analysis (SysML4TA)*. Lenguaje para el modelado estructural de artefactos físicos, que incluye la representación de su geometría nominal y desviada en base a los conceptos de TTRS y a construcciones matemáticas como el Small Displacement Torsor (SDT) [119]. Aunque el lenguaje se presenta de forma genérica para abordar el modeado de cualquier artefacto, estos conceptos dan soporte al modelado del ensamble de proceso, que permite describir la configuración de los recursos de fabricación a nivel de subfase (análisis unietapa).

- *SysML for Deviation Propagation Simulation (SysML4DPSim)*. Lenguaje que especializa conceptos del SysML4MSSim para modelar sistemas de simulación que, además del análisis de la productividad, incorporen el análisis de las desviaciones geométricas en sistemas de fabricación multietapa. Para ello, se incorporan conceptos definidos en SysML4TA que dan soporte al modelado geométrico de productos, utillajes y ensambles de proceso, y se utilizan para emular la propagación de las desviaciones geométricas a lo largo de las etapas consideradas en base a la definición de un espacio de estados (análisis multietapa).

Antes de abordar cada lenguaje, es necesario enfatizar que el objetivo final de estos lenguajes es dar soporte al modelado del sistema de simulación (el sistema de interés), una aplicación software que inicialmente se limita al análisis de la productividad de sistemas de fabricación y que posteriormente se amplía para incorporar el análisis de la calidad geométrica. Para ello también es fundamental tener un profundo conocimiento sobre el sistema analizado (sistema referente), que en el caso de esta investigación es un sistema PPR, es decir, un supersistema formado por el producto, el proceso y el sistema de fabricación. Todas las características del sistema referente influyen en la productividad y la calidad tendrán su respectiva representación en el modelo de simulación. Aunque las características del sistema referente pueden estar opcionalmente descritas en un modelo de especificación (modelo del sistema referente), esta investigación se enfoca en determinar cómo se representan determinadas características del sistema referente en el sistema de simulación, generalmente definiendo estructuras de datos y haciendo uso de flujos de datos, algoritmos y relaciones matemáticas. La Figura 5.1 ejemplifica gráficamente esta diferenciación entre sistema referente y sistema de simulación, y destaca la aplicación de los perfiles SysML propuestos. Aunque es obvio, conviene aclarar que la Figura 5.1 no es un diagrama de clases UML, aunque se hayan utilizado ciertas notaciones UML para las relaciones (dependencias, referencias, especializaciones, etc.) mostradas en la misma, con la idea de aumentar el significado de lo representado para todos aquellos lectores que conocen la notación UML.

Por lo que respecta a la estructura de contenidos del capítulo, cabe indicar que éstos se muestran en tres secciones, dedicadas a la especificación de sendos lenguajes. Cada una de estas secciones se divide a su vez en tres apartados: a) un breve estado del arte que acerca al lector a algunos conceptos básicos y trabajos previos considerados en el desarrollo de los lenguajes; b) una descripción conceptual del metamodelo, que describe los conceptos y relaciones a modelar; y c) la implementación del perfil SysML, en la que se describe brevemente los estereotipos propuestos y algunas de las principales reglas que definen la semántica concreta del lenguaje.

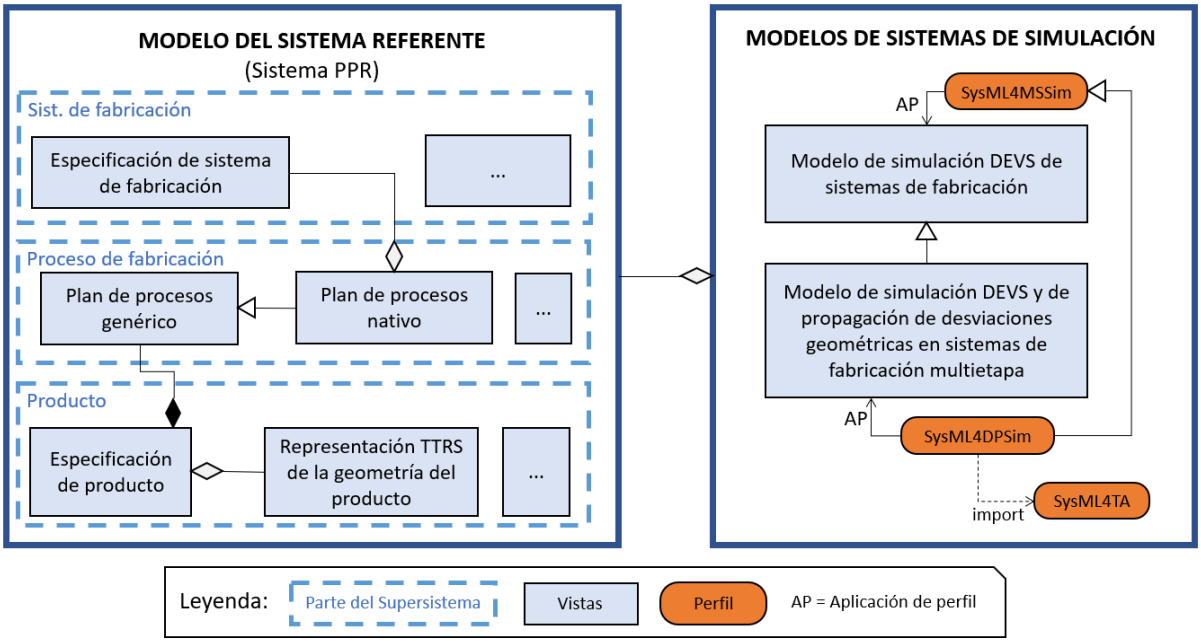


Figura 5.1. Modelos del sistema referente y del sistema de simulación

Antes de abordar cada una de estas secciones, conviene aclarar que la definición completa de los estereotipos de cada perfil, incluyendo la descripción de las reglas que soportan la semántica concreta de cada estereotipo y su implementación en OCL, se recoge en el Anexo A.

Cada uno de estos perfiles se estructura en diferentes paquetes, que permiten organizar los estereotipos en función del tipo de concepto que representan o su ámbito de uso. Esta estructura de paquetes se representa en el diagrama de la Figura 5.2, y la designación de cada paquete se emplea también para estructurar los apartados de este capítulo y del citado Anexo A.

Por último, en relación al diagrama de paquetes de la Figura 5.2, conviene aclarar que el paquete «profile» SysML4DPSim cuenta con dos relaciones de tipo «import» con los paquetes que contienen los otros dos perfiles. Esto es debido a que, para la definición de este perfil, ha sido necesario importar los otros paquetes, de manera que se pudiera acceder y hacer uso de sus construcciones para ser especializadas o integradas en el nuevo perfil.

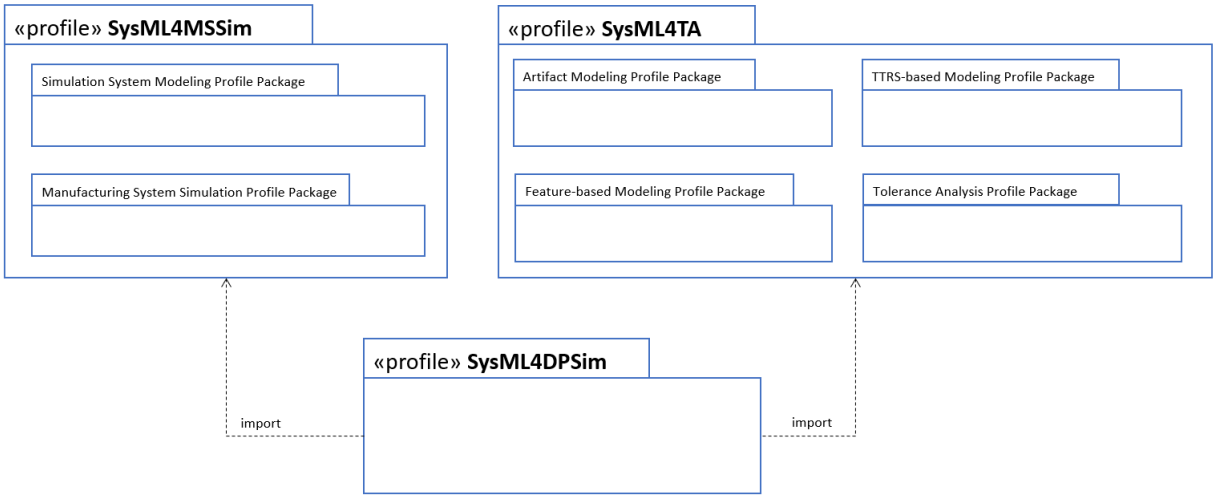


Figura 5.2. Diagrama de paquetes de los tres perfiles y sus relaciones.

5.2. Lenguaje para la simulación de sistemas de fabricación y el análisis de la productividad

La presente sección se centra en el desarrollo de un lenguaje que soporte el modelado en SysML de sistemas de simulación, y en concreto a aquellos construidos con el propósito de analizar la productividad de sistemas de producción por lotes. Estos sistemas se caracterizan por un comportamiento discreto que puede ser descrito mediante el formalismo DEVS, y que integra aspectos tanto del sistema de fabricación como del producto y el proceso.

5.2.1. Introducción

Como se ha comentado anteriormente, los modelos de simulación son modelos software ejecutables desarrollados para el análisis de cierto sistema referente que normalmente suele ser bastante complejo. Estos modelos permiten analizar el rendimiento del sistema simulado como respuesta a diferentes condiciones y supuestos considerados, derivados tanto del estado del sistema referente como de su entorno. Dichas condiciones se definen generalmente asignando valores a los parámetros, unos atributos (propiedades valor, generalmente) característicos de cada modelo de simulación. El modelo de simulación permite ejecutar diferentes experimentos diferenciados por los valores de esos parámetros. Por lo tanto, el valor de los parámetros puede variar entre experimentos, pero se mantienen constantes en un mismo experimento.

En cada experimento, el sistema de simulación debe realizar los cálculos y transformaciones necesarias para obtener, a partir de unas determinadas medidas, indicadores que estén alineados y cuantifiquen el cumplimiento de los requerimientos planteados para el diseño del sistema de simulación. En el caso particular de esta propuesta, los sistemas de simulación desarrollados están orientados a analizar la productividad de los sistemas de fabricación, y para ello se trabajará con medidas básicas e indicadores de rendimiento como la cantidad de unidades producidas por

hora (*throughput*), la duración media de la fabricación de productos o el cumplimiento de los plazos de entrega, entre otros.

Para ello, el sistema de simulación debe contar con los elementos estructurales y comportamentales necesarios para emular el sistema PPR en su conjunto, incluyendo el conjunto de recursos considerados y sus conexiones, así como el flujo de materiales o la ejecución de procesos de fabricación. A continuación, se presentan diferentes subapartados dedicados al modelado de la estructura y el comportamiento de los sistemas de simulación, y a su relación con las características estructurales y comportamentales del sistema referente.

A. Estructura de los modelos de simulación

La estructura de un modelo define, de forma general, el tipo de componentes que lo forman y las relaciones entre ellos. En el caso del modelado de sistemas de simulación (sistemas soft), estos componentes (objetos software) intercambian datos y realizan transformaciones y cálculos. De forma general, los modelos de simulación de sistemas discretos están construidos a partir de un conjunto de objetos que comparten datos a través de conexiones por las que transcurren una serie de unidades de flujo. A continuación, se describen de forma más detallada estos dos tipos de elementos.

Por una parte, una unidad de flujo es un elemento que fluye por el sistema y que, generalmente, tiene una vida finita (se crean y destruyen en determinados instantes). Algunas referencias bibliográficas, como [120], emplean el término "entidad" para este tipo de elemento, pero en este documento se adopta "unidad de flujo" por ser más descriptivo y próximo a su definición. En la simulación de sistemas PPR, las principales unidades de flujo simuladas incluirán datos sobre los materiales, lotes de producto, órdenes de fabricación, etc.

Por otro lado, las entidades permanentes son componentes del sistema de simulación que desempeñan una determinada función y que, a diferencia de las unidades de flujo, tienen existencia en todo el tiempo simulado (su vida no es finita). En algunas referencias bibliográficas, como en [120], se emplea el término "máquina", mientras que otros trabajos se refieren a estos elementos simplemente como "entidades" [43]. Con el fin de evitar confusiones en la terminología, se ha optado por el término "entidad permanente" por poner énfasis en su principal diferencia frente a las unidades de flujo. Otro aspecto característico de las entidades permanentes es que, en muchas ocasiones, su definición incluye un determinado comportamiento, expresado en forma de algoritmos y relaciones matemáticas, así como elementos de conexión (puertos) que dan soporte a los flujos de datos considerados. En el ámbito de esta investigación, diferentes funcionalidades son identificadas, tanto para representar el sistema simulado como para dar soporte a otras funciones propias del sistema de simulación. Entre estas funciones, destacan:

- *Generar unidades de flujo*. Los sistemas de simulación basados en el análisis de flujos de datos requieren de componentes específicos encargados de generar la unidad de flujo en función de una programación o comportamiento definidos. Por tanto, estos componentes representan el inicio del flujo y cuentan con

puertos mediante los cuales transmitir los datos de las unidades de flujo generadas. En el ámbito de esta investigación se generan, por ejemplo, datos sobre órdenes de fabricación e insumos materiales.

- *Transformar las unidades de flujo*. Los sistemas de simulación, y en concreto aquellas partes que emulan el comportamiento del sistema referente, están formadas por componentes a través de los cuales fluyen las unidades de flujo, sufriendo algún tipo de transformación. Por tanto, estos componentes tienen al menos dos puertos, para soportar la entrada y salida de las unidades de flujo. En la simulación de sistemas PPR, esta función está principalmente soportada por las entidades que representan estaciones de trabajo transformadoras, como por ejemplo, estaciones de ensamble o de mecanizado.

- *Soportar procesos de espera*. Además de los procesos transformadores, en la simulación de comportamientos discretos es muy común incluir componentes (buffers) que den soporte a tareas de cola o espera, generalmente debido a que una determinada entidad transformadora no está disponible para admitir la unidad de flujo entrante. Por tanto, estos componentes también tienen al menos dos puertos, para soportar la entrada y salida de las unidades de flujo. En la simulación de sistemas PPR, este tipo de elementos representan almacenes reales del sistema de fabricación, que permiten regular el flujo logístico (de materiales) y servir los lotes de producto con una determinada estrategia (FIFO, LIFO, etc.).

- *Destruir unidades de flujo*. Todo flujo debe tener un punto de fin en el que las unidades de flujo se destruyen. Los componentes que dan soporte a esta función generalmente se llaman sumideros, y deben contar con al menos un puerto para soportar la llegada de unidades de flujo.

- *Monitorizar datos y tomar decisiones*. Además de entidades que intercambian y son atravesadas por unidades de flujo, los sistemas de simulación también incluyen entidades que, sin participar de este flujo, se encargan de monitorizar determinados datos de la simulación, soportando la captura y tratamiento de información clave. En ocasiones, estos componentes también soportan la toma de decisiones, que pueden implicar cambios en las características de otros componentes del sistema de simulación. Un claro ejemplo en la simulación de sistemas PPR es la simulación de los sistemas de control. Además, se pueden considerar también componentes definidos exclusivamente para calcular los indicadores de interés a partir de las medidas y datos manejados en diversas partes del sistema de simulación.

Más allá de la diferenciación entre unidades de flujo y entidades permanentes, es necesario tener en cuenta que las construcciones de ambos tipos deben ser definidas y caracterizadas por una serie de atributos, que son piezas de información (estructuras de datos) que caracterizan un determinado elemento y permiten diferenciarlo de otros semejantes (del mismo tipo). En muchos casos, estos atributos son definidos como parámetros, cuyo valor puede variar entre experimentos. En las simulaciones que se pretende desarrollar, algunos atributos típicos de las unidades de flujo consideradas son, por ejemplo, el tamaño del lote, el tipo de producto, el estado de los productos, etc. En cambio, en las entidades que representan estaciones de trabajo y almacenes

serán atributos comunes el tipo de estación de trabajo o la capacidad de almacenamiento, entre muchos otros.

Complementariamente a lo mencionado anteriormente, es interesante hacer una breve reflexión sobre el término "recurso", una palabra ampliamente extendida y utilizada en diferentes ámbitos y que, debido a la gran variedad de significados y connotaciones implícitas en función del contexto y la intención con la que se utiliza, en ocasiones puede llevar a confusión o duda en su interpretación. En esta investigación, se considera que un recurso no es un tipo de elemento, sino un rol con el que un elemento participa en un determinado sistema. En este sentido, se llamará recurso a cualquier tipo de elemento que es utilizado para poder realizar un determinado proceso o acción, sea este un comportamiento propio o no. Por ejemplo, en una empresa dedicada a la fabricación de mordazas de sujeción, el elemento "mordaza" representa el producto. Sin embargo, si otra empresa emplea una mordaza como utillaje de sujeción, por ejemplo, en una estación de fresado, esa misma realidad (la mordaza) pasa a considerarse como un recurso de fabricación. Por tanto, la diferenciación en principio no está en la definición de ese elemento, sino en la forma en que éste se emplea o participa.

A la hora de modelar un sistema, los roles considerados sí pueden influir en la forma en que un determinado elemento debe modelarse. Por ejemplo, si únicamente se desea modelar la mordaza para considerar su rol como producto, quizás interese definir su geometría, materiales, plan de proceso, etc. En cambio, si se modela el sistema de fabricación en que se utiliza la mordaza como recurso de amarre, posiblemente interese más centrar su definición en el tipo de geometrías que puede amarrar, el intervalo de apriete que es capaz de ejercer, etc. En el ámbito de los sistemas de simulación, es frecuente emplear el término "recurso" para referirse a aquellos objetos de los que únicamente interesa conocer su disponibilidad, porque son necesarios para un determinado proceso, pero que carecen de características adicionales. Sin embargo, a la hora de modelar y simular un sistema de fabricación, el término "recurso" puede requerir una definición del elemento más extensa en función de los intereses, incluyendo no sólo su disponibilidad o capacidad para soportar un proceso, sino también otras características, como su geometría, configuraciones, etc. Es por ello que, en adelante, se emplea el término "recurso" para referirse en un sentido más cercano al dominio de la fabricación, identificando aquellos elementos del sistema de simulación que representan recursos de fabricación con un comportamiento propio (por ejemplo, estaciones de trabajo).

A modo de síntesis de los elementos típicos en un sistema de simulación, se puede decir que las unidades de flujo son los elementos principales desde el punto de vista del análisis, puesto que contienen los datos que se transforman mientras fluyen por el sistema. Sin embargo, desde el punto de vista de la construcción y estructuración del sistema de simulación, los elementos básicos son las entidades permanentes que dan soporte a las diferentes funciones. Además, estas entidades permanentes deben conectarse entre sí para dar soporte al intercambio de datos, que representan interacciones y flujos de diferente naturaleza presentes en el sistema referente. En el caso de la simulación de sistemas PPR, este flujo de datos puede representar, por

ejemplo, la comunicación entre equipos de fabricación (flujo de señales eléctricas, datos, etc.) o la transferencia de piezas desde una estación de trabajo a otra (flujo logístico o de materiales). En este sentido, el modelado de sistemas de simulación en SysML cuenta con la ventaja de que este lenguaje define elementos específicos para modelar diversos tipos de conexiones, tanto en lo referido a la definición de puertos (Port, estereotipo FlowPort, etc.) como de los conectores (Connectors y BindingConnectors), incluyendo mecanismos para la definición del tipo de conexión a través bloques asociativos (AssociationClass estereotipada como Block), que permiten definir las características y comportamientos que se dan en la propia relación entre elementos.

Sin embargo, el modelado de sistemas de simulación con SysML cuenta con una importante limitación en la definición de modelos de instancias (objetos con valores concretos). En este sentido, conviene tener en cuenta que, en modelos de simulación complejos, los componentes se definen instanciando elementos tipo (clases Modelica, por ejemplo), habitualmente definidos en una librería de elementos reutilizables. Estas instancias se crean mediante constructores, operaciones específicas para la creación de objetos mediante las cuales se definen además valores concretos para sus parámetros. Sin embargo, el modelado con SysML no incluye estos mismos mecanismos de instanciación, pero tiene la capacidad de definir instancias, principalmente mediante entidades de tipo InstanceSpecification, en las que se define un objeto vinculado con un determinado tipo (bloque) y en el que se concretan los valores de sus propiedades mediante Slots. En esta investigación, se empleará este tipo de construcciones para definir experimentos concretos en base a los modelos de simulación (bloques) definidos.

Como se ha comentado anteriormente, el objetivo del sistema de simulación es recrear un sistema referente mediante construcciones soft. Sin embargo, en función del tipo de análisis propuesto, la estructura del sistema de simulación puede ser semejante a la estructura del sistema referente, o bien optar por otros enfoques que cubran las necesidades propias del análisis y aporten alguna ventaja, como la simplificación del modelo o la economía computacional. Acercando estos dos enfoques alternativos al caso de la simulación de sistemas PPR (producto-proceso-sistema de fabricación), la simulación puede estar centrada, por ejemplo, en el proceso de fabricación, de manera que el sistema de simulación se construya con una serie de componentes que representen procesos de fabricación de diferentes niveles (operación, fase, subfase, etc.). En este enfoque, el modelo de simulación se estructura según el plan de procesos de los productos a simular, y la activación de algunos procesos puede requerir de la disponibilidad de determinados recursos de fabricación.

Alternativamente, se puede optar por construir un modelo de simulación cuya estructura sea más próxima a la arquitectura física del sistema de fabricación real, de manera que los componentes del sistema de simulación representen diversos recursos de fabricación (estaciones de trabajo, máquinas, etc.). En este enfoque, las unidades de flujo pasan de unos componentes a otros a través de conexiones, representando las transferencias reales de las unidades materiales en la planta. Por su parte, cada componente que representa un recurso de fabricación tiene unos determinados

comportamientos que representan, entre otras cosas, los diversos procesos de fabricación que son capaces de ejecutar.

Desde el punto de vista de la simulación de eventos discretos y el análisis del flujo de materiales y la productividad, las dos opciones comentadas son válidas. De hecho, la mayoría de las librerías comerciales para la simulación de sistemas de fabricación están basadas en un marco de modelado orientado a procesos [121]. Sin embargo, considerando el objetivo último de analizar la propagación de desviaciones geométricas en los productos, para lo cual se requiere de una definición geométrica de los recursos (tratada en la próxima sección), parece más apropiado adoptar un enfoque más cercano a la estructura física del sistema de fabricación. Por este motivo, a partir de este punto, todas las consideraciones planteadas se presentan bajo el enfoque de estructurar el modelo de simulación en base a la arquitectura física del sistema de fabricación analizado.

B. Consideraciones estructurales del sistema referente

Vistas algunas generalidades de los modelos de simulación, es el momento de centrar el discurso en determinadas características estructurales de los sistemas de fabricación que influyen en la construcción de los sistemas de simulación. Conviene recordar que los sistemas de fabricación objeto de estudio en esta investigación son sistemas mecatrónicos complejos, de fabricación discreta, altamente automatizados y flexibles, es decir, que ejecutan una fabricación por piezas o lotes, con procesos automatizados que no consideran la intervención humana, y que están diseñados para soportar diversos procesos de fabricación con pequeños cambios en la configuración de sus recursos.

De forma general, en el ámbito de la fabricación un recurso de fabricación es el rol de cualquier activo que desempeña una función en la producción de bienes (productos). Activos que a su vez pueden formar parte de otros más complejos, obteniendo así una estructura jerárquica de múltiples niveles. De hecho, el propio sistema de fabricación en su conjunto es un recurso compuesto por otros recursos. A la hora de definir un lenguaje para el modelado de las simulaciones de estos recursos, será necesario considerar algunos de estos niveles de agregación claves, especialmente para definir el nivel atómico por debajo del cual no interese entrar en detalle desde la perspectiva de los intereses del análisis planteado.

En relación a la estructuración jerárquica de los recursos, la Figura 5.3 muestra una adaptación de la clasificación de componentes de la Industria 4.0 propuesta en [122], en la que además se proponen ejemplos del ámbito de la fabricación metal-mecánica. Esta figura también sirve para establecer el límite del concepto “recurso” según los criterios mencionados, siendo los recursos atómicos los situados en el nivel de Estación de Trabajo (4) por tener un comportamiento propio y por influir directamente en las características de calidad que se tratarán más adelante, conformando lo que se denomina una máquina configurada, uno de los elementos clave en la construcción del ensamble de proceso necesario para el modelado de las desviaciones geométricas.

Capa	Descripción	Ejemplos
9.- Red de producción	Incluye la planificación estratégica de ventas a largo plazo	Compañía, sistema de producción
8.- Fábrica	Conjunto de actividades e instalaciones necesarias para producir productos finales o un bien intermedio.	- Planta o sistema de fabricación
7.- Línea de producción	Sección de un sistema de producción que producen y procesan componentes del producto final utilizando diferentes tecnologías de fabricación.	- Línea de producción de una determinada familia de productos
6.- Segmento de línea de producción	Unidades de trabajo conectadas mediante buffers, controlando las perturbaciones del flujo de producto/componente/material.	- Sección de mecanizado: unidades de torneado, fresado y taladrado con almacenes intermedios.
5.- Unidad de trabajo	Ejecuta un proceso atónico (más pequeño, no divisible) en la producción del producto final.	- Unidad de torneado: Estación de torneado + mesa de brutos + mesa de piezas mecanizadas
4.- Estación de trabajo	Realiza un conjunto de funciones auxiliares y de valor añadido necesarias para el proceso atómico.	- Estación de torneado: torno + garras autocentrantes + portaplaquitas + plaquitas
3.- Grupo funcional	Realiza una función auxiliar o de valor agrañadido requerida en el proceso atómico.	- Torno - Fresadora
2.- Componente	Desempeña una función en un grupo funcional.	- Conjunto plato + plaquitas - Mordaza
1.- Elemento constructivo	No desarrolla ningún proceso, pero es necesaria para la funcionalidad al componente (2).	- Plaquita de corte. - Boca de una mordaza

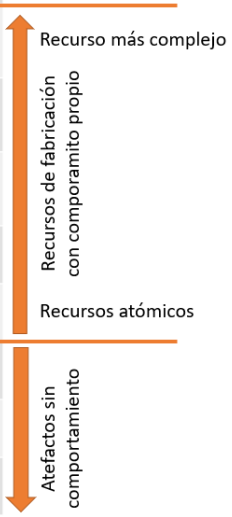


Figura 5.3. Clasificación jerárquica de componentes de la Industria 4.0 basada en [122]

Conviene aclarar que, en muchas propuestas del ámbito de la fabricación, el término *recurso* también se aplica a elementos que participan de forma pasiva, formando parte de otros recursos más complejos, como es el caso de los utillajes, por ejemplo. Sin embargo, en el contexto de esta investigación se reserva el término *recurso* a aquellos componentes que desempeñan de forma activa una tarea/proceso (estación de trabajo, unidad de mecanizado, etc.) y que, por tanto, tendrán un comportamiento propio (ClassifierBehavior en SysML). A esta definición responden tanto los recursos por los que fluyen los materiales y productos procesados (almacenes, estaciones de mecanizado o de ensamble, etc.), como los sistemas de control, cuyas decisiones determinan el flujo de los materiales y las condiciones de trabajo.

Siguiendo con la estructuración de los recursos, es necesario tener en cuenta que en los sistemas reconfigurables que se pretenden simular la modularidad es una característica esencial, de manera que están diseñados para ser capaces de soportar la fabricación de diversos productos reemplazando algunos componentes o partes de los recursos. Así pues, algunos recursos admiten diversas configuraciones alternativas, que fundamentalmente son estructuras físicas diferentes, como el montaje de un determinado utillaje o herramienta, por ejemplo, aunque también se pueden considerar cambios de naturaleza soft (cargar un determinado código de control numérico, por ejemplo). Por tanto, a la hora de simular estos recursos se debe considerar un proceso de setup (cambiar de configuración y/o ajustar el recurso) previo a la ejecución de un determinado proceso de fabricación, así como las pertinentes variables y mecanismos que permitan controlar las modificaciones en las características de la representación del recurso.

Además de representar el sistema de fabricación, el otro elemento clave a considerar en las simulaciones de sistemas PPR es el producto, cuya representación puede tener diversos grados de complejidad y detalle en función del análisis planteado. En las simulaciones planteadas, los productos serán representados por las unidades de flujo,

cuyos atributos deben dar soporte a las características clave de los productos que participan del análisis propuesto. En el caso de que la simulación se limite a estudiar la productividad del sistema de fabricación y recrear el flujo de materiales, el producto participa como unidad material de flujo que se transmite de unos recursos a otros, si bien esta entidad que representa una unidad de producto debe contener información relativa al análisis planteado, como el instante de inicio de su fabricación, los tiempos de espera o de almacenamiento, los tiempos de procesado, etc. Más allá de este planteamiento, es posible plantear simulaciones más complejas en las que sea necesario incluir información de otra naturaleza (datos sobre la geometría, por ejemplo), incrementando la complejidad de la estructura de información y, en consecuencia, del modelo, como se comentará en próximas secciones (apartado 5.3). En cualquier caso, sea cual sea el detalle con el que se modela el producto, otra información que también es necesaria para cualquier tipo de simulación de sistemas de fabricación es la hoja de ruta de cada tipo de producto, es decir, datos sobre el plan de procesos nativo, que incluye la definición de los procesos de fabricación a los que debe ser sometido y los recursos específicos por los que debe pasar.

C. Comportamiento de los modelos de simulación

Aunque el modelado de la estructura y el comportamiento van de la mano, dedicamos este apartado a tratar con mayor detalle cuestiones comportamentales de los sistemas de simulación. De forma más específica, el discurso se centra en aquellos sistemas de simulación que dan soporte al análisis de sistemas dinámicos y discretos, como es el caso de los sistemas de fabricación por lotes sobre los que se pretende desarrollar el análisis de la productividad. Los sistemas de simulación que permiten este tipo de análisis dinámicos cuentan siempre con una variable que emula el paso del tiempo a modo de reloj, permitiendo utilizar dicha variable en expresiones matemáticas que dan soporte a comportamientos dinámicos continuos, o definir instantes temporales o la duración de determinados procesos en la simulación de comportamientos discretos. Además, en la simulación de eventos discretos se deben considerar los diferentes estados del sistema (espacio de estados), así como los eventos que desencadenan el paso de un estado a otro. Por ello, el comportamiento de los modelos de eventos discretos debe considerar, directa o indirectamente, aspectos que se pueden agrupar en tres conjuntos primitivos:

- *Conjunto de tiempo (Time set - T).* Número no negativo que representa el paso del tiempo, que es un valor real en el caso de modelos contínuos y entero en el caso de modelos discretos.

- *Espacio de estados (Space State - S).* Un modelo cambia con el tiempo de un estado a otro. Un estado actual en un momento determinado se cuantifica mediante los valores de una colección de variables de estado. El conjunto de todas las posibles combinaciones de variables de estado se llama espacio de estados.

- *Conjunto de eventos (Event set - E).* Un evento ocurre instantáneamente en un momento particular y puede causar un cambio de estado.

Alternativamente al modelado en base a estos conjuntos primitivos, otros modelos emplean otros conjuntos relacionados, como las actividades o las entidades, a partir de los cuales se pueden deducir los conjuntos primitivos [123]. En la literatura se han encontrado diferentes técnicas de modelado, destacando la clasificación propuesta en [123], en la que se clasifican los modelos de eventos discretos precisamente en función de su caracterización estructural, es decir, según los tipos de conjuntos empleados para dar soporte a cada orientación, siendo el tiempo el único tipo de información común a todas las categorías. Los cuatro tipos de modelos considerados en esta clasificación son:

- *Modelos orientados a estados*. Modelos en los que se definen los conjuntos S (puede ser finito o infinito), E (siempre finito) y T. En la representación de estos modelos, los nodos son estados, relacionados a través de transiciones que se activan al producirse determinados eventos. A este grupo pertenecen modelos como muchos modelos de Markov o autómatas clásicos.

- *Modelos orientados a actividades*. Modelos basados en conjuntos de lugares (Place set - P) y actividades (Activity set - A), además del tiempo (T). Un lugar representa una localización en un espacio, dentro del modelo (un nodo de un gráfico o un punto respecto de un sistema de coordenadas, por ejemplo), indicando dónde está una entidad (token). Por su parte, una actividad puede ser entendida como un comportamiento atómico que ocurre durante un tiempo, delimitada entre un evento de inicio y un evento de fin. Una actividad provoca que una entidad se mueva de un lugar a otro, lo que en las Redes de Petri se conoce como transiciones.

- *Modelos orientados a eventos*. Al igual que en los modelos orientados a estados, los orientados a eventos manejan los conjuntos S, E y T. Sin embargo, hay casos en los que el tamaño del espacio de estado es muy grande, manteniendo un número reducido de tipos de eventos. En estos casos, los eventos son definidos explícitamente, mientras que los estados son reemplazados por un conjunto de variables de estado. La combinación de estas variables define los múltiples estados posibles en el sistema, mientras que cada evento puede provocar una serie de cambios en las variables. En cuanto a la representación gráfica, esta vez los nodos no representen estados (como en la orientación a estados), sino eventos, y los arcos indican que un evento puede causar otro evento.

- *Modelos orientados a procesos*. Modelos definidos con la visión de la interacción de procesos, que capturan el comportamiento de una entidad que fluye a través del sistema e interacciona con recursos. Así, este enfoque diferencia entre componentes activos (procesos) y pasivos (recursos), de manera que los procesos provocan acciones que cambian el estado del sistema, mientras que los recursos no son capaces de provocar directamente acciones de cambio.

Además de la clasificación presentada anteriormente, la literatura también indica que existen diversos enfoques y formalismos que dan soporte al modelado del comportamiento. Por ejemplo, DEVS es uno de los formalismos más extendidos debido a ventajas que ofrece, como la facilidad de implementación, versatilidad y consumo computacional. Por su parte, la orientación a objetos, además del principio de

encapsulamiento, también establece mecanismos muy útiles desde la perspectiva del modelado de comportamiento, como son los mecanismos de herencia en las estructuras jerárquicas, las relaciones de composición que permiten la combinación o acoplamiento de modelos (soportando, por ejemplo, un DEVS acoplado) [123]. A la hora de desarrollar la propuesta de modelado, será necesario determinar el enfoque más adecuado en los diversos niveles del sistema (comportamientos atómicos, compuestos, globales, etc.) y los formalismos adoptados en cada caso, así como las capacidades y los conceptos manejados en el lenguaje de modelado utilizado.

En lo referido a las capacidades de SysML para modelar comportamientos dinámicos, este lenguaje ofrece tres alternativas: diagramas de las máquinas de estado, de actividades o de secuencia. A pesar de que un mismo comportamiento se puede modelar de diferentes formas, cada una de estas alternativas enfatiza en un aspecto diferente del comportamiento, ya sea su temporalidad, los flujos que activan determinadas operaciones, o los estados por los que pasa el sistema. En nuestro caso, la solución adoptada es la definición de máquinas de estado, puesto que es un formalismo de eventos discretos que cuenta con una representación gráfica con la que los usuarios pueden definir e interpretar este tipo de comportamiento de manera intuitiva, pero además es un formalismo que fácilmente se puede trasladar al formalismo DEVS. Sin embargo, el modelado de comportamientos con SysML cuenta con una limitación importante, y es que ofrece elementos de modelado muy generales (actividades, estados, etc.), que no permiten llegar a una implementación detallada del comportamiento (definir qué sucede en cada estado o actividad) con el propio lenguaje SysML. Para cubrir esta limitación, existe la posibilidad de definir expresiones opacas en las que definir anotaciones implementadas en otros lenguajes, como Java, C++ o incluso lenguaje natural. Obviamente, estas expresiones opacas no son interpretables en el entorno de modelado SysML ni son objeto de verificación en las comprobaciones de consistencia, pero pueden funcionar como anotaciones para escribir fragmentos de código en otros lenguajes que puedan ser útiles en la transformación de los modelos SysML a dichos lenguajes.

Por otra parte, conviene recordar que los sistemas de simulación se emplean principalmente para analizar sistemas estocásticos, en los que es necesario incluir componentes probabilísticos para obtener una solución estimada a partir de unas entradas o unas condiciones en las que influyen factores con cierta aleatoriedad. En el caso de la simulación del comportamiento discreto de los sistemas de fabricación, esta aleatoriedad afecta a los tiempos en los que se producen determinados eventos.

D. Consideraciones comportamentales del sistema referente

Como se ha comentado anteriormente, la simulación de los sistemas de fabricación presenta una estructura jerárquica de recursos que puede ser ciertamente compleja. A la hora de abordar el modelado del comportamiento de estos recursos, es fundamental determinar a qué nivel conviene abordar este comportamiento. En modelos construidos a partir de elementos de librería, como suele ser habitual en los sistemas de simulación, lo más conveniente es que cada uno de estos elementos tenga definido su propio comportamiento, de manera que el comportamiento del sistema complejo

que se pretende analizar emerge de la interacción entre sus componentes. Así, en el caso de la simulación de sistemas de fabricación, cada uno de los recursos atómicos debe contar con elementos suficientes que describan, de forma autónoma, su comportamiento. Los recursos más complejos, incluido el sistema de fabricación en su conjunto, tendrán un comportamiento que emerge de sus componentes y de las interacciones entre ellos. Por este motivo, resulta imprescindible definir el nivel más atómico en el que se aborda el modelado del comportamiento, y asegurar mecanismos que permitan la adecuada y consistente interacción entre elementos.

En el caso del desarrollo de este lenguaje, existen diversas soluciones válidas a la hora de analizar la productividad del sistema de fabricación y el comportamiento de las diversas etapas, en función del alcance o el nivel de detalle con el que se desea analizar el sistema. Sin embargo, atendiendo a los objetivos más globales de la investigación, los sistemas de simulación se extenderán posteriormente para incluir el análisis de la calidad, que se fundamentará en la simulación de ensambles de proceso, es decir, el montaje del producto, los utillajes y las herramientas empleadas en una etapa. Esta división del proceso, que engloba todas las operaciones realizadas en un mismo amarre (en el que se define un ensamble de proceso), es conocida como subfase. Así pues, en el desarrollo de este primer lenguaje centrado en la productividad también se supone que el nivel más atómico al que se analizará el sistema será la configuración de la unidad de producción para cada subfase.

Finalmente, conviene hacer una reflexión sobre cuál de los enfoques comentados en el punto anterior para modelar comportamiento discreto encaja mejor en el caso de la simulación de sistemas de fabricación. Por ejemplo, poniendo el foco de atención en el flujo de entidades (productos emulados) a través de los recursos simulados, en los que se activan una serie de comportamientos o acciones, este enfoque parece encajar con el escaneo de actividades adoptado en las Redes de Petri, en las que los nodos representan lugares (places) y las entidades fluyen de unos nodos a otros. Sin embargo, también resulta interesante el enfoque adoptado en la interacción de procesos, en el que se muestra una interacción entre los procesos (componentes activos) y los recursos (componentes pasivos). Esta alternativa encajaría más en un modelo de simulación estructurado según la funcionalidad, centrada en la representación de los procesos y donde los recursos de fabricación únicamente se consideran como objetos que deben estar disponibles para poder activar un proceso. Sin embargo, esta orientación no encaja con la estructuración adoptada, orientada a replicar la arquitectura física del sistema de fabricación. Por último, la programación de eventos se centra en los instantes en que se producen determinados cambios, que se representan como nodos. Sin embargo, SysML no dispone de elementos y diagramas que den soporte a este enfoque. Por tanto, se puede concluir que para el modelado de comportamiento acoplado se adapta el escaneo de actividades, de manera que las entidades (unidades de flujo) fluyen a través de las localizaciones del sistema (que representan estaciones de trabajo), activando a su paso una serie de comportamientos en cada uno de estos componentes. Conviene apuntar que el flujo de estas entidades estará gobernado por una serie de reglas y eventos que se detallarán en el Capítulo 6, dedicado al desarrollo de elementos de librería.

5.2.2. Descripción del metamodelo para simulación de sistemas de fabricación

En el presente apartado se presenta un modelo conceptual que describe el metamodelo del lenguaje propuesto para el modelado de sistemas de simulación para el análisis de sistemas de fabricación discretos. Para ello, además de la descripción textual de los conceptos y relaciones consideradas, se incluyen algunos diagramas conceptuales (diagramas UML) para representar la sintaxis abstracta de este lenguaje. El apartado se estructura en un primer bloque dedicado a introducir y describir los conceptos generales para el modelado del sistema de simulación, y un segundo bloque dedicado a introducir y definir los conceptos para la representación del sistema de fabricación en el modelo de simulación.

A. Sistema de simulación: elementos básicos

Un sistema de simulación es un sistema software y, por tanto, es un sistema formado por entidades software que presentan un comportamiento dinámico que transforman, de acuerdo a unos algoritmos y cálculos, los datos que representan el resto del sistema referente simulado. Ahora bien, atendiendo a las diferentes funciones de cálculo/transformación que son necesarias para construir el sistema de simulación, el metamodelo propuesto contempla diversos tipos de elementos que se representan en la Figura 5.4.

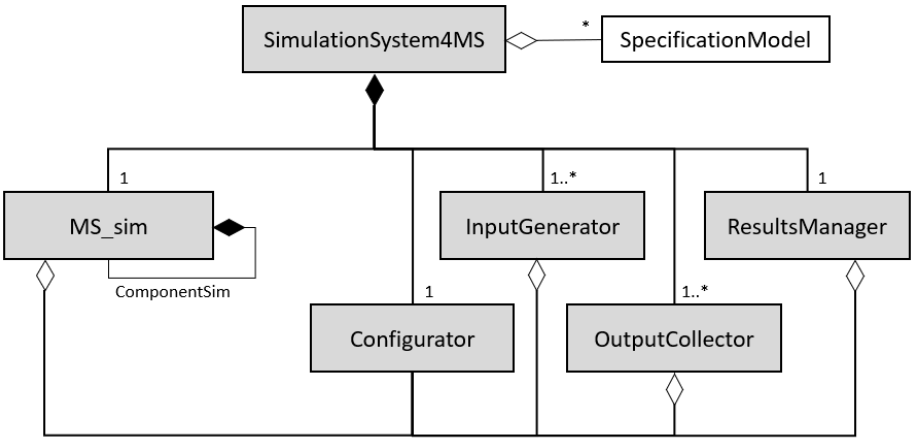


Figura 5.4. Sintaxis abstracta de sistemas de simulación.

Estos conceptos, especificados para el caso particular de la simulación de sistemas de fabricación, responden a las siguientes definiciones:

- *SimulationSystem4MS*. Modelo del sistema de simulación en su conjunto, que aunque en este caso es específico para la simulación de un sistema de fabricación, su estructura responde a la de un sistema general para la simulación de otros tipos de sistemas referentes. Puesto que gran parte de los datos utilizados en su definición representan las características del sistema referente a simular, se establece una relación de agregación con el modelo de especificación del sistema referente, que permitirá el acceso a su información y el establecimiento de mecanismos que aseguren la consistencia entre modelos.

- *MS_sim*. Parte del sistema de simulación que emula el comportamiento del sistema de fabricación, que es el sistema referente considerado. Cabe indicar que, como se detalla más adelante, los recursos de fabricación pueden ser complejos y estar formados por otros recursos. Siguiendo esta estructuración de los recursos, el sistema de fabricación puede ser considerado como un tipo de recurso (como se representa en la Figura 5.6 mediante la relación de especialización de ProcessingResource_sim) y, en particular, el recurso de mayor nivel de agregación considerado en el análisis (no existe ningún recurso que contenga un MS_sim, ya que su único poseedor puede ser un elemento de tipo SimulationSystem4MS). Como se trata del elemento central de la simulación, se comentará con el debido detalle en los próximos apartados.

- *Configurator*. Elemento estructural (no tiene comportamiento propio) que agrupa todos los parámetros característicos de la simulación, facilitando al usuario editar sus valores para definir los experimentos particulares. Estos parámetros pueden formar parte de la definición del MS_sim (por ejemplo, el valor medio y desviación típica de los tiempos de proceso, capacidades máximas de los buffers, etc.), ser datos relativos a las especificaciones (tanto de los tipos de producto procesados como de los recursos), o caracterizar otros elementos del sistema de simulación (por ejemplo, frecuencia media del lanzamiento de órdenes en el InputGenerator). Por este motivo, los datos manejados en el Configurator deben ser accesibles para el resto de componentes del sistema de simulación, definiendo relaciones de agregación (referencias) entre componentes.

- *InputGenerator*. Elemento cuya función es generar/crear las unidades de flujo que representan los insumos materiales, que en el caso considerado son las principales entradas (inputs) para el MS_sim. Por tanto, los InputGenerators representan el inicio de un flujo de entidades que posteriormente transcurre por el MS_sim y que representa el flujo logístico o flujo de materiales.

- *OutputCollector*. Elemento sumidero en el que se destruyen las entidades del flujo simulado. Por tanto, el OutputCollector es el punto final del flujo de datos que representa el flujo de materiales.

- *ResultsManager*. Elemento encargado de recopilar información de interés de los diversos componentes del sistema de simulación y dar soporte al tratamiento matemático/estadístico necesario para obtener, a partir de los datos simulados, los indicadores y medidas de rendimiento deseados, generalmente expresados en los requerimientos del sistema de simulación.

La construcción de un modelo de simulación, además de especificar los elementos comentados, también implica establecer relaciones (asociaciones) entre ellos, que se concretan en conexiones entre puertos. Por estos puertos, denominados de forma genérica DataPort, se intercambian datos de la simulación que pueden representar diferentes realidades del sistema referente. Con el fin de identificar algunas de estas realidades para el caso de la simulación de sistemas de fabricación, se proponen dos especializaciones del DataPort, representadas en la Figura 5.5 y descritas a continuación:

- *FU_Port*. Puerto que da soporte a la transferencia de datos sobre las unidades de flujo simuladas, es decir, las entidades que representan lotes de productos, dando soporte al flujo logístico.

- *C_Port*. Puerto que da soporte a la transferencia de datos sobre la comunicación entre recursos, el flujo de control y la sincronización de tareas y comportamientos.

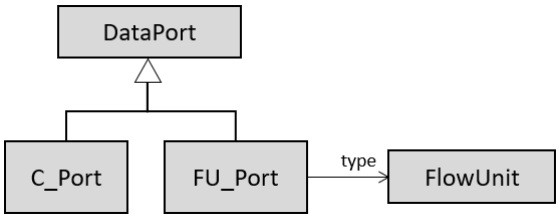


Figura 5.5. Sintaxis abstracta de puertos en un sistema de simulación.

Conviene comentar además que, en ocasiones, para facilitar la tarea de modelado y reducir el número de conectores, es interesante integrar diversos puertos en un único puerto más complejo. Existen diversos mecanismos para lograr esto, como por ejemplo la especialización o la anidación. Sin embargo, estas consideraciones serán aplicadas más adelante en tareas de modelado de nivel M1, tanto en la construcción de librerías como en el desarrollo de modelos de simulación particulares, pero no tendrán efecto a nivel de lenguaje. Esto es debido a que las reglas que se pretenden considerar en los conceptos manejados a nivel de metamodelo están centradas en la semántica del dominio, y no incorporan aspectos en cuanto a la estrategia de modelado, como puede ser la agrupación de construcciones.

B. Sistema de simulación: estructura del MS_Sim

Este punto se dedica a describir los aspectos vinculados con el modelado estructural del MS_sim. La Figura 5.6 representa los principales conceptos de la sintaxis abstracta que emulan los diversos tipos de recursos de fabricación, incluido el sistema de fabricación en su conjunto.

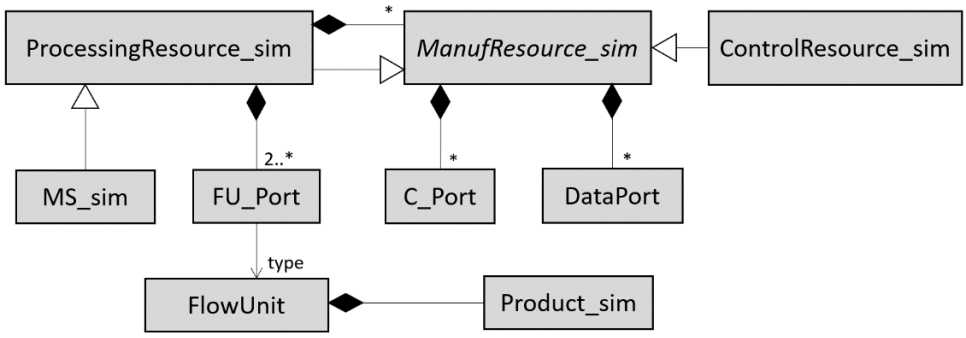


Figura 5.6. Sintaxis abstracta de sistemas de fabricación simulados.

A continuación, se describen los nuevos conceptos del metamodelo incorporados en la Figura 5.6, especialmente aquellos que dan soporte a la simulación de diversos tipos de recursos de fabricación.

- *ManufResource_sim*. Concepto abstracto que representa cualquier recurso del sistema de simulación, tanto recursos procesadores como de control y, por tanto, este concepto se especializa en ProcessingResource_sim y ControlResource_sim. Un aspecto característico de cualquier ManufResource_sim es que posee puertos de tipo DataPort y C_Port para dar soporte al flujo de datos, y en especial aquellos que representan información de control. Además, estos elementos siempre tendrán un comportamiento, como se comentará en detalle en el siguiente punto C dedicado al comportamiento.

- *ProcessingResource_sim*. Especialización del ManufResource_sim que emula un recurso procesador, es decir, un recurso por el que fluye una unidad de flujo (lote de productos), ejecutando unos procesos que modifican algunas de las propiedades tanto de los lotes (localización, tiempos de flujo, etc.) como de los productos que lo forman (por ejemplo, su estado de fabricación). Por tanto, un ProcessingResource_sim maneja y transforma datos relacionados con los productos y los procesos de fabricación simulados. Un ProcessingResource_sim puede estar compuesto por otros ManufResource_sim, tanto de procesado como de control. Además, debe contar con FU_Ports que permitan establecer conexiones con otros ProcessingResource_sim, para dar soporte al flujo de datos que representa el flujo logístico. Conviene recordar que el propio MS_sim es una especialización del ProcessingResource_sim.

- *ControlResource_sim*. Especialización del ManufResource_sim que emula un recurso controlador, es decir, aquellos elementos del sistema de fabricación que no participan del flujo logístico, sino únicamente del flujo de control, soportando la monitorización y toma de decisiones. Se establece que el MS_sim debe contar con un ControlResource_sim como unidad de control central, conectado con los otros componentes de tipo ProcessingResource_sim. Para el resto de ProcessingResource_sim el componente de tipo ControlResource_sim es opcional.

- *FlowUnit*. Elemento que representa una unidad de flujo, es decir, un lote de productos, que contendrá un número de productos mayor o igual a la unidad. Por tanto, estará compuesto por uno o más Product_sim, que representan cada unidad material o de producto. Cualquier puerto del tipo FU_port debe ser de un tipo FlowUnit.

Considerando los diferentes tipos los recursos de fabricación procesadores, conviene diferenciar dos especializaciones del ProcessingResource_sim:

- *TransformResource_sim*. Elemento que emula un recurso de fabricación transformador en el que el producto sufre un cambio en sus características físicas. Aunque en este lenguaje no se definan de forma explícita dichas características, lo que sí se considera es que la unidad de flujo entrante (lote de insumos materiales) no es igual a la de salida (lote de productos procesados). Por tanto, los FU_Ports de entrada y salida deben estar definidos mediante bloques FlowUnit diferentes.

- *LogisticResource_sim*. Elemento que emula un recurso de fabricación logístico, es decir, que participa en el flujo de materiales, pero no modifica las características físicas de los productos procesados, aunque puede influir en

otras propiedades como incrementar el tiempo de producción total del producto. Dos claros ejemplos de este tipo de recursos son los almacenes y los transportes. Dado que no se modifica estructuralmente el producto, en los LogisticResource_sim los FU_Ports de entrada y salida deben ser del mismo tipo.

Comentadas estas especializaciones, conviene indicar además que cualquier elemento que simule un recurso de fabricación (ManufResource_sim), sea cual sea su tipo específico, tiene asociada una especificación, es decir, unos datos de diseño. Este tipo de información no pertenece al elemento ManufResource_sim, puesto que puede afectar a varios recursos del mismo tipo. La solución adoptada es definir los datos de especificación en el bloque Configurator, que centraliza la definición de parámetros de simulación, y relacionar mediante relaciones de agregación (referencia) los datos de diseño con los parámetros de cada ManufResource_sim. Además, en el caso de los recursos transformadores, es interesante considerar que pueden ofrecer diversas configuraciones para ejecutar diversas operaciones de fabricación. Para caracterizar esta configurabilidad, los TransformResource_sim tienen siempre una propiedad llamada ActiveConfiguration que identifica la configuración adoptada en cada instante. La Figura 5.7 representa estos conceptos y relaciones.

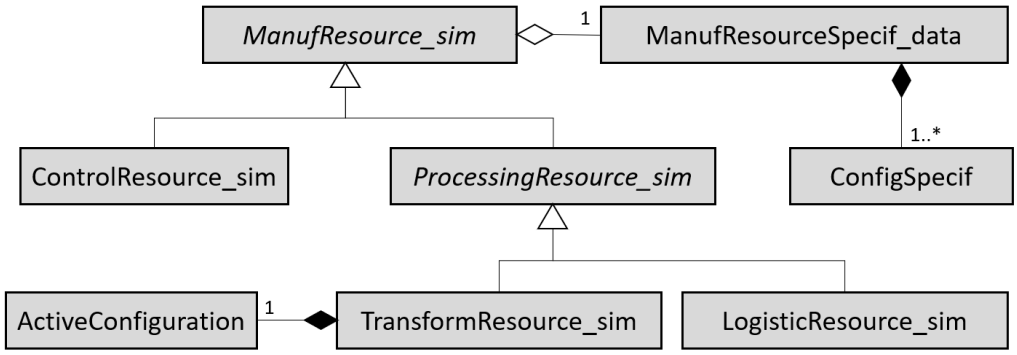


Figura 5.7. Sintaxis abstracta de la simulación de recursos de fabricación.

Otro elemento fundamental en la simulación de sistemas PPR es la emulación de los productos procesados, incluyendo la definición de datos sobre sus especificaciones y la caracterización de los productos emulados en función de los intereses del análisis. A la hora de modelar la especificación del producto (ProductSpecif_data) es necesario definir algunos aspectos, como por ejemplo, el plan de procesos nativo, detallando tanto los procesos como los recursos por los que debe pasar para su fabricación. A pesar de que en los sistemas de simulación ejecutables esta información puede estar codificada en estructuras de datos como vectores o matrices, en esta propuesta se explotan las capacidades de SysML para definir el plan de procesos nativo como una actividad (NativeProcessPlan). Esta orientación permite al usuario introducir la información relativa a las hojas de ruta de una manera más gráfica y descriptiva. Sin embargo, durante la transformación a modelos ejecutables, esta parte del modelo será interpretada y definida de la forma más sencilla posible (vectores o matrices de datos) para lograr un modelo lo más ligero posible. Así pues, cada actividad de tipo NativeProcessPlan está compuesta por una o más acciones (ManufProcess) que representan procesos de fabricación. Se define además un atributo booleano

(isAtomic) para identificar los procesos atómicos que, en esta propuesta, representarán subfases, es decir, operaciones de fabricación ejecutadas en un mismo amarre. Además, puesto que se pretende definir el plan de procesos nativo, es necesario asignar a cada subfase el recurso concreto del sistema de fabricación en la que se lleva a cabo. Para ello, se considera el concepto ResourceAllocation, que es una especialización del AllocateActivityPartition de SysML, un elemento que específicamente tiene esta funcionalidad: asignar conjuntos de acciones a determinados componentes del sistema.

Además del plan de procesos, la especificación debe definir otras características de producto como la identificación de los estados por los que pasa el producto (StateSpecif) a lo largo de su fabricación, y que define el tipo de objeto que fluye de un proceso a otro. En función del tipo de análisis, estos estados pueden contener información de diferente naturaleza, como, por ejemplo, el modelado geométrico del producto en cada estado, una cuestión tratada en la sección 5.3 pero que de momento no se considera en este planteamiento.

Por otra parte, a la hora de emular los productos procesados (Product_sim) es necesario relacionar cada uno de estos elementos con su respectiva especificación (ProductSpecif_data) mediante una relación de agregación (referencia). Además, debe caracterizarse cada Product_sim con unas propiedades básicas, entre las que se encuentra la identificación del estado de fabricación en el que se encuentra el producto emulado (ActiveState), que apunta a uno de los StateSpecif definidos en la especificación. La Figura 5.8 describe gráficamente algunos de estos conceptos.

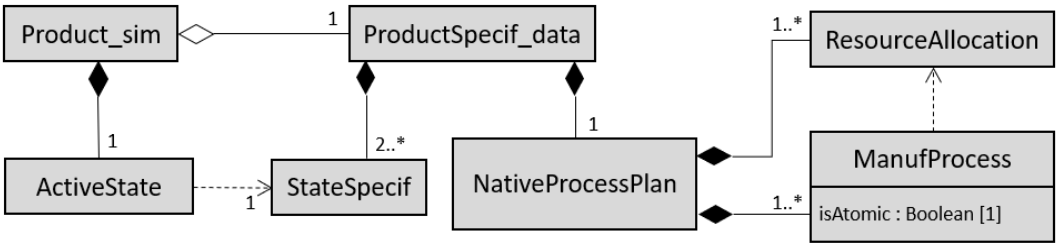


Figura 5.8. Sintaxis abstracta de producto emulado y su especificación.

C. Sistema de simulación: comportamiento discreto

Este apartado se centra en establecer los principios básicos a nivel de lenguaje que gobernarán el modelado del comportamiento en los sistemas de simulación propuestos. Como es bien sabido, las simulaciones planteadas deben emular el comportamiento dinámico característico de los sistemas de fabricación por lotes que se pretenden analizar. En este sentido, tal y como se ha comentado en el apartado 5.1.1.C, existen diversos enfoques válidos para el modelado de comportamientos dinámicos discretos, cada uno de los cuales pone el foco en unos determinados aspectos (eventos, actividades, procesos, ...). A continuación, se presenta la solución adoptada en esta propuesta, que combina algunos de estos enfoques en función del nivel de granularidad tratado.

En primer lugar, conviene aclarar que todos los recursos de fabricación tienen un comportamiento. Sin embargo, según su complejidad, este comportamiento puede ser implementado o no de forma explícita en el modelo de simulación. En el caso de los recursos de fabricación más complejos, como es el propio sistema de fabricación en su conjunto, este comportamiento no se puede modelar directamente, sino que emerge de elementos comportamentales más simples que se acoplan, interactuando entre sí. Este mecanismo de composición, tan característico de la orientación a objetos, también está presente en formalismos como DEVS, en el que se diferencia entre el DEVS atómico y el DEVS acoplado. Por tanto, es necesario definir una serie de elementos del modelo que, al menos a nivel de comportamiento, serán atómicos, es decir, no pueden poseer componentes con comportamiento. La composición de estos elementos comportamentales atómicos y las interacciones entre ellos deben dar soporte a comportamientos acoplados. En este sentido, los recursos de fabricación considerados atómicos serán los que representen estaciones de trabajo (capa 4 de la Figura 5.3). Cualquier recurso de fabricación de capas superiores que se simule tendrá un comportamiento acoplado.

Sin embargo, además de los elementos que representan partes del sistema de fabricación, el sistema de simulación tiene otros componentes que también deben tener su propio comportamiento, como es el caso del InputGenerator o del OutputCollector. Con el fin de unificar la forma en que el comportamiento se modela en esta propuesta, se ha creado una metaclase abstracta (que no se puede instanciar) para representar cualquier tipo de elemento con comportamiento, a la que se le llama *BehavioralElement_sim*. Esta metaclase cuenta con un atributo booleano (isAtomic) que permite diferenciar entre aquellos elementos atómicos y compuestos. Esta diferenciación será importante puesto que únicamente los de tipo atómico tienen un comportamiento definido explícitamente. Además de identificarlos por el valor del atributo isAtomic, se establece que los elementos atómicos no pueden tener propiedades de tipo BehavioralElement_sim, es decir, partes con comportamiento propio. En cambio, en el caso de los BehavioralElement_sim compuestos (isAtomic=false), no tendrán un comportamiento definido explícitamente, pero necesariamente estarán compuestos por otros BehavioralElement_sim (atómicos o no), de manera que el comportamiento de la construcción compuesta emerge de sus partes y las conexiones entre ellas.

Por su parte, el comportamiento definido en los elementos atómicos queda representado con la metaclase STM_MainBehavior, que es una máquina de estados que está activa desde que el objeto o instancia de dicha clase existe (equivalente al concepto ClassifierBehavior de SysML). Esta máquina de estados puede resolverse mediante el formalismo DEVS atómico en el modelo de simulación, y está principalmente vinculada con la orientación a estados descrita anteriormente. Atendiendo a la forma de modelar elementos con comportamiento en SysML, otra característica de los BehavioralElement_sim atómicos es que sus puertos deben definirse como BehaviorPorts, de manera que su comportamiento accede y utiliza la información contenida en los puertos de este tipo sin necesidad de conectarse con partes internas del bloque. Esto no ocurre en los BehavioralElement_sim no atómicos, cuyos puertos deben conectarse con las partes internas para que de dicha estructura

emerja el comportamiento. La Figura 5.9 muestra gráficamente algunos de los conceptos y relaciones mencionados, representando además algunas de las especializaciones de BehavioralElement_sim como InputGenerator y OutputCollector.

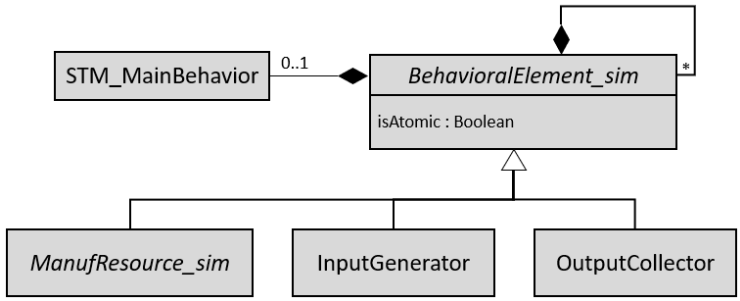


Figura 5.9. Sintaxis abstracta de elementos con comportamiento.

5.2.3. Implementación del metamodelo: perfil SysML4MSSim

Atendiendo al metamodelo descrito en el apartado anterior, este apartado presenta la implementación del lenguaje como un perfil SysML llamado SysML4MSSim (SysML for Manufacturing System Simulation). A continuación, se detallan los estereotipos considerados, incluyendo su definición y la descripción en lenguaje natural de las reglas incluidas en cada estereotipo. El perfil está estructurado en dos paquetes, como se muestra en la Figura 5.10, con el fin de agrupar conceptos y estereotipos de diferentes ámbitos, aunque los estereotipos de diferentes paquetes pueden estar interrelacionados. Estos dos paquetes son:

- SSMPP → Simulation System Modeling Profile Package
- MSSPP → Manufacturing System Simulation Profile Package

A continuación, se dedica un subapartado a cada uno de estos paquetes, en los que se enumeran en formato de tabla los estereotipos definidos, incluyendo una breve definición de los mismos. La especificación completa del perfil, incluyendo las reglas OCL definidas en cada estereotipo, se recoge en el Anexo A.1.

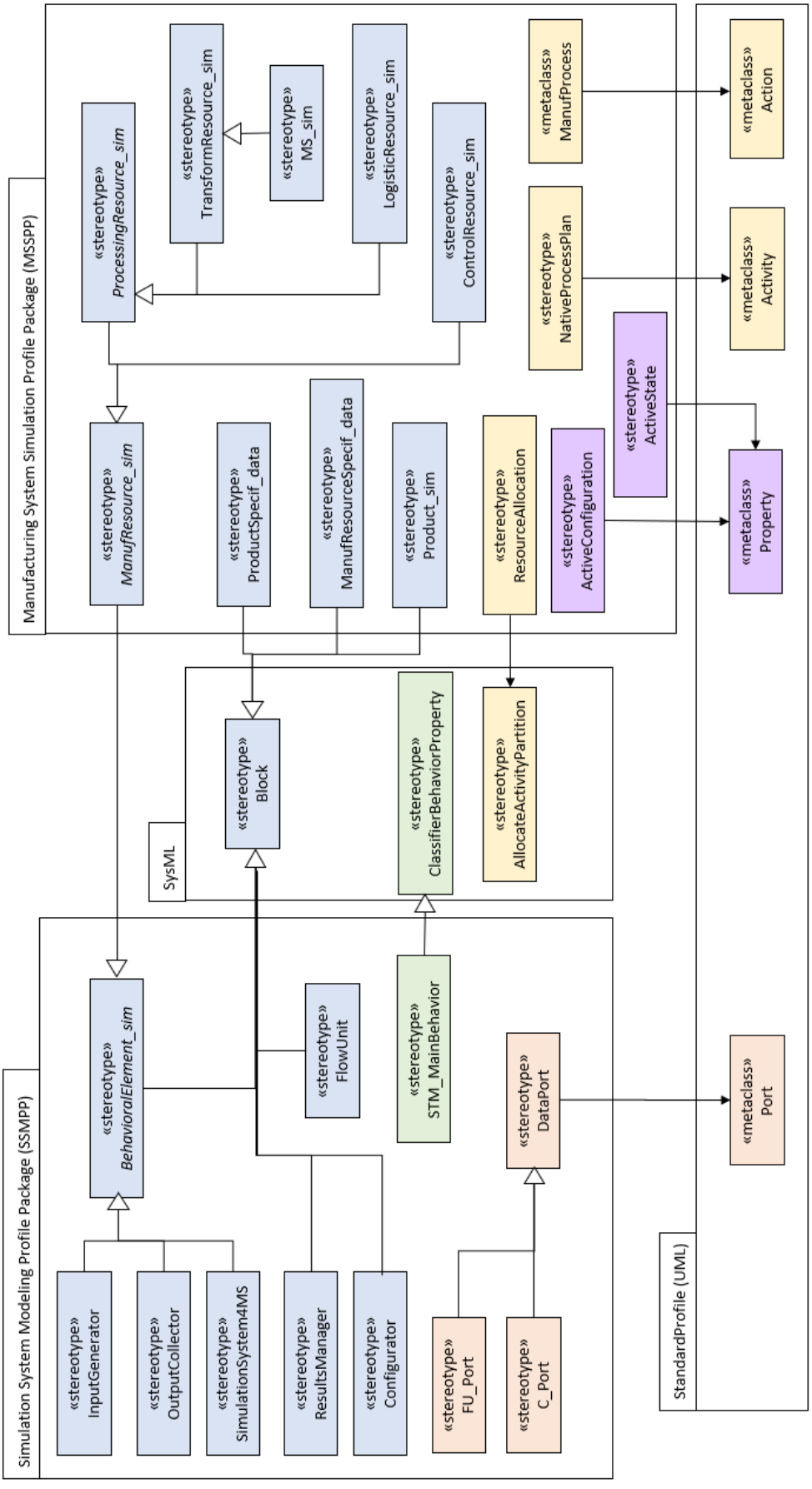


Figura 5.10. Diagrama de paquetes del perfil SysML4MSSim.

A. Simulation System Modeling Profile Package (SSMPP)

Tabla 5.1. Estereotipos del paquete SSMPP.

Estereotipo	**Descripción**
«BehavioralElement_sim»	Estereotipo abstracto que extiende el bloque para representar cualquier parte del sistema de simulación (o el sistema en su conjunto) que tiene un comportamiento. Un atributo booleano (isAtomic) distingue los elementos atómicos de los compuestos. Los atómicos deben tener definido un comportamiento «STM_MainBehavior», mientras que los compuestos no pueden tener este comportamiento definido. Los compuestos están formados por una o más partes de tipo *«BehavioralElement_sim»*.
«STM_MainBehavior»	Especialización del estereotipo «ClassifierBehaviorProperty» que representa el comportamiento propio de un «BehavioralElement_sim» Block de tipo atómico. Este comportamiento debe ser siempre definido con una máquina de estados.
«SimulationSystem4MS»	Bloque que representa el sistema de simulación en su conjunto. Entre sus propiedades, debe tener una parte tipeada por un «Configurator» Block, otra por «ResultsManager» Block y otra por «MS_sim» Block. Además, debe tener al menos una propiedad tipeada por un «InputGenerator» Block y al menos otra tipeada por un «OutputCollector» Block.
«Configurator»	Bloque que funciona como estructura de datos en el que se definen los principales parámetros del sistema de simulación.
«InputGenerator»	Especialización del *«BehavioralElement_sim»* para identificar los bloques en los que se inicia el flujo de datos que representa el flujo de materiales, creando unidades de flujo con cierta periodicidad en función del comportamiento definido. Debe tener al menos un puerto de tipo «FU_Port».
«OutputCollector»	Especialización del *«BehavioralElement_sim»* para identificar los bloques en los que termina el flujo de datos que representa el flujo de materiales, destruyendo las unidades de flujo a su llegada a este bloque. Debe tener al menos un puerto de tipo «FU_Port».
«ResultsManager»	Bloque en el que se definen los cálculos necesarios para obtener las medidas de rendimiento deseadas a partir de los datos manejados en la simulación. Para ello, debe tener al menos un puerto de tipo «DataPort» a través del cual obtener dichos datos de otras partes del sistema de simulación.
«DataPort»	Puerto que se emplea para intercambiar datos con otras partes del sistema de simulación.
«FU_Port»	Especialización del «DataPort» para identificar aquellos puertos en los que se manejan datos que representan unidades de flujo y que, por tanto, participan del flujo de datos que representa el flujo de materiales. Estos puertos están siempre definidos por «FlowUnit» Blocks.
«C_Port»	Especialización del «DataPort» para identificar aquellos puertos en los que se manejan datos relacionados con el control y la sincronización

	de los procesos, ya sea entre partes que representan recursos procesadores o entre éstos y la unidad de control central.
«FlowUnit»	Bloque que representa una unidad de flujo, es decir, un lote de productos, y por tanto está compuesto por al menos una parte de tipo «Product_sim».

B. Manufacturing System Simulation Profile Package (MSSPP)

Tabla 5.2. Estereotipos del paquete MSSPP.

Estereotipo	**Descripción**
«ManufResource_sim»	Estereotipo abstracto que es una especialización del estereotipo *«BehavioralElement_sim»*. Se define para identificar (a través de sus especializaciones) cualquier bloque en el que se defina un recurso de simulación, es decir, las partes del sistema de simulación que representan recursos del sistema de fabricación. Cada bloque estereotipado con especializaciones de «ManufResource_sim» debe tener una relación de agregación con un «ResourceSpecif_data».
«ManufResourceSpecif_data»	Bloque que se emplea para definir la información de especificación relativa a un determinado tipo de recursos de fabricación considerados en la simulación. Cuenta con una propiedad booleana (isTransformer) para identificar las especificaciones de recursos transformadores, ya que éstos deben tener al menos una propiedad «ConfigSpecif».
«ProcessingResource_sim»	Estereotipo abstracto que es una especialización del estereotipo *«ManufResource_sim»*. Se define para identificar los bloques en los que se define un recurso procesador, es decir, aquellas partes del sistema de simulación que representan recursos del sistema de fabricación y por las que fluyen unidades materiales o de producto. Por tanto, los «ProcessingResource_sim» deben tener al menos dos puertos de tipo «FU_Port» Port. Este estereotipo cuenta además con un atributo (type), de tipo ProcessingResourceTypes para diferenciar entre los bloques que representan recursos de fabricación transformadores frente a los de carácter logístico.
«TransformResource_sim»	Especialización del *«ProcessingResource_sim»* para representar los recursos de fabricación transformadores, es decir, aquellos que modifican las características del producto y añaden valor (estaciones de ensamblaje, mecanizado, etc.). Debe poseer una propiedad estereotipada como «ActiveConfiguration»
«ActiveConfiguration»	Propiedad de un «TransformResource_sim» Block que identifica la configuración actual del recurso de fabricación.
«ConfigSpecif»	Propiedad de un «ManufResourceSpecif_data» Block que define una de las posibles configuraciones del recurso de fabricación.
«LogisticResource_sim»	Especialización del *«ProcessingResource_sim»* para representar los recursos de fabricación logísticos, es decir, aquellos que no modifican las características del producto (almacenes, transportes, etc.). Un «LogisticResource_sim» Block no puede tener ninguna propiedad tipeada por un «TransformResource_sim» Block.

«MS_sim»	Especialización del «TransformResource_sim» que se emplea para identificar el bloque que emula el sistema de fabricación en su conjunto en el sistema de simulación. Por ser el recurso de fabricación raiz, el modelo únicamente puede contener un uso del «MS_sim» Block, poseído por un «SimulationSystem4MS» Block.
«ControlResource_sim»	Especialización del *«ManufResource_sim»* para identificar los bloques del sistema de simulación que emulan un recurso de control, es decir, aquellas partes del sistema de fabricación encargadas de la monitorización, control y toma de decisiones, pero sin fluir ningún tipo de material por ellos. Por tanto, estos bloques no pueden tener ningún «FU_Port», pero sí deben tener al menos un «C_Port» para soportar el intercambio de datos sobre el control de los procesos.
«Product_sim»	Bloque se emplea para definir la información relativa a las unidades de producto, cuya composición define una unidad de flujo (lote) que fluirá por el sistema. Los «Product_sim» Blocks deben tener una relación de agregación (referencia) con un «ProductSpecif_data» Block y una propiedad estereotipada como «ActiveState»
«ActiveState»	Propiedad de un «Product_sim» Block que identifica el estado de fabricación en el que se encuentra el producto simulado.
«ProductSpecif_data»	Bloque se emplea para definir la información de especificación relativa a un tipo de producto considerado en la simulación. Debe incluir una «NativeProcessPlan» Activity.
«StateSpecif»	Propiedad de un «ProductSpecif_data» Block que define uno de los estados por los que pasa el producto simulado durante su fabricación.
«NativeProcessPlan»	Actividad mediante la cual se definen los diversos procesos por los que debe pasar un determinado tipo de producto durante su fabricación. Todas las acciones definidas deben ser de tipo «ManufProcess» Action
«ManufProcess»	Acción que necesariamente pertenece a una «NativeProcessPlan» Activity y que representa un proceso de fabricación en el plan de producción nativo. Cuenta con un atributo booleano (isAtomic) para identificar los procesos atómicos, que en esta propuesta se referirá a las subfases, es decir, todas las operaciones de fabricación que se realizan en una misma máquina y un mismo amarre. Además de identificar el proceso, cada acción debe estar contenida en un «ResourceAllocation» para indicar el recurso de fabricación atómico en que se ejecuta.
«ResourceAllocation»	Especialización del estereotipo «AllocateActivityPartition» de SysML empleado para asignar recursos a uno o más procesos del plan de procesos nativo. Por tanto, un «ResourceAllocation» siempre estará definido dentro de un «NativeProcessPlan» Activity, y todas las acciones contenidas deben ser de tipo «ManufProcess» Action.

5.3. Lenguaje para el análisis de tolerancias

En esta sección se define un lenguaje para el modelado en SysML de artefactos físicos (piezas/ensambles), asegurando la consistencia de los modelos orientados al análisis de superficies y de las relaciones entre ellas. Este lenguaje aborda tanto la definición nominal de la geometría como las tolerancias y las geometrías desviadas, poniendo especial atención en la definición de las relaciones tanto entre superficies de una misma pieza (en función de unas especificaciones geométricas) como entre superficies de piezas diferentes (relaciones de ensamble).

El enfoque de esta sección es generalista con el fin de presentar un lenguaje que puede ser utilizado para el modelado de cualquier tipo de artefactos. Sin embargo, conviene aclarar que el fin último de esta propuesta es dar soporte al modelado geométrico del ensamble de proceso (producto y utillaje considerados en el amarre), que será el elemento central para el análisis de la calidad geométrica en los sistemas de simulación propuestos, superando las limitaciones o simplificaciones (linealizaciones) asumidas en los modelos de simulación desarrollados en las experiencias previas (Capítulo 3). Por lo tanto, los conceptos de este lenguaje se emplearán en la construcción de sistemas de simulación orientados al análisis de la calidad geométrica en sistemas de fabricación multietapa, pero esta será una cuestión abordada en la sección 5.4. Además, cabe comentar que una versión extendida del lenguaje fue presentada en [31], incluyendo aspectos vinculados con la definición y análisis de cadenas funcionales. Sin embargo, en esta sección únicamente se tratarán las cuestiones relevantes de cara a la definición de artefactos que den soporte a los sistemas de simulación propuestos.

5.3.1. Introducción

El modelado geométrico y el toleranciado de productos son cuestiones muy presentes en el dominio de la ingeniería mecánica, siendo tratadas en múltiples normas y publicaciones científicas, como se ha comentado durante el Capítulo 2. Además de los estándares y normas ampliamente consolidados, en las últimas décadas son muchos los trabajos que se han centrado en el modelado de productos considerando su estructura física y la especificación GD&T con diversos enfoques o propósitos. De entre la numerosa literatura publicada en los últimos años, esta revisión se ha centrado en estudiar aquellas propuestas del dominio de la Ingeniería Mecánica que adoptan SysML, y prestando especial atención a aquellas que tratan el modelado de superficies. Por ejemplo, algunos trabajos de los últimos años han explorado la integración de modelos SysML con otros modelos específicos de dominio, como en [124] o [125], o el mapeo de los modelos específicos de dominio en un modelo de sistema genérico común compatible con SysML [126]. Más centrada en el modelado de superficies es la propuesta “C&C-M for SysML” [127] [128], que plasma los conceptos básicos del enfoque Contact and Channel [129] en SysML. Entre los trabajos centrados en la definición de perfiles SysML destacan SysML4Mechatronics [130], un perfil orientado a modelar sistemas mecatrónicos en los que integrar diversas vistas, o SysML4FMArch [131], un perfil desarrollado para dar soporte al modelado de arquitecturas funcionales de sistemas mecánicos. Otro ejemplo es el perfil GERTRUDe [15] enfocado en la especificación de requerimientos geométricos en etapas tempranas de diseño y el

modelado de algunos componentes de arquitecturas físicas, incluyendo restricciones basadas en TTRS. La adopción de los principios de TTRS y su implementación como perfil SysML han sido una referencia importante para el desarrollo de este trabajo.

Sin embargo, estos trabajos abordan el diseño físico (*embodiment*) con un enfoque conceptual, identificando superficies y sus relaciones, pero son pocos los trabajos que utilizan las capacidades de SysML (o perfiles SysML) para modelar matemáticamente tolerancias, dando soporte al análisis de variabilidad geométrica mediante simulación, un tema fundamental en esta investigación. Una de las referencias que se centran en esta problemática es [132], un trabajo en el que se presenta una librería de elementos modelados en SysML para representar elementos de toleranciado, como diversos datums y zonas de tolerancia. Esta propuesta está basada en el modelo TTRS para definir las restricciones de posición relativas a las condiciones impuestas entre features. Sin embargo, se centra en el desarrollo de elementos de librería, pero no aborda el modelado de tolerancias a nivel de lenguaje.

Ante esta situación, se decidió desarrollar un lenguaje propio para el modelado y análisis de tolerancias que está basado en algunos antecedentes que se detallan a continuación. Para ello, el resto del apartado se estructura alrededor de las siguientes cuestiones:

- *Modelado conceptual de ensambles*. Análisis de propuestas sobre modelado de artefactos (piezas y ensambles) y la especificación de su geometría basada en el concepto de feature.
- *Modelado TTRS de superficies y sus relaciones*. Análisis del modelo TTRS, que se adopta como principal referencia para la representación de superficies y las relaciones entre éstas según su clase de invarianza.
- *Modelado matemático de las desviaciones geométricas*. Análisis del uso de torsores de pequeños desplazamientos en el modelado de desviaciones geométricas.

A. Modelado conceptual de ensambles

El modelado de ensambles, como parte de la definición del principio de solución en etapas tempranas de diseño, ha sido una de las áreas de investigación fundamentales para mejorar la interoperabilidad en entornos CAx. Entre los metamodelos que se han propuesto con esta orientación destacan los desarrollados por el NIST, fundamentalmente el Core Product Model (CPM) [57] y el Open Assembly Model (OAM) [56]), y ciertas propuestas adicionales que los extienden, como el Modified OAM [58]). Estas referencias manejan múltiples conceptos, expuestos en modelos UML, aunque desde la perspectiva de esta investigación destacan los que se presentan a continuación.

El concepto central de CPM es el de *artefacto,* entendido como una entidad distintiva en un producto, ya sea un componente, pieza, ensamble o subensamble [57]. A pesar de que la clase Artifact de CPM considera tanto la función, como la forma y el comportamiento, en esta propuesta se pondrá el foco en la forma, es decir, su

definición geométrica. Así pues, en adelante el concepto *artefacto* se referirá a una realidad física con forma, y se considerarán (igual que OAM) dos especializaciones: *pieza* y *ensamble* (clases Part y Assembly en OAM). Una *pieza* se refiere a un artefacto atómico, que no se puede descomponer, mientras que un *ensamble* es un artefacto compuesto por otros artefactos, ya sean piezas u otros ensambles (llamados habitualmente *subensambles*).

Dejando a un lado la estructura de componentes, y profundizando en la definición geométrica de un artefacto, CPM presenta el concepto *feature* como un rasgo característico de forma (*form feature*) [57]. La clase Feature de CPM agrega tanto una forma (clase Form) como una función (clase Function). Sin embargo, en esta investigación no se hará especial referencia a la función, asumiendo que las especificaciones geométricas ya están establecidas atendiendo a los requisitos funcionales, pero en el documento se emplea el término "feature" por su extendido uso en el campo de la ingeniería para referirse a elementos geométricos característicos del artefacto.

Por su parte, OAM está orientado al modelado de ensambles, definiendo dos tipos de artefacto: pieza o ensamble. Además, OAM presenta una especialización del feature de CPM empleada para soportar relaciones de ensamble entre artefactos, la clase AssemblyFeature, un término que se adoptará en esta investigación.

En lo referido a las asociaciones entre entidades, son de especial interés las *asociaciones de features*, que OAM concreta para el caso de las relaciones de ensamble a través de la clase AssemblyFeatureAssociation. Sin embargo, en esta propuesta serán de interés tanto las relaciones entre features de artefactos diferentes que dan soporte a las relaciones de ensamble, como las relaciones entre features de un mismo artefacto que dan soporte a especificaciones dimensionales y geométricas del mismo. Por otra parte, cabe aclarar que, a pesar de que en modelos como OAM se presentan otras clases que son agregaciones de la AssemblyFeatureAssociation (ArtifactAssociation y AssemblyAssociation), éstas no se considerarán para el desarrollo del lenguaje propuesto.

Abordando ahora el modelado de tolerancias, en OAM la *tolerancia* es entendida como la variabilidad admisible en la especificación de una determinada geometría, si bien en [58] también se vincula a las clases AssemblyFeatureAssociation y ArtifactAssociation. Conviene aclarar que, aunque en OAM se presentan múltiples especializaciones de la clase Tolerance (FormTolerance, ProfileTolerance, OrientationTolerance, LocationTolerance, etc.), en esta propuesta se limita el alcance a tratar especificaciones de orientación, posición y tamaño.

Por último, es digno de mención el uso del concepto *representación*, adoptado en OAM para definir la clase AssemblyFeatureAssociationRepresentation. Con esta clase no solo se están modelando asociaciones de ensamble entre features, sino que se considera explícitamente la posibilidad de modelar representaciones de la misma, por ejemplo, definiendo restricciones desde un punto de vista paramétrico o cinemático. Este uso del concepto *representación* se ha considerado especialmente interesante en el desarrollo de la propuesta y será adoptado, si bien con ciertas modificaciones.

B. Modelado TTRS de superficies y sus relaciones

De forma complementaria a las propuestas anteriores, orientadas a mejorar la interoperabilidad en contextos CAx, otras teorías y modelos adoptan una orientación más relacionada con el análisis de la geometría, y que generalmente se adoptan en las etapas tempranas de diseño de producto para analizar el principio de solución a través del modelado de superficies activas y sus relaciones, entre otros aspectos. Entre estos modelos, existen propuestas orientadas al análisis cualitativo, como es el caso de Contact and Channel Approach (C&C2-A) [59] y otros trabajos basados en este enfoque, como [127] o [128]. Sin embargo, en el contexto de esta investigación, son de especial interés las propuestas orientadas al análisis cuantitativo, estableciendo las bases para soportar el modelado matemático de las superficies y las relaciones entre ellas. Con esta orientación, la principal referencia considerada en esta revisión del estado del arte ha sido el modelo de Technological and Topological Related Surfaces (TTRS) [60], que establece unos conceptos básicos para el modelado matemático de geometría en función de la clase de invarianza de las superficies y que, como se comentará más adelante, también es muy interesante para representar zonas de tolerancia (TZ) de acuerdo con los estándares Geometric Dimensioning and Tolerancing (GD&T) [54].

En un espacio euclídeo, las superficies pueden tener hasta seis grados de libertad: tres traslaciones y tres rotaciones. En este sentido, las superficies pueden clasificarse en siete *clases de invarianza* en función de cuáles de estos grados de libertad son variantes (aquellos en los que un cambio modifica la posición de la superficie en el espacio) y cuáles son invariantes (aquellos en los que un cambio mantiene la superficie invariante). Así, las siete clases de invarianza son: esférica, cilíndrica, plana, helicoidal, de revolución, prismática y compleja. En esta propuesta, como en muchos de los trabajos consultados, no se tratará el caso de superficies helicoidales, centrándose en las seis clases restantes. Según el modelo TTRS, cada clase de invarianza se caracteriza por un conjunto mínimo de elementos geométricos de referencia, llamados *MGREs* (Minimal Reference Geometric Element), necesarios para definir la posición y orientación de la superficie. Estos conjuntos están formados a su vez por uno, dos o hasta tres elementos, conocidos como *RGEs* (Reduced Geometric Element), que pueden ser de tres tipos: puntos, líneas o planos. La combinación de estos elementos, relacionados entre sí mediante restricciones geométricas, permite definir los MGREs característicos de las siete clases de invarianza.

Sin embargo, el modelo TTRS pone el foco de atención en la relación entre superficies, con lo que se centra en caracterizar las posibles relaciones entre superficies de las diferentes clases de invarianza. Así, un *TTRS* es el conjunto formado por dos superficies (o una superficie y un TTRS, o dos TTRSs) pertenecientes al mismo sólido (aspecto topológico) y localizados en la misma cadena cinemática en un determinado mecanismo (aspecto tecnológico). Algunos autores han trasladado también esta definición a las relaciones entre superficies de sólidos diferentes, dando lugar al concepto de *pseudoTTRS*, un enfoque muy interesante desde la perspectiva del modelado de ensambles.

La forma en que TTRS asocia dos superficies es mediante la definición de restricciones entre sus MGREs, o de forma más detallada, entre los RGEs que conforman sus respectivos MGREs. El modelo TTRS caracteriza todas las posibles combinaciones, dando lugar a 44 casos en función de las clases de invarianza involucradas y al tipo de relación definida entre ellas. Cada uno de estos casos define un todo (conjunto de superficies y relaciones entre ellas) que se caracteriza por su propia clase de invarianza y, por tanto, por su propio MGRE. Sin embargo, esta cuestión no es explotada en el estado actual de la propuesta, que se centra en la implementación de las relaciones TTRS como expresiones matemáticas (fundamentalmente productos vectoriales y escalares), soportando tanto especificaciones de piezas como relaciones de ensamblaje.

		MGRE						
		Spherical	Planar	Cylindrical	Helical	Revolution	Prismatic	Complete
MGRE	Spherical	1-coincidence 2-otherwise	3-always	4-coincidence 5-otherwise	14	20-coincidence 21-otherwise	29-always	38-always
	Planar		6-parallel 7-otherwise	8-perpendicular 9-parallel 10-otherwise	15	22-perpendicular 23-otherwise	30-parallel 31-otherwise	39-always
	Cylindrical			11-coincidence 12-parallel 13-otherwise	16 17	24-coincidence 25-otherwise	32-parallel 33-otherwise	40-always
	Helical				18 19	26	34	41
	Revolution					27-coincidence 28-otherwise	35-always	42-always
	Prismatic						36-parallel 37-otherwise	43-always
	Complete							44-always

Figura 5.11. Los 44 tipos de asociación entre MGRE. Elaboración propia basada en [31]

Estos 44 casos se muestran en la Figura 5.11, donde se utiliza un código de colores para diferenciar tres conjuntos particulares de asociaciones de MGRE: asociación entre MGREs simples, es decir, aquellos compuestos por un único RGE (verde); asociaciones con al menos un MGRE complejo, es decir, aquellos compuestos por más de un RGE (amarillo); y asociaciones donde participa un MGRE helicoidal (gris), identificado específicamente porque no son consideradas en este trabajo. Por lo tanto, el alcance del trabajo abordará los 35 casos restantes. Cabe mencionar que la diferenciación entre asociaciones verdes y amarillas es importante porque los casos con MGRE complejos están soportados por una combinación de restricciones características de los 13 casos simples.

Desde la perspectiva de esta propuesta, el análisis de superficies por pares y las restricciones entre ellas es una vista muy interesante para la representación de especificaciones y tolerancias geométricas, como se ha explorado en [61]. Así, se considera que el modelo TTRS es adecuado y muy interesante tanto para representar superficies y relaciones entre ellas como para representar zonas de tolerancia (TZ) de acuerdo con los estándares GD&T, especialmente en casos básicos como las zonas de tolerancia planas, cilíndricas o esféricas. Estas TZ se pueden definir por su correspondiente MGRE (según su clase de invariancia), y su tamaño (básicamente su ancho o diámetro). Según los estándares de GD&T, se pueden construir TZs más complejas como resultado de la intersección o unión de TZ simples.

C. Modelado matemático de superficies desviadas y sus relaciones

En cuanto al modelado de geometrías con defectos, es necesario aclarar que el modelo TTRS no hace referencia a la distinción entre geometrías nominales y desviadas, un aspecto fundamental en esta investigación para cuantificar los errores en la ubicación/orientación de la pieza en el ensamble. Una solución es modelar la geometría desviada mediante la definición de sus desviaciones de posición y/o de orientación (en función de la clase de invarianza) respecto de la nominal, para lo cual es necesario combinar los conceptos TTRS con ciertas construcciones matemáticas que permitan definir dichas desviaciones. En este sentido, para el desarrollo de este trabajo se ha explorado el uso del Small Displacement Torsor (SDT) [119] para soportar la definición de geometrías desviadas. Los valores del SDT definen las desviaciones máximas aplicables a cada grado de libertad variante, calculadas de acuerdo a las geometrías implicadas y la zona de tolerancia (considerando su tipo y tamaño). Más allá de seleccionar una construcción matemática adecuada, el modelado de la geometría combinando conceptos TTRS y aspectos de la implementación matemática requiere tener en cuenta una serie de consideraciones que se comentan a continuación.

Como se ha comentado anteriormente, el modelo TTRS define a cada superficie mediante una combinación de RGEs, que pueden ser un punto, una línea o un plano. Estos elementos geométricos básicos deben estar definidos por una expresión matemática apropiada, en este caso una combinación de coordenadas y vectores expresados en un sistema local cartesiano, específico para cada pieza considerada. Esta descripción geométrica local permite la especificación independiente de cada pieza, pero no es compatible con la definición de relaciones de ensamblaje. Para ello, todas las geometrías involucradas en el conjunto analizado deben expresarse con respecto a un sistema global común, lo que permite la definición de expresiones matemáticas que soportan pseudo-TTRS. Por esta razón, la definición local de las geometrías debe transformarse a una definición global, cambiando el sistema de referencia considerado en cada caso. Esta transformación puede estar respaldada por una matriz de transformación que es característica de cada pieza. Una vez expresadas todas las geometrías de cada pieza con respecto a un sistema común, todas las restricciones TTRS se implementan como expresiones matemáticas entre ellas (fundamentalmente productos vectoriales y escalares), soportando tanto especificaciones de piezas como relaciones de ensamblaje.

En lo referido al modelado de geometrías nominales y desviadas, se adopta el siguiente procesamiento matemático de la geometría nominal de cada pieza. En primer lugar, se define la geometría nominal respecto del sistema de referencia local, trasladando la información que habitualmente se expresa en la especificación de cada pieza por separado. Esta tarea incluye también el modelado de las zonas de tolerancia. Seguidamente, se calculan las geometrías desviadas respecto de las nominales aplicando un vector SDT compatible con los límites impuestos por la zona de tolerancia. Estas geometrías desviadas también se expresan con respecto a los sistemas de coordenadas locales. A continuación, se realiza la transformación de coordenadas de geometrías desviadas a un sistema de coordenadas global, aplicando una matriz de transformación paramétrica, cuyos valores deben calcularse a partir de las relaciones de ensamblaje que restringen la posición/orientación de la pieza.

5.3.2. Descripción del metamodelo para el análisis de tolerancias

Este apartado describe el metamodelo de un lenguaje diseñado para dar soporte a la especificación geométrica de productos y al análisis de tolerancias basado en TTRS, asegurando la consistencia entre la especificación y el análisis. Para ello, el metamodelo se presenta utilizando varios diagramas conceptuales (diagramas UML) para representar la sintaxis abstracta de este lenguaje. El apartado se estructura en tres partes en las que se abordan las siguientes cuestiones:

- Modelado de artefactos, definiendo la estructura jerárquica de composición de piezas y ensambles.
- Especificación de la geometría de los artefactos basada en la definición de features.
- Ampliación del modelo anterior incluyendo elementos orientados al análisis, específicamente una representación del artefacto basada en el modelo TTRS que debe mantener la consistencia con la especificación basada en features.

A. Modelado de la estructura básica de un ensamble

Una de las metaclases principales del metamodelo es Artifact, que corresponde al concepto *artefacto* descrito anteriormente. Se trata de una metaclase abstracta que tiene dos especializaciones: Part y Assembly. Tal y como se representa en la Figura 5.12, un Assembly se compone de al menos dos usos de la metaclase Artifact, ya sean Parts u otros Assemblies.

Por otra parte, un Artifact puede contener tanto Features como representaciones TTRS de dichos features, soportadas por la metaclase F_TTRS_repr. Aunque estas dos metaclases (Feature y F_TTRS_repr) serán descritas en mayor detalle en los siguientes apartados, conviene mencionar que entre ellas existe una asociación llamada "representación", de manera que una instancia de Feature puede tener (o no) su representación TTRS, pero cualquier instancia de F_TTRS_repr tiene que estar necesariamente asociada a una instancia de Feature.

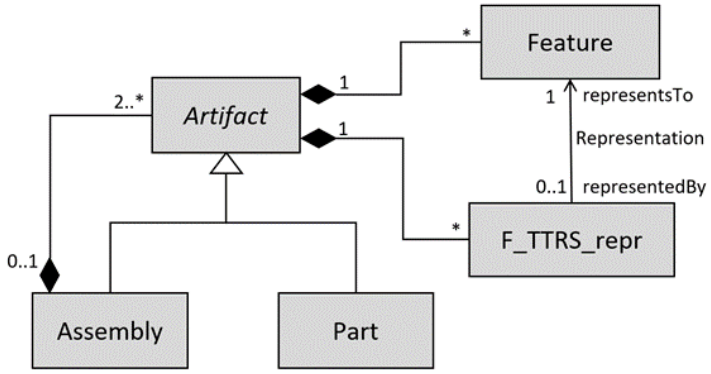


Figura 5.12. Sintaxis abstracta de artefactos.

B. Modelado de artefactos basado en features

La especificación geométrica de los artefactos se basa en el concepto de feature descrito anteriormente, así como las relaciones entre ellos. Esta propuesta adopta una conceptualización unificada de asociaciones de features, tanto para asociaciones entre features pertenecientes a una misma pieza (asociaciones intra-pieza orientadas a especificar la pieza) como para asociaciones entre features de diferentes piezas (asociaciones inter-pieza orientadas a definir relaciones de ensamble). Como se muestra en la Figura 5.13, un Artifact se compone de Features que definen elementos geométricos característicos de los artefactos, aunque este tipo de características se definen habitualmente en los artefactos de tipo pieza (Part). Además, algunos de estos features se ofrecen al entorno para participar en relaciones de ensamble, dando lugar a la metaclase AssemblyFeatures, un concepto que se aplicará en la definición de puertos de tipo Feature.

Por otra parte, se definen diversas metaclases orientadas a dar soporte a la especificación de tolerancias geométricas compatibles con el estándar GD&T. Así, la metaclase GT_Specification se emplea para establecer tolerancias geométricas entre features de una misma pieza, identificando los features que actúan como datum y target, y definiendo el tamaño de la zona de tolerancia. Esta metaclase se emplea tanto para las relaciones binarias (una referencia y un target) como para especificaciones más complejas donde se requiere más de una referencia o datum. Para ello, una GT_Specification tiene Features definidos como puertos, que se representan con la metaclase FeatureUsage. Uno de los puertos FeatureUsage debe tener el rol de target, mientras que se definen uno, dos o hasta tres puertos FeatureUsage con el rol de datum. Además, cuenta con una propiedad de tipo Real llamada ToleranceSize, que representa el tamaño de la zona de tolerancia.

La especificación de condiciones funcionales también representa una relación entre features (de la misma o de diferentes piezas de un ensamble) que se pretende analizar. Este concepto está soportado por la metaclase FC_Specification, que tiene una estructura muy similar a la metaclase GT_Specification anteriormente descrita. Esto es debido a que la FC_Specification puede considerarse como una GT_Specification cuyo valor no es especificado, sino que depende del resto de especificaciones de la pieza.

Por último, cabe indicar que los usos de features como target o datum de una tolerancia geométrica o condición funcional (instancias de la metaclase FeatureUsage) cuentan con dos propiedades: una de tipo Role para definir precisamente el rol que adoptan en la especificación (datum o target); y otra de tipo Order para definir el orden en caso de ser un datum (la especialización None se define como alternativa cuanto el rol es target).

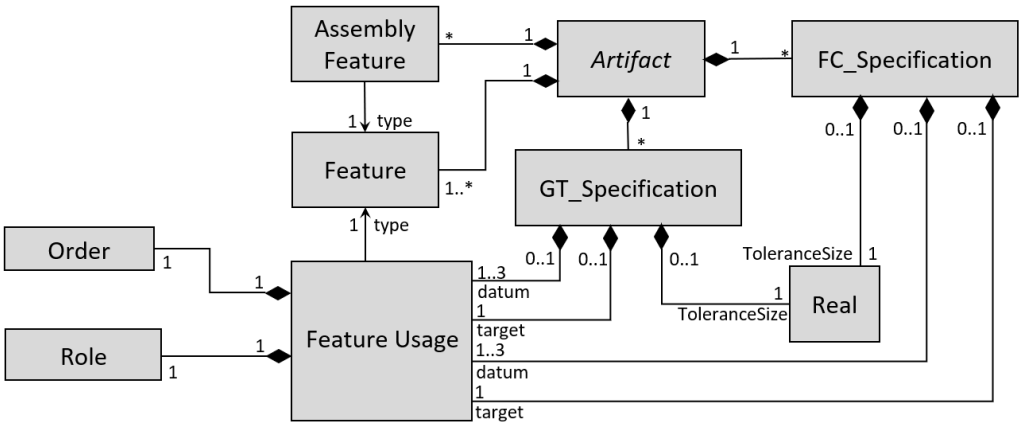


Figura 5.13. Sintaxis abstracta de especificación de artefactos basada en features.

C. Modelado de artefactos basado en TTRS

El lenguaje propuesto define una serie de metaclases para dar soporte a la representación de artefactos orientadas al análisis, incorporando conceptos propios del modelo TTRS para la definición matemática de la posición y orientación de las geometrías según sus clases de invarianza. Así pues, tal y como se propone en el modelo TTRS y se representa en la Figura 5.14, cada clase de invariancia se caracteriza por un MGRE particular, y un MGRE se compone de uno, dos o tres RGE y restricciones entre ellos, llamadas en este metamodelo RGE_Constraints.

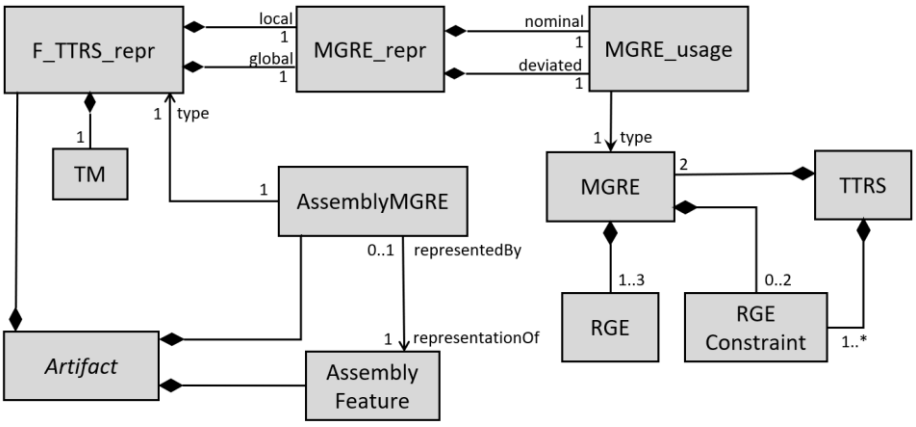


Figura 5.14. Sintaxis abstracta de representación basada en TTRS.

Por su parte, la metaclase F_TTRS_repr soporta la representación basada en TTRS de un Feature. Esta metaclase incluye dos propiedades de tipo MGRE_repr: “local” y “global”. La metaclase MGRE_repr posee dos propiedades MGRE_usage (de tipo

MGRE) que soportan la definición nominal y desviada de una geometría. Por tanto, el uso “local” contiene la definición matemática del feature (nominal y desviado) con respecto al sistema de coordenadas local del artefacto, mientras que el uso “global” es la definición del mismo feature pero respecto a un sistema de coordenadas global, común para todas las piezas que participan en el ensamble más general. Por este motivo, ambas definiciones están relacionadas, y “global” se calcula a partir de "local" a través de una transformación de coordenadas (TM). En esta propuesta, la transformación se rige por una matriz de transformación homogénea característica de cada Part que se define como una propiedad de tipo TransformationMatrix. A su vez, cada F_TTRS_repr debe contar con un puerto de tipo TM, que se conecta con la matriz de transformación propia de la Part a la que pertenece el F_TTRS_repr, para soportar las transformaciones de los valores de coordenadas locales a globales y viceversa, haciendo uso de la matriz de transformación adecuada.

Para caracterizar las 35 combinaciones de MGRE consideradas en este trabajo, también se incluye el concepto TTRS como una entidad con dos propiedades MGRE y restricciones matemáticas entre sus RGE (RGE_constraint), mientras que la metaclase AssemblyMGRE identifica el uso de MGRE en relaciones de ensamble, es decir, la representación basada en TTRS de los AssemblyFeatures descritos anteriormente. Los AssemblyMGRE siempre están asociados a F_TTRS_repr que definen su tipo, y las conexiones entre estos elementos permiten definir las relaciones de ensamble entre diferentes artefactos.

La representación basada en TTRS también se emplea para definir matemáticamente las tolerancias geométricas y zonas de tolerancia. La metaclase GT_TTRS_definition representa la definición matemática de tolerancias geométricas basada en TTRS. Mientras que la especificación (GT_Specification) se limitaba a identificar features que actúan como datum y target, y definir el tamaño de la zona de tolerancia, la metaclase GT_TTRS_definition soporta la definición matemática, incluyendo todas las ecuaciones que permiten construir la zona de tolerancia respecto a las geometrías datum y definir una geometría target dentro de dichos límites. Para ello, se consideran las siguientes entidades en el metamodelo:

- TZ_TTRS_definition que permite definir la zona de tolerancia en términos TTRS;
- TTRS_repr_usage son aquellas representaciones que se emplean como datum o como target de la tolerancia geométrica;
- restricciones matemáticas (usos de bloques TTRS) para construir la TZ_TTRS_definition con respecto a las geometrías datum (TTRS_repr_usage con el rol “datum”);
- SpecifValidation es una entidad con restricciones que permiten validar la corrección de la tolerancia geométrica definida, comprobando que sea coherente con la definición nominal del artefacto a través de la combinación de algunos de los 13 casos simples de asociaciones MGRE.

- TargetConstruction, es una entidad con restricciones que permiten construir una geometría target desviada con respecto a la TZ_TTRS_definition, aplicado un torsor de pequeños desplazamientos (SDT);

- targetDeviation es un uso de un vector (DeviationVector) cuyos componentes especifican el valor de desviación en cada grado de libertad variante.

La Figura 5.15 presenta estos conceptos y las relaciones existentes entre ellos.

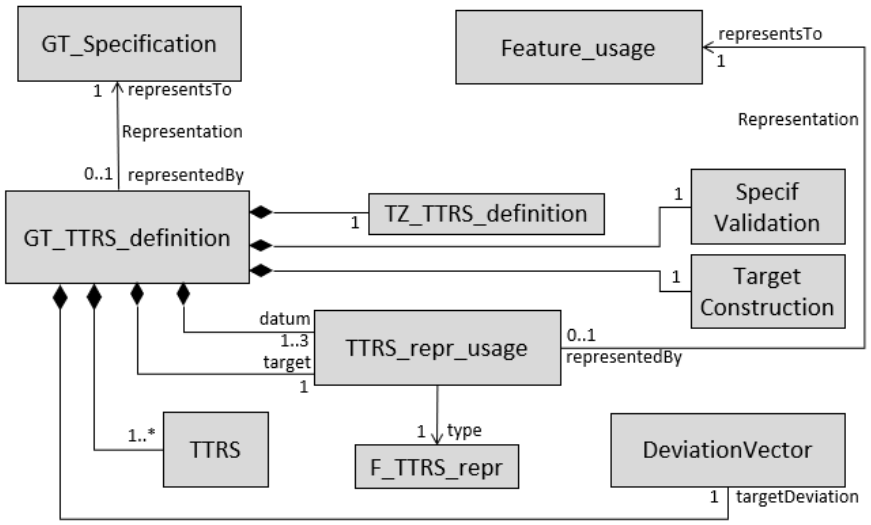


Figura 5.15. Sintaxis abstracta de definición de tolerancias geométricas basada en TTRS.

5.3.3. Implementación del metamodelo: perfil SysML4TA

Atendiendo al metamodelo descrito en el apartado anterior, este apartado presenta la implementación del lenguaje como un perfil SysML llamado SysML4TA (SysML for Tolerance Analysis). A continuación, se detallan los estereotipos considerados, incluyendo su definición y la descripción en lenguaje natural de las reglas incluidas en cada estereotipo.

Como se ha comentado en la descripción del metamodelo, este lenguaje involucra conceptos de cuatro ámbitos: modelado de artefactos, especificación basada en Features, representación basada en TTRS y análisis de tolerancias. Por este motivo, el perfil está estructurado en cuatro paquetes (Figura 5.16):

- ArtMPP → Artifact Modeling Profile Package

- FMPP → Feature-based Modeling Profile Package

- TTRS_MPP → TTRS-based Modeling Profile Package

- TAPP → Tolerance Analysis Profile Package

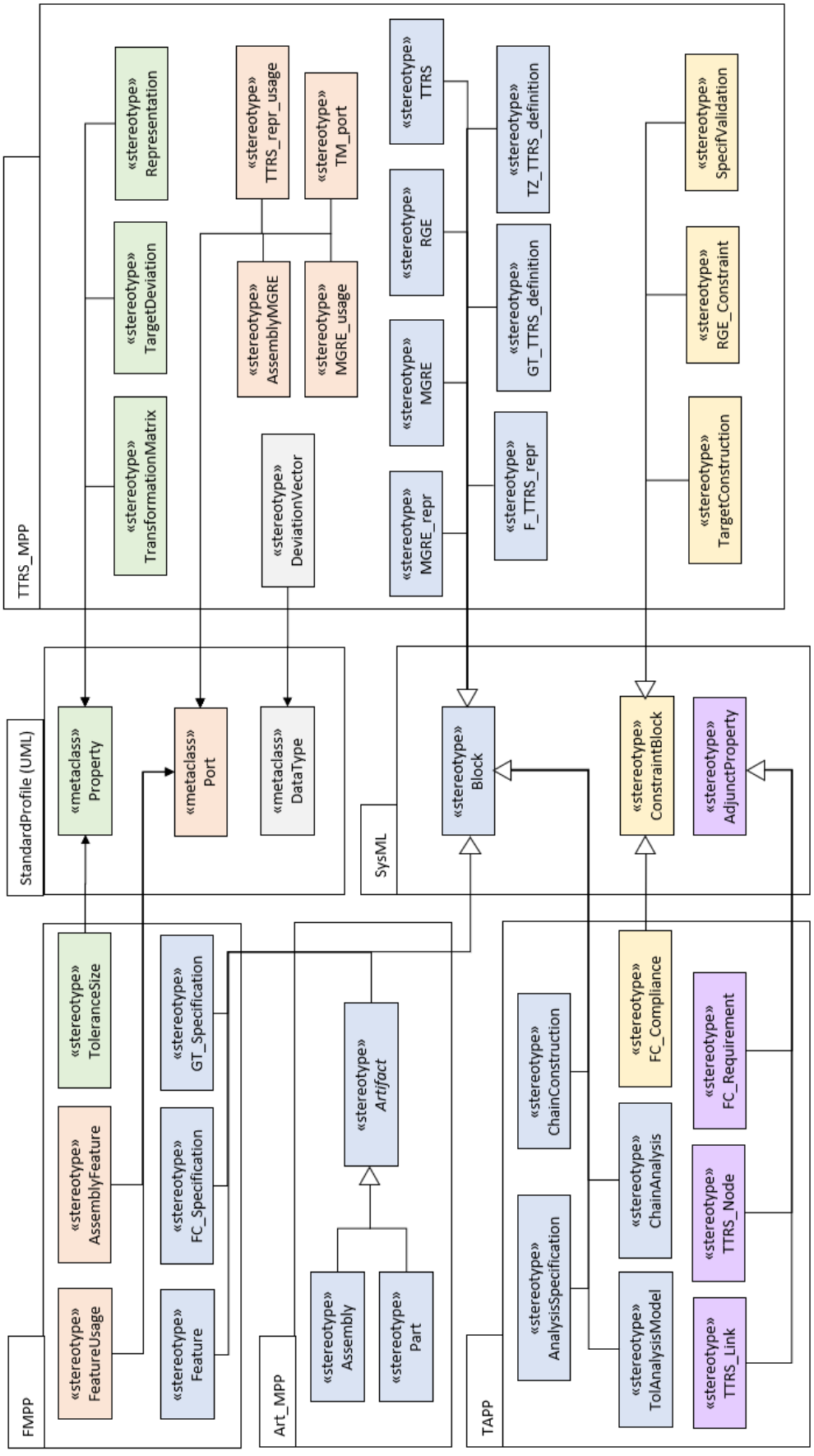


Figura 5.16. Diagrama de paquetes del perfil SysML4TA.

A continuación, se dedica un subapartado a cada uno de estos paquetes, en los que se enumeran en formato de tabla los diversos estereotipos definidos, incluyendo una breve definición de los mismos. La especificación completa del perfil, incluyendo las reglas OCL definidas en cada estereotipo, se recoge en el Anexo B.

A. Artifact Modeling Profile Package (Art_MPP)

Tabla 5.3. Estereotipos del paquete SysML4TA.

Estereotipo	Descripción
«Artifact»	Estereotipo abstracto que especializa al bloque para definir un artefacto, una entidad distintiva en un sistema físico.
«Part»	Bloque que define una pieza, un artefacto atómico que no puede ser descompuesto en otros artefactos.
«Assembly»	Bloque que define un ensamble, un artefacto complejo que está compuesto por al menos dos artefactos con relaciones de ensamble entre sí.

B. Feature-based Modeling Profile Package (FMPP)

Tabla 5.4. Estereotipos del paquete FMPP.

Estereotipo	Descripción
«Feature»	Bloque en el que se especifica una porción de la forma de un artefacto.
«FC_Specification»	Bloque en el que se especifica una condición funcional, definiendo el límite de la variabilidad aceptable entre un feature objetivo y una o más referencias. Para ello, debe contar con una propiedad que defina el tamaño de la zona de tolerancia («ToleranceSize» Property) y entre dos y cuatro «Feature usage» Ports para conectar el bloque con las geometrías referencia y datum.
«GT_Specification»	Bloque en el que se especifica una especificación geométrica, definiendo el límite de la variabilidad aceptable entre un feature objetivo y una o más referencias. A diferencia de las condiciones funcionales, la especificación establece un límite que siempre se debe cumplir, mientras que el valor de las posibles desviaciones en una condición funcional es un valor calculado mediante un determinado análisis de tolerancias. Para ello, debe contar con una propiedad que defina el tamaño de la zona de tolerancia («ToleranceSize» Property) y entre dos y cuatro «Feature usage» Ports para conectar el bloque con las geometrías referencia y datum.
«ToleranceSize»	Propiedad valor de tipo Real en la que se especifica el tamaño de una zona de tolerancia, ya sea de una especificación o de una condición funcional.
«FeatureUsage»	Puerto que representa un feature participando de una especificación geométrica o de una condición funcional, ya sea como elemento de referencia o como target. Por tanto, el tipo de estos puertos siempre será un «Feature» Block. Para una completa definición, este estereotipo cuenta con atributos que permiten diferenciar su rol (referencia o target), y el orden en el caso de ser una de varias referencias.
«AssemblyFeature»	Puerto que representa un feature ofrecido por un artefacto a su entorno para participar en relaciones de ensamble con otros artefactos. Por tanto, el tipo de

	estos puertos siempre será un «Feature» Block, y debe ser un puerto poseído por un «Artifact» Block.

C. TTRS-based Modeling Profile Package (TTRS_MPP)

Tabla 5.5. Estereotipos del paquete TTRS_MPP.

Estereotipo	**Descripción**
«F_TTRS_Repr»	Bloque que representa un feature en base a TTRS, incluyendo la definición de dos propiedades tipeadas por «MGRE_repr» Blocks que representen su posición y orientación respecto a los sistemas de referencia local y global, respectivamente. Además, entre ambas propiedades tipeadas debe definirse una ConstraintBlock que dé soporte a la transformación entre los sistemas de referencia, aplicando una matriz de transformación característica de la pieza poseedora («Part» Block) que se transmite al «F_TTRS_Repr» Block a través de un «TM_Port» Port.
«MGRE»	Elemento geométrico mínimo de referencia (concepto del modelo TTRS) para una superficie de acuerdo a su clase de invarianza, compuesto por N propiedades de tipo «RGE» Block y N-1 «RGE_Constraint» ConstraintBlocks, siendo N un valor entre 1 y 3.
«MGRE_repr»	Bloque que representa un feature en base a TTRS, incluyendo la definición de dos «MGRE_usage» behavior Ports que representan la definición nominal y desviada.
«MGRE_usage»	Puerto de comportamiento (behavior Port) que define un MGRE, diferenciando mediante el atributo booleano “isDeviated” si se trata de una definición nominal o desviada. Estos puertos siempre están definidos por un «MGRE» Block.
«RGE»	Bloque que representa un elemento geométrico reducido (concepto del modelo TTRS), empleado para la definición de un MGRE, de modo que cualquier uso (propiedad tipeada por un «RGE» Block) debe pertenecer a un «MGRE» Block.
«TTRS»	Bloque que representa la relación entre un par de MGRE definiendo restricciones de posición/orientación entre ellos. Por tanto, debe poseer dos puertos tipeados por «MGRE» Blocks y al menos una «RGE_Constraint» ConstraintBlock
«RGE_Constraint»	Bloque restricción (ConstraintBlock) entre dos RGEs, por lo al menos dos de sus parámetros deben estar tipeados por «RGE» Blocks.
«Representation»	Propiedad que representa otra propiedad del modelo desde un punto de vista diferente, al cual apunta la propiedad del estereotipo “representsTo”.
«AssemblyMGRE»	Puerto que representa en base al modelo TTRS un «AssemblyFeature», es decir, una geometría que un artefacto ofrece a su entorno para definir relaciones de ensamble. Este tipo de puerto siempre estará tipeado por un «F_TTRS_Repr» Block.
«Transformation_Matrix»	Propiedad en la que se define la matriz homogénea de transformación característica de una pieza y empleada para cambiar del sistema local de

	la pieza al global del ensamble. Esta propiedad debe ser única (no pueden definirse dos en un mismo bloque) y su poseedor siempre será un «Part» Block.
«TM_Port»	Puerto de comportamiento (behavior Port) de un «F_TTRE_repr» Block que se emplea para conectarse con la «Transformation_Matrix» Property del «Part» Block al que pertenece dicha representación TTRS.
«GT_TTRS_definition»	Bloque que define la representación matemática basada en TTRS de una tolerancia geométrica. Cuenta entre 2 y 4 «TTRS_Repr_usage» Ports para definir las geometrías target y referencias. Debe tener una propiedad tipeada por un «TZ_TTRS_definition» Block para la construcción de la zona de tolerancia, y una o más propiedades tipeadas por «TTRS» Blocks para definir los mecanismos de construcción de la zona de tolerancia a partir de las referencias. Adicionalmente, cuenta con dos propiedades tipeadas por «SpecifValidation» y «TargetConstruction» ConstraintBlocks.
«SpecifValidation»	Bloque restricción (ConstraintBlock) encargado de validar la consistencia de una tolerancia geométrica y devolviendo un booleano como resultado de la validación.
«TargetConstruction»	Bloque restricción (ConstraintBlock) encargado de definir un MGRE desviado para la geometría target aplicando un vector de desviaciones, dentro de los límites impuestos por la zona de tolerancia establecida.
«TZ_TTRS_definition»	Bloque que define el espacio geométrico utilizado para controlar la posición/orientación de una geometría target. Se caracteriza por su propio MGRE y un tamaño de la zona de tolerancia, que se puede definir mediante una AdjuntProperty que apunte a la «ToleranceSize» Property de un «GT_Specification» Block.
«TTRS_Repr_usage»	Puerto que se emplea para definir las geometrías target y referencia en una tolerancia geométrica, identificando su rol y el orden de las referencias de forma similar al «Feature_usage».
«TargetDeviation»	Propiedad de un «GT_TTRS_definition» Blockque expresa las desviaciones en los grados variantes del MGRE target. Esta propiedad siempre debe estar tipeada por un «DeviationVector» DataType.
«DeviationVector»	DataType definido para expresar el vector de pequeños desplazamientos de un MGRE desviado respecto al MGRE nominal.

5.4. Lenguaje para la simulación de desviaciones geométricas en procesos de fabricación multietapa

La presente sección se centra en el desarrollo de un lenguaje que dé soporte al modelado en SysML de sistemas para la simulación de la productividad en sistemas de fabricación multietapa, incluyendo la simulación de la calidad geométrica de los productos fabricados. Para ello, se integran conceptos del lenguaje SysML4MSSim (sección 5.2) y del lenguaje SysML4TA (sección 5.3), incorporando además otros

conceptos adicionales vinculados principalmente a la definición y flujo de las desviaciones geométricas del producto a lo largo del sistema simulado.

5.4.1. Introducción

Como se ha comentado en la sección 5.3. los conceptos manejados en el modelo TTRS permiten dar soporte al modelado geométrico de artefactos, lo cual no solo es aplicable al producto, sino que también a artefactos más complejos como el ensamble de proceso, que describe el amarre de un determinado producto y los utillajes empleados. Además, construcciones matemáticas como el SDT permiten abordar de forma coherente no sólo el análisis de las geometrías nominales, sino también de las desviadas, dando soporte al análisis de la calidad en un planteamiento unietapa. Sin embargo, para abordar el análisis multietapa es necesario construir un sistema en base a los conceptos comentados en la sección 5.1 y adaptar la definición de las unidades de flujo consideradas, incluyendo los datos propios de las desviaciones de los productos simulados en base a TTRS.

Con todo ello, el lenguaje presentado en esta sección cuenta con conceptos que especializan los manejados en el lenguaje SysML4MSSim descrito en la sección 5.2, incluyendo construcciones propias del lenguaje SysML4TA descrito en la sección 5.3. De igual manera que en los casos anteriores, el lenguaje se describe inicialmente mediante un metamodelo, y posteriormente se implementa como perfil SysML. Cabe indicar que este nuevo perfil debe importar los dos anteriores, de manera que tenga acceso a los estereotipos previamente definidos y se puedan definir especializaciones de los mismos, así como hacer referencia a ellos en las nuevas reglas OCL que soportan la semántica estática del lenguaje.

5.4.2. Descripción del metamodelo para la simulación de desviaciones geométricas en procesos multietapa

En el presente apartado se presenta un modelo conceptual que describe el metamodelo del lenguaje propuesto, abordando tanto el modelado geométrico de los ensambles de proceso como el flujo de las desviaciones geométricas del producto a lo largo del sistema de fabricación simulado. Para ello, además de la descripción textual de los conceptos y relaciones consideradas, se incluyen algunos diagramas conceptuales para representar la sintaxis abstracta de este lenguaje. Dado que algunos de los conceptos manejados en esta descripción pertenecen a lenguajes previamente presentados (secciones 5.2 y 5.3), en estos diagramas se identifica la referencia a dichos conceptos previos empleando bordes discontinuos, enfatizando que no pertenecen estrictamente al lenguaje descrito en esta sección.

En primer lugar, es conveniente indicar que la construcción fundamental para abordar el análisis de las desviaciones geométricas en cada una de las etapas de fabricación consideradas es el ensamble de proceso, es decir, el artefacto compuesto por el producto y la máquina configurada. Cada recurso transformador que se simula debe contar con un ensamble de proceso que se emplea para calcular las desviaciones del producto resultante en base a las características del subproducto entrante y las propias

características de los recursos empleados (utillajes, herramientas, etc.). Esta construcción se emplea para analizar la calidad en cada etapa, y posteriormente los datos de las desviaciones fluirán y serán tratados para abordar el análisis multietapa. Por tanto, es necesario modelar tanto el artefacto producto como los artefactos propios de la máquina configurada (utillajes, herramientas, máquina, etc.) así como las relaciones entre ellos. Esas cuestiones se tratan en mayor detalle a continuación.

Por una parte, se aborda el modelado geométrico de los productos. Para ello se deben extender los conceptos Product_sim y ProductSpecif_data del lenguaje SysML4MSSim (sección 5.3), con el fin de incluir las construcciones y datos necesarios para la definición de su geometría, tanto nominal como desviada. Así, el concepto ProductSpecif_data se ha especializado (DevProductSpecif_data) para incluir en la especificación del producto el modelado de artefactos (ProductArtifact) que describen físicamente el producto en alguno de sus estados de fabricación. Conviene indicar que ProductArtifact es una especialización del concepto Artifact presentado en el lenguaje SysML4TA (sección 5.3) que necesariamente contiene la representación basada en TTRS de la geometría nominal y las tolerancias consideradas según las especificaciones de diseño del tipo de producto modelado.

Por otra parte, por lo que se refiere a la simulación de las unidades de producto simuladas (Product_sim), se deben incorporar los datos relativos a las desviaciones en las características clave, no todo el artefacto. Para ello, se propone la especialización DevProduct_sim, que incluye el ProductDeviations, una especialización de DeviationVector que almacena los datos sobre las desviaciones de cada una de las geometrías de dicha unidad de producto. Cabe resaltar que con este enfoque se evita que cada DevProduct_sim contenga la definición matemática del artefacto completo simulado, sino que se limita a almacenar las variables mínimas que caracterizan la geometría desviada. Estos datos fluirán por el sistema de simulación, siendo empleadas en los cálculos y transformaciones de cada etapa y evolucionando para representar los cambios introducidos en la geometría del producto procesado. La Figura 5.17 representa gráficamente algunos de estos conceptos.

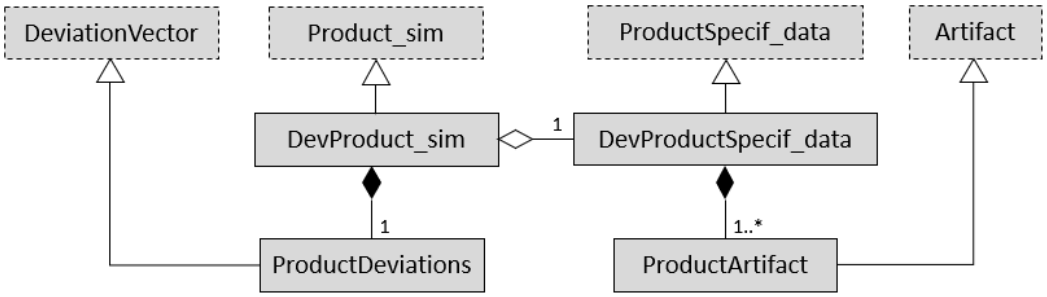


Figura 5.17. Sintaxis abstracta de productos emulados incorporando desviaciones geométricas.

De forma similar, los elementos para la simulación de recursos de fabricación, y en particular de los transformadores (TransformResource_sim), se especializan en DevTransformResource_sim para incluir la definición geométrica de los recursos. Al igual que en el caso del producto, la definición de la geometría nominal se modela a nivel de especificación (DevManufResourceSpecif_data) mediante la construcción de un artefacto (ConfigMachineArtifact), aunque esto es únicamente obligatorio para los

recursos transformadores atómicos que influyen en las características geométricas del producto. El término ConfigMachineArtifact hace referencia al artefacto de la máquina configurada, es decir, un artefacto compuesto tanto por la propia máquina como por herramientas y utillajes necesarios para una determinada subfase, el nivel atómico en el que se analiza el proceso. Por tanto, un recurso transformador puede tener más de una configuración y, por tanto, más de un ConfigMachineArtifact que la describa. Por su parte, las desviaciones particulares de cada uno de los recursos se definirán en una especialización de DeviationVector (ResourceDeviations). La Figura 5.18 muestra estos conceptos y relaciones.

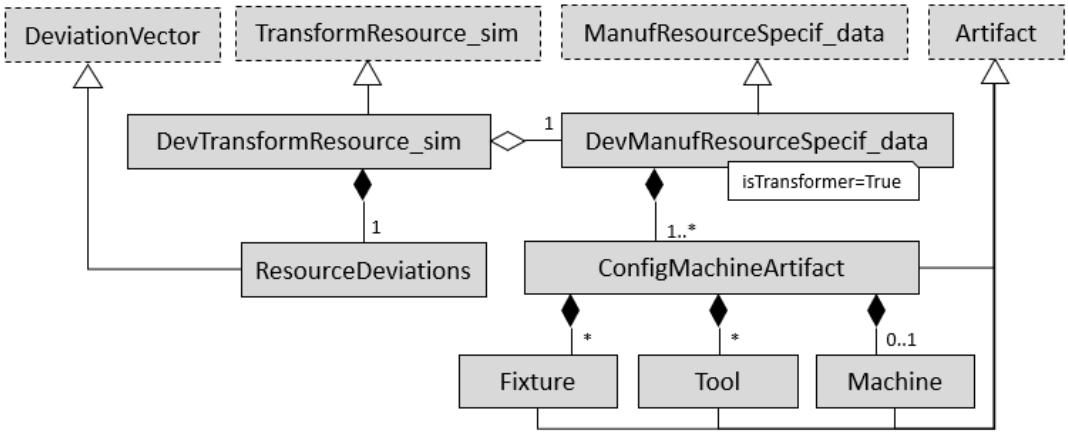


Figura 5.18. Sintaxis abstracta sobre la simulación de recursos con desviaciones geométricas.

Definida ya la geometría tanto del producto como de los recursos, es momento de abordar el modelado del ensamble de proceso, es decir, el artefacto compuesto por el producto y la máquina configurada. Así pues, tal y como se muestra en la Figura 5.19, cada una de las estaciones de trabajo (DesTransformResource_sim atómicas) consideradas debe contar con al menos un ProcessAssembly que relacione una configuración de máquina (ConfigMachineArtifact) con el producto (ProductArtifact) correspondiente al proceso de fabricación analizado en cada caso. En este ProcessAssembly se establecen las relaciones de ensamble entre producto y recurso que determinan las expresiones matemáticas necesarias para obtener las características de calidad del producto resultante.

Cada recurso transformador emulado incorpora un ProcessAssembly en el que se establecen las relaciones matemáticas entre el producto y la máquina configurada, y que se emplea para obtener las desviaciones del producto resultante en dicha etapa del proceso. Estas construcciones resuelven el problema unietapa, pero las simulaciones propuestas deben dar soporte al análisis de sistemas multietapa, en los que las desviaciones no solo se transmiten, sino que van acumulando el efecto de los diversos procesos de fabricación. Considerando los principios del espacio de estados, en el cálculo de las desviaciones del producto resultante de una determinada etapa intervienen tanto las desviaciones del subproducto entrante como las desviaciones introducidas como resultado de la operación de fabricación, en las que generalmente influyen desviaciones del propio recurso (errores en herramientas, útiles, etc.). Por este motivo, cada estación de trabajo simulada debe contener datos relativos a las desviaciones de los recursos empleados en el procesado, así como recibir a través de las unidades de flujo datos sobre las desviaciones del producto entrante.

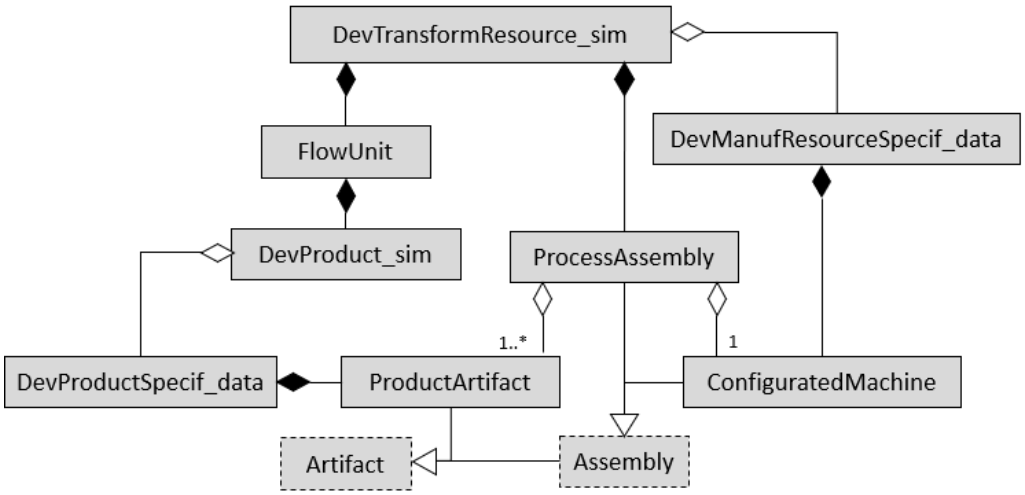


Figura 5.19. Sintaxis abstracta del ensamble de proceso emulado.

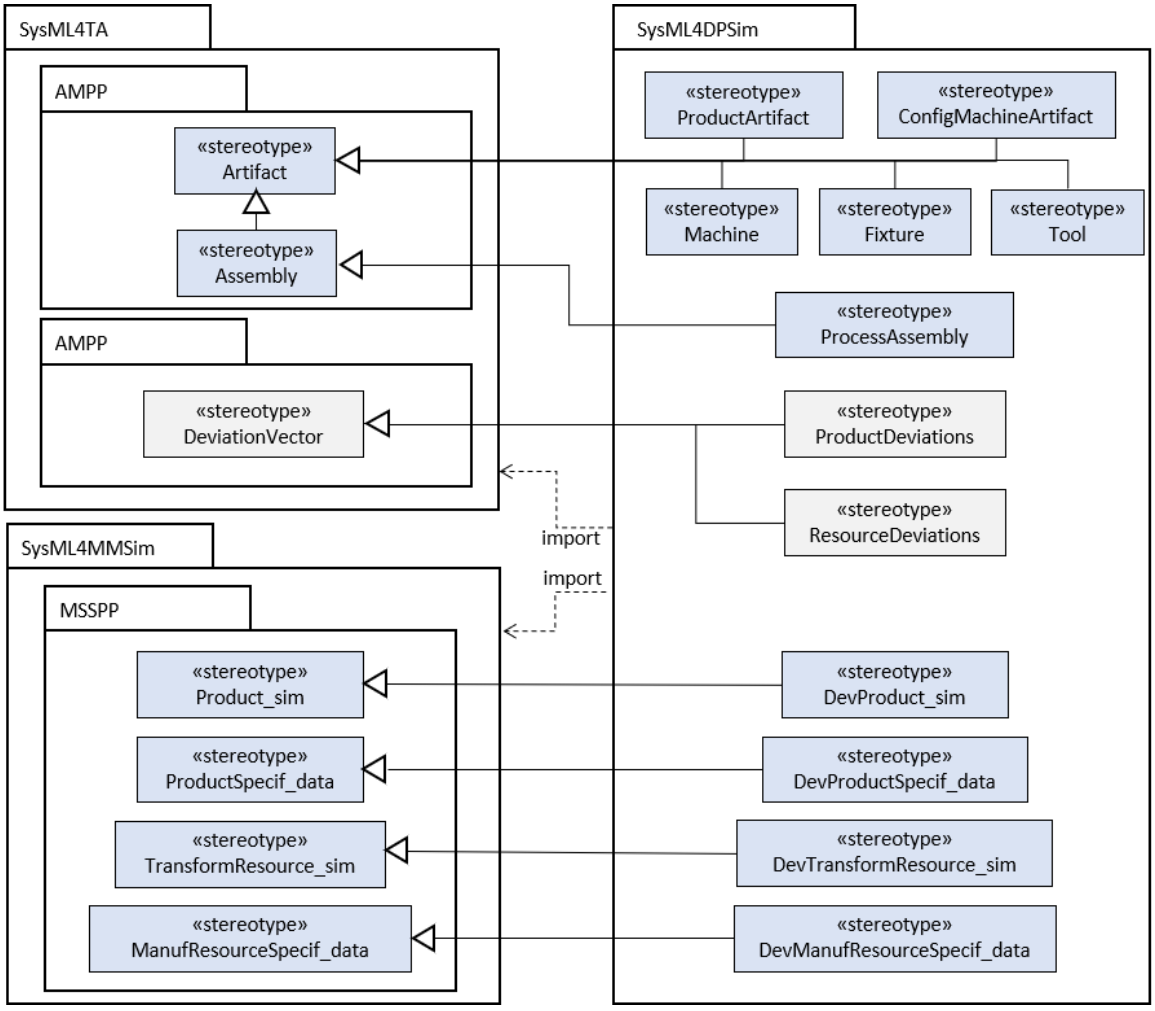


Figura 5.20. Diagrama de paquetes del perfil SysML4GDPS.

5.4.3. Implementación del metamodelo: perfil SysML4GDPS

Atendiendo al metamodelo descrito, este apartado presenta la implementación del lenguaje como un perfil SysML llamado SysML4GDPS (SysML for Geometric Deviation Propagation Simulation), cuyo contenido se muestra en la Figura 5.20.

En la Tabla 5.6 se definen brevemente los estereotipos considerados. La especificación completa del perfil, incluyendo las reglas OCL definidas en cada estereotipo, se recoge en el Anexo C.

Tabla 5.6. Estereotipos del paquete SSMPP.

Estereotipo	**Descripción**
«DevProduct_sim»	Especialización del estereotipo «Product_sim» para representar productos simulados incluyendo información sobre desviaciones geométricas. Por tanto, un «DevProduct_sim» Block debe incluir una propiedad tipeada por un «ProductDeviations» DataType.
«ProductDeviations»	Especialización del estereotipo «DeviationVector» empleado para definir las desviaciones geométricas en un «DevProduct_sim» Block.
«DevProductSpecif_data»	Especialización de «ProductSpecif_data» para representar la especificación de productos incluyendo información sobre su definición geométrica y tolerancias. Por tanto, un «DevProductSpecif_data» Block debe incluir una propiedad tipeada por un «ProductArtifact» Block.
«ProductArtifact»	Especialización del estereotipo «Artifact» para soportar la definición geométrica y tolerancias a nivel de especificación de un tipo de producto.
«DevTransformResource_sim»	Especialización de «TransformResource_sim» para representar recursos transformadores simulados incluyendo información sobre desviaciones geométricas, expresadas en una propiedad del «DevTransformResource_sim» Block tipeada por un «ResourceDeviations» DataType.
«ResourceDeviations»	Especialización del estereotipo «DeviationVector» empleado para definir las desviaciones geométricas en un «DevTransformResource_sim» Block.

«DevManufResourceSpecif_data»	Especialización de «ManufResourceSpecif_data» para representar la especificación de recursos de fabricación incluyendo información sobre su definición geométrica y tolerancias. Por tanto, un «DevManufResourceSpecif_data» Block debe incluir una propiedad tipeada por un «ConfigMachineArtifact» Block.
«ConfigMachineArtifact»	Especialización del estereotipo «Artifact» para soportar la definición geométrica y tolerancias a nivel de especificación de una máquina configurada.
«Machine»	Especialización del estereotipo «Artifact» para soportar la definición geométrica y tolerancias a nivel de especificación de una máquina, como parte de un «ConfigMachineArtifact» Block.
«Fixture»	Especialización del estereotipo «Artifact» para soportar la definición geométrica y tolerancias a nivel de especificación de un utillaje, como parte de un «ConfigMachineArtifact» Block.
«Tool»	Especialización del estereotipo «Artifact» para soportar la definición geométrica y tolerancias a nivel de especificación de una herramienta, como parte de un «ConfigMachineArtifact» Block.
«ProcessAssembly»	Especialización del estereotipo «Assembly» para soportar la definición geométrica y tolerancias a nivel de especificación de un ensamble de proceso, es decir, compuesto por una máquina configurada y uno o más productos. Por ello, un «ProcessAssembly» Block debe tener una propiedad tipeada por un «ConfigMachineArtifact» Block y al menos una propiedad tipeada por un «ProductArtifact» Block.

6. Transformaciones entre SysML y Modelica

Tal y como se comentó en la descripción de la metodología (Capítulo 4), una vez que se ha modelado el sistema de simulación en SysML y se ha comprobado su consistencia, mediante el uso de los perfiles propuestos en el capítulo anterior (Capítulo 5), el siguiente paso es transformar dichos modelos a un lenguaje textual que permita implementar estos modelos de simulación en código ejecutable. En el caso de esta propuesta, y como se ha comentado anteriormente (sección 4.3), Modelica ha sido el lenguaje seleccionado para este fin.

Aunque una posibilidad consiste en la definición manual de modelos Modelica, replicando los modelos SysML, la intervención directa del modelador incrementa de forma considerable el riesgo de introducir errores e inconsistencias, tanto intra-modelo (inconsistencias en el modelo textual según el metamodelo de Modelica) como inter-modelo (inconsistencias entre el modelo textual Modelica y el modelo SysML de referencia). Además, esta estrategia supone un elevado consumo de tiempo y recursos, como se constató tras las primeras experiencias de modelado con Modelica (ver Sección 3.4). Como alternativa, en este capítulo se exploran diversas soluciones que permiten automatizar esta transformación, implementando una aplicación que interpreta el modelo SysML y escribe el modelo textual Modelica correspondiente.

El capítulo se estructura en tres secciones. En primer lugar, se repasan trabajos previos sobre la transformación de modelos, y más concretamente sobre la transformación entre modelos SysML y Modelica, que han sido considerados a la hora de establecer las bases de la propuesta. Por lo que respecta a la segunda sección, cabe indicar que se centra en la propuesta de transformación de modelos SysML a Modelica, basada en el enriquecimiento del modelo SysML con un perfil específico y la posterior ejecución de aplicaciones, cuyos algoritmos también se mostrarán. Finalmente, en la tercera sección se presenta el desarrollo de una aplicación de transformación sencilla, que ha sido implementada para transformar modelos construidos a partir de librerías previamente definidas en ambos lenguajes (SysML y Modelica). A pesar de su limitado alcance, esta experiencia pone en valor la validez de la propuesta planteada y de los lenguajes y entornos seleccionados.

6.1. Antecedentes y bases de la propuesta

El uso de modelos SysML como base para la definición de modelos de simulación ejecutables ha sido un área bastante estudiada en los últimos años, fundamentalmente

enfocada en la transformación de estos modelos a otros definidos en diferentes lenguajes y entornos de simulación, como Arena [133] o Simulink [134] [135]. Sin embargo, en base a los intereses de esta investigación y a la metodología propuesta, el estudio se va a centrar en la transformación entre modelos SysML y modelos de simulación Modelica. Una de las primeras propuestas orientadas para resolver esta problemática fue el perfil ModelicaML [136] [137], una extensión SysML desarrollada para definir modelos de comportamiento Modelica aprovechando la notación y las capacidades gráficas de SysML. Para ello, se propuso emplear tanto diagramas propios de SysML (algunos de ellos con modificaciones) como diagramas adicionales ("Equation diagram" y "Simulation diagram"). Esta propuesta permite al usuario definir modelos Modelica utilizando una interfaz SysML, pero en la misma no se trata en detalle cómo realizar la transformación de los modelos.

Como continuación de esta experiencia, algunos autores del grupo que desarrolló ModelicaML participaron, años después, en una iniciativa del Object Management Group (OMG) orientada a especificar transformaciones bidireccionales entre SysML y Modelica, definiendo el perfil SysML4Modelica, que extiende SysML para dar soporte a los conceptos propios del lenguaje Modelica [138] [139]. En este punto, es necesario aclarar que, aunque Modelica tiene una sintaxis completamente definida, no existe una definición explícita de su metamodelo. Ante esta circunstancia, diversos trabajos han propuesto metamodelos expresados en MOF [94] para el lenguaje Modelica, como es el caso de las analizadas en [140] o de las mostradas en [141], pero la solución más extendida a la hora de integrar ambos lenguajes es el mencionado perfil SysML4Modelica.

Antes de entrar en detalle en aspectos propios de la semántica de Modelica y su modelado en SysML, a continuación se va a comentar uno de los múltiples trabajos que proponen el uso del perfil SysML4Modelica para soportar las transformaciones entre modelos [142]. En este trabajo se explora tanto la transformación automática de modelos SysML a modelos Modelica, como la transformación inversa de los resultados de la simulación (Modelica) al modelo SysML, para poder compararlos con los requisitos iniciales. Por lo que respecta al proceso de transformación de modelos SysML a Modelica, se plantea un proceso que se desarrolla en dos etapas totalmente diferenciadas. En la primera, conocida habitualmente como transformación de modelo a modelo (Model-to-Model Transformation - MMT), se enriquece el modelo SysML, incorporando la semántica propia de Modelica mediante la aplicación del perfil SysML4Modelica. Aunque esta tarea no se puede automatizar, en la misma se pueden definir mecanismos para comprobar la consistencia, por ejemplo, verificando que se cumple la semántica estática implementada como reglas en el perfil. En la segunda etapa, el modelo enriquecido se transforma a un modelo de texto (Model-to-Text Transformation - MTT), en este caso a un modelo Modelica. Finalmente, conviene aclarar que en este trabajo se hace uso de dos lenguajes específicos para implementar las aplicaciones de transformación MMT y MTT (Atlas Transformation Language y Acceleo respectivamente), pero este procedimiento es igualmente válido haciendo uso de otros lenguajes alternativos, como se explora más adelante.

6.1.1. Mapping de conceptos básicos de Modelica a SysML

A partir de las referencias consultadas, particularmente las centradas en el perfil SysML4Modelica [138], a continuación, se detallan las principales relaciones existentes a nivel de sintaxis abstracta (conceptos) y de semántica estática entre los lenguajes SysML y Modelica. Para ello, se ha elaborado la Tabla 6.1, que sintetiza las metaclases UML o estereotipos SysML que se corresponden con algunos de los principales conceptos manejados en Modelica.

Tabla 6.1. Elementos de SysML para la definición de modelos Modelica

Elemento SysML	**Elemento Modelica**	**Comentarios**
Model	Root model (ejecutable)	El modelo raíz de Modelica es aquel que se puede compilar para obtener el modelo ejecutable, y tiene su correspondencia en el concepto Model de SysML.
Package	package	En ambos casos, la estructuración y organización de elementos del modelo se realiza mediante paquetes.
Abstract block	partial model / block / class	En Modelica, la definición parcial de un modelo (o bloque o clase) no se puede instanciar (no es posible definir un constructor para un modelo parcial), sino que suele definirse para soportar características comunes de varios modelos que extienden dicha definición parcial. Este concepto corresponde a la definición de bloques abstractos, sobre las que se definen especializaciones que heredan sus características y añaden otras nuevas.
Block	model / block / class	A pesar de que Modelica diferencia entre los elementos de tipo Model, Block y Class, con pequeñas diferencias entre ellos, en SysML todos ellos equivalen al concepto Block.
Block	connector	Aunque en Modelica se define un tipo de puerto a través del concepto Connector, en SysML se realiza mediante la creación de un Block.
Attribute	variable	En Modelica, es posible definir variables con diferentes modificadores para identificar los parámetros, constantes, variables discretas, etc. En SysML, el concepto de variable corresponde al de atributo, aunque SysML no cuenta con especializaciones que correspondan con los modificadores de Modelica. En algunos lenguajes se propone una especialización de este estereotipo para incluir su propio atributo, en el que establecer el modificador correspondiente de una enumeración definida.
Constraint	equation	Las ecuaciones de Modelica se deben expresar mediante Constrainst, con expresiones opacas que definan la ecuación.
Part	component	La instanciación de un modelo (o bloque o clase) Modelica se representa en SysML como un uso de un bloque (parte).
Connector	connect equation	Las ecuaciones que en Modelica establecen conexiones entre puertos tienen su equivalente en SysML en el concepto de conector, o directamente el BindingConnector.

FunctionBehavior	function	Define un algoritmo capaz de calcular unos valores (outputs) a partir de unos argumentos (inputs). SysML no cuenta con ninguna construcción específica para este tipo de elementos, y se limita a implementarlos como expresiones opacas que especifican una FunctionBehavior.

Aunque en las propuestas consultadas se manifiesta la existencia de muchas correlaciones entre conceptos de ambos lenguajes, estas pueden ser de diferente naturaleza. Las observaciones mostradas en la Tabla 6.1 sobre las relaciones más básicas permiten mostrar que en muchas ocasiones existen elementos en ambos lenguajes que, más allá de la terminología empleada, se refieren al mismo tipo de concepto, como la Clase de UML y la correspondiente "clase" de Modelica. Sin embargo, también muestra ejemplos en los que ambas sintaxis abstractas se diferencian, planteando la necesidad de ampliar el metamodelo SysML mediante la creación de perfiles, de manera que los nuevos estereotipos den soporte a la semántica específica de Modelica. Es el caso, por ejemplo, del tratamiento de los puertos o puntos de conexión entre elementos. En UML se define la metaclase Puerto, cuya definición se expresa en una Clase (o Bloque en SysML). En cambio, en Modelica las definiciones de estos puntos de conexión tienen su propia designación ("connector"), mientras que su uso o instanciación no tiene un nombre distintivo de cualquier otro componente. Además, también es reseñable el uso diferente de un mismo término. En Modelica se llama "connector" al tipo de punto de conexión, mientras que en UML y SysML el conector es el elemento que conecta dos puertos.

6.1.2. Aplicaciones para la automatización de transformaciones

La transformación automática de modelos requiere establecer las correspondientes relaciones de mapeo entre conceptos de los dos lenguajes (origen y destino) considerados, pero también la definición de una aplicación ejecutable que dé soporte a las transformaciones y genere el nuevo modelo. Hay que tener en cuenta, además, que en las transformaciones de modelo a texto no se produce solo un cambio en el metamodelo considerado (sintaxis abstracta y semántica estática), sino que también se realiza un cambio en la sintaxis concreta, porque los formatos y las notaciones (origen y destino) son distintas, como ocurre en las transformaciones de modelos SysML a modelos textuales Modelica. Por lo tanto, esta aplicación de transformación debe asegurar la correcta definición y estructuración del nuevo modelo según la sintaxis concreta del lenguaje de destino.

Uno de los formalismos más extendidos/utilizados para expresar las reglas de transformación entre modelos es el estándar de la OMG Query/View/Transformation Language (QVT) [118]. Esto se debe a que es compatible con el enfoque Model-Driven Architecture (MDA) [36] y a que está basado en MOF y UML. Al respecto, conviene aclarar que QVT no es en realidad un lenguaje único, sino que se trata de un conjunto de lenguajes que incluye el QVT Relational y el QVT Operational, y además que ha servido como formalismo para desarrollar otros lenguajes.

En lo referido a lenguajes con los que implementar las aplicaciones de transformación, existen en el mercado diversas alternativas, como Atlas Transformation Language

(ATL) [143], Kermeta [144], o XTend [145]. Sin embargo, en este trabajo se explora el uso de la familia de lenguajes Epsilon, una iniciativa desarrollada en el marco del proyecto Eclipse GTM [146], en el que también fue desarrollado el software Papyrus, la herramienta elegida en esta investigación para el modelado con SysML.

El núcleo de Epsilon es el lenguaje de objetos Epsilon (EOL) [147], un lenguaje que da soporte a los conceptos expresados con OCL y que proporciona características para la modificación de modelos, el acceso a múltiples modelos, construcciones convencionales de programación (variables, bucles, etc.), la elaboración de perfiles, etc. EOL se puede utilizar como un lenguaje general de gestión de modelos, pero también se utiliza como base para definir lenguajes más específicos, orientados a dar soporte a tareas como la comparación de modelos (Epsilon Comparison Language - ECL) [148], la fusión de modelos (Epsilon Merging Language - EML) [149], la validación de modelos (Epsilon Validation Language - EVL) [150] o transformación de modelo a texto (Epsilon Generation Language - EGL) [151].

Atendiendo a todo lo expuesto y a los intereses de la propuesta, surge la necesidad de explorar si el lenguaje EGL es válido para soportar las transformaciones planteadas entre SysML y Modelica, y si es factible la implementación de una aplicación basada en EOL en el entorno de modelado elegido (Papyrus).

6.2. Mecanismo de transformación propuesto

Como se ha comentado anteriormente, la transformación de modelos SysML a modelos textuales de Modelica ha sido ampliamente explorada en la literatura con resultados prometedores. Sin embargo, teniendo en consideración los intereses de esta investigación, la adopción de alguna de estas propuestas y el desarrollo de transformaciones automáticas excede el alcance de este trabajo, que se limita a plantear las bases que sustentarían el mecanismo de transformación.

Esta sección se estructura en cuatro apartados, presentando inicialmente los fundamentos de la propuesta. En segundo lugar, se describe el perfil SysML desarrollado para dar soporte a las transformaciones, y cuya especificación se detalla en el Anexo D. El tercer apartado se dedica a plantear el algoritmo necesario para transformar los modelos SysML a texto Modelica y que, en base a la propuesta presentada, debe ser definido en un código EGL. Finalmente, se comenta brevemente la aplicación que permite ejecutar las transformaciones, y que será definida mediante un código EOL.

6.2.1. Bases de la propuesta

El esquema que representa gráficamente la transformación de modelos se muestra en la Figura 6.1, diferenciando dos niveles en las tareas de modelado y transformación: desarrollador y usuario. Se entiende por usuario a quien construye modelos y experimentos de simulación haciendo uso de librerías SysML de elementos predefinidos, por lo que no tiene que ser un experto en el lenguaje de modelado (SysML y Modelica). En cambio, el desarrollador es el encargado de definir estas librerías de

elementos reutilizables, lo que requiere de un conocimiento más profundo de los lenguajes y dominios que son objeto de interés.

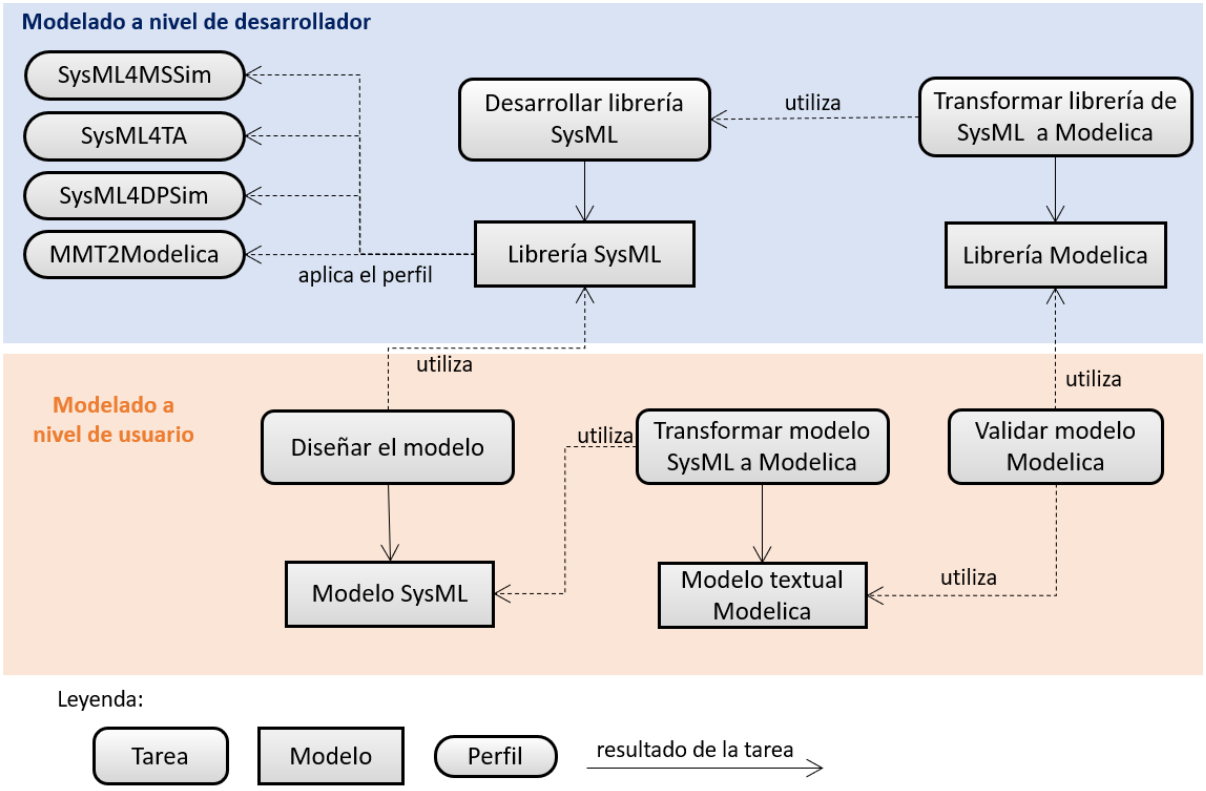


Figura 6.1. Detalle de las tareas de modelado a nivel de desarrollador y usuario.

El desarrollo de las librerías SysML no se limita a un modelado estructural, debiéndose contemplar también aspectos comportamentales (ecuaciones, algoritmos, funciones, etc.). La complejidad que esto supone requiere de comprobaciones de consistencia que, como se ha propuesto, implican la aplicación de perfiles como SysML4MSSim, SysML4TA o SysML4DPSim (ver Capítulo 5). Pero, además, la librería debe ser enriquecida con la aplicación de otro perfil que establece los conceptos básicos de Modelica necesarios para facilitar las transformaciones. Este perfil SysML, denominado Model-to-Model Transformations to Modelica (perfil MMT2Modelica), será presentado en el siguiente apartado (6.2.2) y su implementación detallada se recoge en el Anexo D. Como se ve en la Figura 6.1, el modelado a nivel de desarrollador incluye además una actividad que tiene por función crear librerías en Modelica a partir de su definición en SysML, una tarea que podría automatizarse mediante la implementación de aplicaciones de transformación MMT. Para ello, se propone el desarrollo de una aplicación implementada con lenguajes de la familia Epsilon, que debe acceder e interpretar modelos SysML enriquecidos, y generar automáticamente modelos Modelica equivalentes, replicando la estructura de paquetes y definiendo completamente todos los elementos en lenguaje Modelica. La generación automática de librerías Modelica a partir de librerías SysML, cuya consistencia ha sido previamente validada, permite asegurar también la consistencia de las construcciones Modelica, que quedan a disposición de los usuarios de este lenguaje de simulación.

El mismo mecanismo de transformación que se aplica a nivel de desarrollador también es aplicable a nivel de usuario, de manera que los usuarios pueden definir sus propios modelos de simulación y experimentos en un entorno de modelado SysML, haciendo uso de las librerías. Una vez definidos estos modelos en SysML, la aplicación de transformación accede e interpreta los modelos de usuario, creando el modelo texto Modelica correspondiente. Para ello, la transformación puede apoyarse en las librerías Modelica disponibles o crear construcciones nuevas si es necesario. Conviene remarcar, de nuevo, que el modelado a partir de unos elementos de librería reutilizables, cuya consistencia ha sido validada, permite asegurar también la consistencia de los modelos de simulación y de experimentos Modelica resultantes de la transformación.

Cabe recordar que, como se ha comentado en la descripción de la metodología propuesta (Capítulo 4), se ha seleccionado la herramienta Papyrus para el modelado con SysML. Se trata de un entorno de modelado que admite trabajar con lenguajes Epsilon, incluyendo la implementación y la ejecución de aplicaciones de transformación de modelos. En este sentido, la propuesta explora el uso de Epsilon Generation Language (EGL) y Epsilon Object Language (EOL) en este entorno, y plantea los algoritmos necesarios para abordar dichas transformaciones.

6.2.2. Lenguaje para la transformación de modelos SysML a texto Modelica

Este apartado describe de forma resumida los principales conceptos manejados en este lenguaje específico para dar soporte a las transformaciones de modelos SysML a texto Modelica, y que posteriormente se implementarán como estereotipos del perfil MMT2Modelica. La mayoría de estos conceptos están orientados a identificar tipos de elementos Modelica, incluyendo algunas reglas básicas propias de su semántica estática. Estas reglas no están descritas con el detalle y rigor necesario, pero se muestran para ejemplificar el potencial del uso de perfiles como medio para incorporar la semántica de otro lenguaje como Modelica.

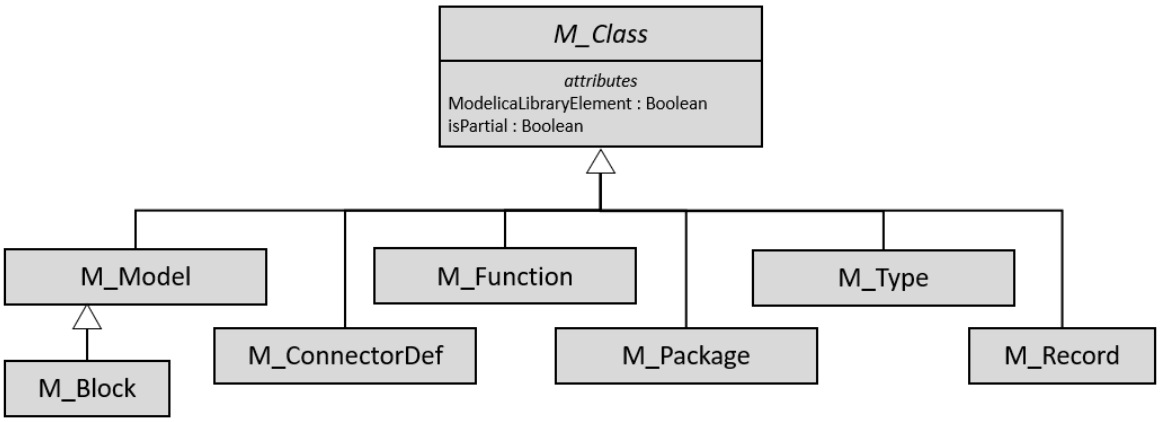


Figura 6.2. Sintaxis abstracta de las clases básicas Modelica

Gran parte de los conceptos manejados en este lenguaje son especializaciones de M_Class, un concepto que representa cualquier clase Modelica y que en esta propuesta

se define como abstracto para que no sea instanciable y sea necesario utilizar alguna de sus especializaciones, aunque el lenguaje Modelica sí admite la definición de clases. M_Class se caracteriza por tener dos atributos booleanos: "ModelicaLibraryElement" se emplea para identificar aquellas construcciones que también están disponibles en la librería Modelica, mientras que "isPartial" indica si un elemento es parcial (no completo) y, por tanto, que no se pueden instanciar, un mecanismo similar al uso de bloques abstractos en los modelos SysML. Las principales especializaciones de M_Class se muestran en la Figura 6.2, que describe una parte de la sintaxis abstracta del metamodelo propuesto.

Una de las clases fundamentales de Modelica es el "model", representado como M_Model en este metamodelo, aunque también es habitual emplear el concepto de bloque (M_Block) para referirse a aquellos modelos en los que los puertos tienen una direccionalidad definida (input/output). Como se muestra en la Figura 6.3, un M_Model está compuesto por un conjunto propiedades (*M_Properties*) de diferentes tipos, como son los componentes (M_Component), puertos (M_ConnectionUse), parámetros (M_Parameter) o variables (M_Variable). Además, en un M_Model se pueden definir ecuaciones (M_Equation) en las que definir comportamientos como expresiones matemáticas, y conexiones (M_Connection), mediante las cuales relacionar diferentes propiedades parte o puertos, generalizadas con el concepto abstracto *M_Part*. Conviene comentar además *M_Property* y *M_Part* se definen como conceptos abstractos para establecer características comunes a diversos tipos de propiedades. Por ejemplo, *M_Property* cuenta con atributos que permiten definir de modificadores ("inner", "outer", "replaceable", etc.), o la especificación de tamaños de estructuras de datos (matrices, vectores, etc.) que van más allá del concepto "multiplicity" de SysML.

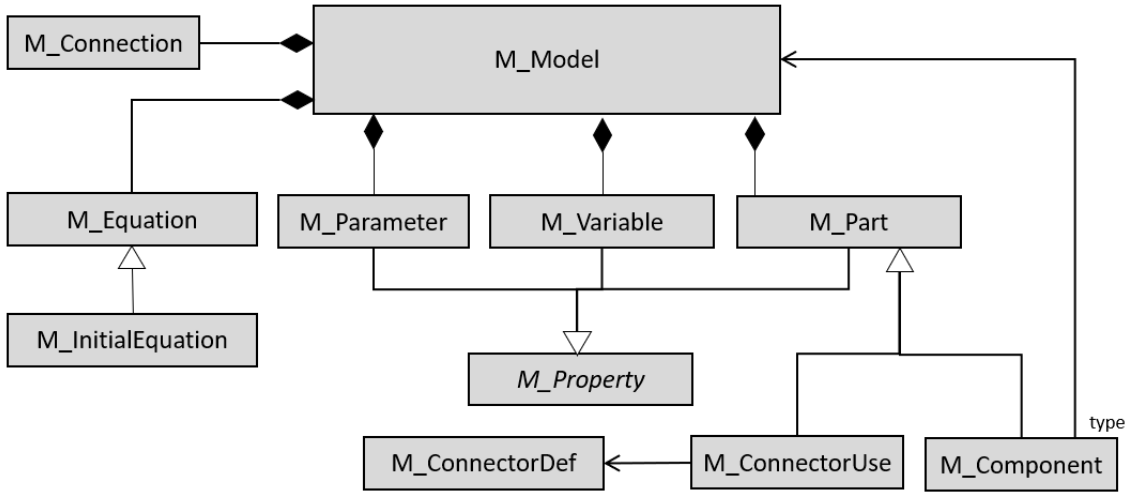


Figura 6.3. Sintaxis abstracta de M_Model.

Otra clase típica en la construcción de librerías y modelos Modelica es la definición de funciones (M_Function), que básicamente se emplean para definir algoritmos. La llamada a una función debe incluir la definición de unos argumentos, que son datos de entrada (inputs) a partir de los cuales la función ejecuta algoritmos para obtener unos resultados (outputs). Las librerías y modelos Modelica también suelen incluir la

definición de paquetes (M_Package), tipos (M_Type) y clases “record” (M_Record). Adicionalmente, aunque el metamodelo de Modelica no hace distinción, se propone el concepto de M_Experiment para identificar aquellos modelos que tienen valores en sus parámetros y, por tanto, son ejecutables, una construcción que en SysML se modelará mediante InstanceSpecifications.

Comentados los principales conceptos del lenguaje propuesto, a continuación, se pasa a describir los estereotipos del perfil MMT2Modelica, que se presentan en la Tabla 6.2.

Tabla 6.2. Estereotipos del perfil MMT2Modelica

Estereotipo	**Descripción**
«M_Class»	Estereotipo abstracto que generaliza todos los estereotipos del perfil incorporando dos atributos booleanos comunes a todos ellos. El atributo ModelicaLibraryElement indica si el elemento estereotipado está disponible en la librería de Modelica o no, mientras que el atributo isPartial indica si se trata de una definición parcial (bloque abstracto).
«M_Model»	Especialización de *«M_Class»* que representa un modelo Modelica.
«M_Block»	Especialización de «M_Model» en el que todos los «M_ConnectorUse» definidos deben estar definidos como “input” o “output”.
«M_Property»	Estereotipo abstracto que representa cualquier propiedad de una clase. Incorpora el atributo MultiplicityParameter para definir una multiplicidad no está determinada por un valor entero, sino que hacer referencia a un parámetro definido en el modelo. MultiplicityParameter es de tipo «M_Parameter», es decir, que apunta a una propiedad estereotipada como «M_Parameter» o puede ser nulo. El estereotipo *«M_Property»* también cuenta también con atributos booleanos que permiten identificar si dicha propiedad es un reemplazable, o de tipo “outer” o “inner”.
«M_Part»	Estereotipo abstracto que especializa el *«M_Property»* para identificar una propiedad de un «M_Class» Block (o cualquiera de sus especializaciones).
«M_ConnectorUse»	Estereotipo que especializa el «M_Part» para definir un puerto perteneciente a un «M_Class» Block (o cualquiera de sus especializaciones), y que está tipeada por un «M_ConnectorDef» Block.
«M_ConnectorDef»	Estereotipo de un Block que contiene la definición de un «M_ConnectorUse» Port
«M_Component»	Estereotipo que especializa el «M_Part» para definir una propiedad de un «M_Class» Block (o cualquiera de sus especializaciones) tipeada por un «M_Class» Block (o cualquiera de sus especializaciones)
«M_Conection»	Estereotipo de un conector que une dos propiedades estereotipadas con especializaciones de *«M_Part»*
«M_Equation»	Estereotipo que extiende la Constraint para definir, en su expresión opaca, el campo “equation” de un modelo Modelica.
«M_InitialEquation»	Especialización del estereotipo «M_Equation» empleado para identificar las ecuaciones que establecen las condiciones iniciales de la simulación.

«M_Variable»	Especialización del *«M_Property»* para identificar una propiedad valor de un «M_Class» Block (o cualquiera de sus especializaciones). Cuenta con atributos booleanos que permiten identificar si es una variable discreta.
«M_Parameter»	Especialización del *«M_Property»* para identificar una propiedad valor que representa un parámetro de un «M_Class» Block (o cualquiera de sus especializaciones), es decir, propiedades que deben adoptar un valor en la instanciación del bloque.
«M_Function»	Estereotipo que extiende el FunctionBehavior para definir, mediante una expresión opaca, un algoritmo. Aunque el FunctionBehavior admite la definición de propiedades de tipo input, output, etc. similares a las propias de las funciones Modelica, en el perfil no se aborda esta definición detallada, expresando la función como una expresión opaca.
«M_Package»	Estereotipo que especializa el «Package» de SysML.
«M_Type»	Estereotipo que especializa el «Enumeration» de SysML.
«M_Record»	Estereotipo que especializa el «DataType» de SysML.
«M_Experiment»	Estereotipo que especializa el «InstanceSpecification» de SysML para identificar la instancia de un bloque en el que se define un modelo de simulación, asignando valores a sus parámetros y definiendo, por tanto, un experimento concreto.

6.2.3. Definición del algoritmo de transformación

La automatización de las transformaciones de modelos requiere de la implementación de códigos que definen toda una serie de algoritmos, que son diseñados para acceder al modelo origen (SysML) y navegar por su contenido, con el objetivo de traducirlo a un modelo texto equivalente definido en el lenguaje destino (Modelica). Con este propósito, en esta propuesta se explora el desarrollo de códigos implementados con EGL.

Antes de abordar cada una de las partes de un EGL, conviene comentar algunas consideraciones generales que diferencian los modelos SysML y Modelica, y que se resuelven con mecanismos aplicados a lo largo de la transformación. Es el caso, por ejemplo, de la identificación de cualquier elemento del modelo (de tipo de UML) incluyendo el directorio de paquetes en el que se define. En SysML, cualquier elemento de tipo NamedElement (metaclase de UML) tiene una propiedad llamada QualifiedName, que es una cadena de caracteres que define la sucesión de paquetes en la que está contenido el bloque, junto con el nombre del propio bloque. En esta cadena de caracteres la separación entre el nombre de un paquete y el de un elemento empaquetado (un subpaquete, por ejemplo) se emplea dos veces el signo dos puntos (“::”). En cambio, en Modelica también se emplea una construcción similar para referenciar elementos, si bien en este caso la sintaxis concreta de Modelica emplea un punto (“.”) como separación entre continente y contenido. Por tanto, para utilizar el QualifiedName de SysML en la transformación de modelos es necesario modificar la cadena de caracteres, para lo cual se utiliza la función “replaceAll()”.

Los códigos EGL propuestos se estructuran en dos grandes partes, una inicial dedicada a definir las variables que dan soporte a información, que será utilizada de forma repetitiva a lo largo del código, y una segunda parte en la que se implementa un algoritmo que tiene tres bucles anidados dedicados a identificar y transformar los paquetes, bloques e instancias del modelo SysML, tal y como se comenta a continuación.

A. Transformación de la estructura de paquetes

El bucle general de transformación de modelos se inicia con la identificación de los paquetes del modelo SysML a transformar, estereotipados como «M_Package». Para cada uno de estos paquetes se escanea su contenido para listar los clasificadores (bloques, subpaquetes, etc.) definidos. A partir de esta información, se ejecuta un bucle interno en el que se repiten toda una serie de operaciones orientadas a definir un paquete Modelica por cada paquete «M_Package» de SysML.

Adoptando esta estrategia, aunque se someta a transformación todo un modelo SysML, el elemento que realmente se transforma a Modelica es un paquete SysML estereotipado como «M_Package», que puede ser el paquete general del modelo (en SysML, Model es una especialización de Package) o alguno de los subpaquetes particulares definidos en el modelo. El paquete «M_Package» más general se transforma en un paquete Modelica, definido en el fichero generado por la transformación.

El bucle empleado para definir cada paquete empieza definiendo el encabezado del paquete Modelica, escribiendo el término “package” seguido del nombre (name) del paquete SysML correspondiente. A continuación, se transforman las relaciones con otros paquetes, definiendo lo que en Modelica son mecanismos de extensión o de importación. Seguidamente se aborda la definición de subpaquetes Modelica, transformando los subpaquetes «M_Package» de SysML mediante la ejecución de forma anidada del mismo bucle que se está describiendo. Por último, se transforman los diferentes bloques SysML definidos y estereotipados como «M_Model» o «M_Block», para lo cual se ejecuta el bucle descrito en el siguiente punto. Finalmente, tras definir el último de los modelos o bloques Modelica, se acaba la definición del paquete Modelica mediante el término “end” seguido del nombre (name) del paquete SysML y el signo punto y coma (“;”), con el que se cierra la definición del elemento.

Además, conviene comentar que cuando el modelo de usuario se ha construido a partir de librerías el número de elementos definidos por el usuario, y por lo tanto objeto de la transformación, es habitualmente muy reducido, llegando en ocasiones a contener un único bloque en el que se define el modelo de simulación mediante usos (relaciones de composición) de elementos predefinidos. Esto permite abordar transformaciones sencillas basadas en el uso de elementos de librería existentes. Por el contrario, cuando el modelo se ha construido desde cero, como es el caso del desarrollo de las librerías, en muchas ocasiones se tienen que considerar estructuras complejas de paquetes y elementos, que requieren un proceso de transformación más complejo y detallado. Sin

embargo, la propuesta basada en bucles permite abordar ambos supuestos con el mismo código.

B. Transformación de bloques SysML

En este punto se aborda la descripción del bucle del algoritmo que transforma los bloques SysML estereotipados como «M_Model» o «M_Block» en modelos o bloques Modelica respectivamente. En el caso de la transformación de otros elementos, como tipos, funciones o conectores Modelica, el bucle sería muy similar, obviando algunas de las partes presentadas a continuación que no aplicarían, según el caso. Dada la complejidad que pueden llegar a tener este tipo de transformaciones, la descripción se divide en diversos puntos, que abordan la definición de las partes que caracterizan un modelo Modelica.

Encabezado

Las primeras líneas del texto en el que se define un modelo, bloque o conector Modelica se encabezan con el término "model", "block" o «connector» respectivamente, seguido del nombre del bloque SysML correspondiente.

La siguiente línea del código Modelica se emplea para definir extensiones, es decir, si el bloque definido extiende un bloque existente. Este mecanismo es equivalente a la relación de especialización de SysML, por lo que es necesario comprobar si el bloque SysML a transformar se ha definido como especialización de otro bloque. De ser así, se utiliza el término "extends" seguido del identificador (QualifiedName adaptado a Modelica) del bloque al que se especializa. Además, tanto SysML como Modelica cuentan con mecanismos similares con los que una especialización puede definir propiedades que reemplazan propiedades definidas en su generalización. En SysML, esta característica se indica en el campo "RedefinedProperty" de la nueva propiedad, referenciando la reemplazada. Por tanto, el código de transformación debe navegar por todas las propiedades del bloque SysML para comprobar si en alguna de ellas este campo "RedefinedProperty" tiene algún valor (no nulo), en cuyo caso se traslada al código Modelica entre paréntesis, escribiendo la palabra "redeclare" seguida del QualifiedName de la nueva definición (tipo de la propiedad) y el nombre de la propiedad.

Parámetros y variables

Tras el encabezado, las siguientes líneas definen las variables y parámetros. Para ello, se identifica el conjunto de propiedades estereotipadas como «M_Variable» y «M_Parameter», y para cada una de ellas se escribe una línea de código Modelica que cuenta con las siguientes partes:

- *Modificadores*. Términos de la sintaxis abstracta de Modelica que indican alguna característica de la propiedad, por ejemplo, "discrete", "inner", "outer", etc. La transformación debe consultar los atributos del estereotipo «M_Variable» o «M_Parameter» para comprobar si debe definir algún modificador en el código Modelica. Por ejemplo, si en el estereotipo

«M_Variable» se define isDiscrete=True, en el código Modelica se escribe el modificador “discrete” al inicio de la línea de la definición de la variable.

- *Tipo*. Elemento de definición del tipo de variable o parámetro, que en el modelo SysML puede ser una primitiva (real, entero, booleano, etc.), una Enumeration estereotipada como «M_Type», o un DataType estereotipado como «M_Record». En estos dos últimos casos, la transformación debe explicitar su QualifiedName, mientras que en el caso de las primitivas se debe escribir el término Modelica correspondiente (Real, Integer, Boolean, etc.).

- *Nombre de la variable*. Corresponde con el nombre de la propiedad SysML a transformar.

- *Valores*. En el caso de los parámetros, es posible que en SysML se hayan definido sus valores concretos mediante InstanceSpecifications para definir experimentos concretos. Por ello, se explora si existen instancias del bloque, en cuyo caso se escribe un signo de igualdad (=) y los valores de los slots definidos.

- *Tamaño*. En Modelica es posible definir propiedades con múltiples tamaños, no solo un entero como sucede en SysML. Esto permite definir vectores y matrices (vector de vectores) en Modelica, pero complica su definición en SysML. Para superar esta diferencia, el estereotipo «M_Property» incluye la propiedad MultiplicityParameter mediante la cual definir una multiplicidad diferente a las tratadas habitualmente en SysML.

Componentes y puertos

La siguiente parte del código define los componentes del modelo Modelica, que en SysML está modelados con parts (usos de bloques) estereotipadas como «M_Part» o con puertos estereotipados como «M_ConnectorUse». Para expresar estos elementos en Modelica es necesario definir sus modificadores (si los tuviera), su QualifiedName y su nombre (rol de la parte o nombre del puerto), de forma similar a la definición de variables comentada anteriormente.

Llegados a este punto, se ha completado la parte del bucle dedicada a la definición de las propiedades del modelo Modelica, principalmente variables, parámetros, componentes y puertos. Para ejemplificar las operaciones de transformación comentadas en los ítems anteriores, en la Figura 6.4 se representa la definición de un sencillo bloque «M_Model» en SysML y, seguidamente, se mostrará el código correspondiente en Modelica.

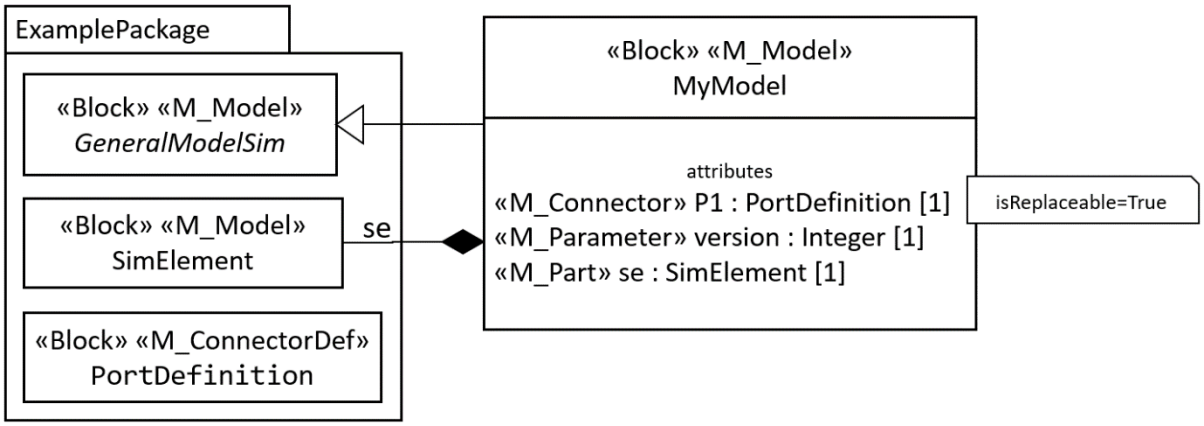


Figura 6.4. Ejemplo de transformación de propiedades

```
model MyModel
  extends ExamplePackage.GeneralModelSim;
  parameter Integer version;
  ExamplePackage.SimElement se;
  replaceable ExamplePackage.PortDefinition P1;
end MyModel;
```

Ecuaciones y algoritmos

Tras la definición de todas las propiedades, las siguientes líneas de un modelo Modelica se dedican a la especificación de ecuaciones. En primer lugar, se comprueba si es necesario definir ecuaciones iniciales, que establecen la inicialización de algunas variables. Para ello, el algoritmo debe identificar si existe en el modelo SysML alguna restricción estereotipada como «M_InitialEquation», en cuyo caso se define el encabezado "initial equation" y se transcribe el código expresado en la OpaqueExpression de dicha restricción.

Seguidamente, se inicia la sección correspondiente a las ecuaciones con el término "equation" y se transcriben las expresiones opacas de las restricciones estereotipadas como «M_Equation», si las hubiera. La sección de ecuaciones también se emplea en Modelica para establecer las conexiones entre componentes internos. Para ello, se identifican los conectores en el bloque SysML, que deben ser BindingConnectors estereotipados como «M_Connection». Estos conectores se expresan en Modelica como ecuaciones de tipo "connect", indicando entre paréntesis los puertos de las propiedades que está relacionando el conector.

De forma similar se procede con las restricciones estereotipadas como «M_Algorithm», transcribiendo las expresiones opacas al texto Modelica precedidas por el encabezado "algorithm" propio de las funciones Modelica.

De nuevo, en la Figura 6.5 se presenta un sencillo ejemplo para ilustrar las transformaciones vinculadas con la sección de ecuaciones.

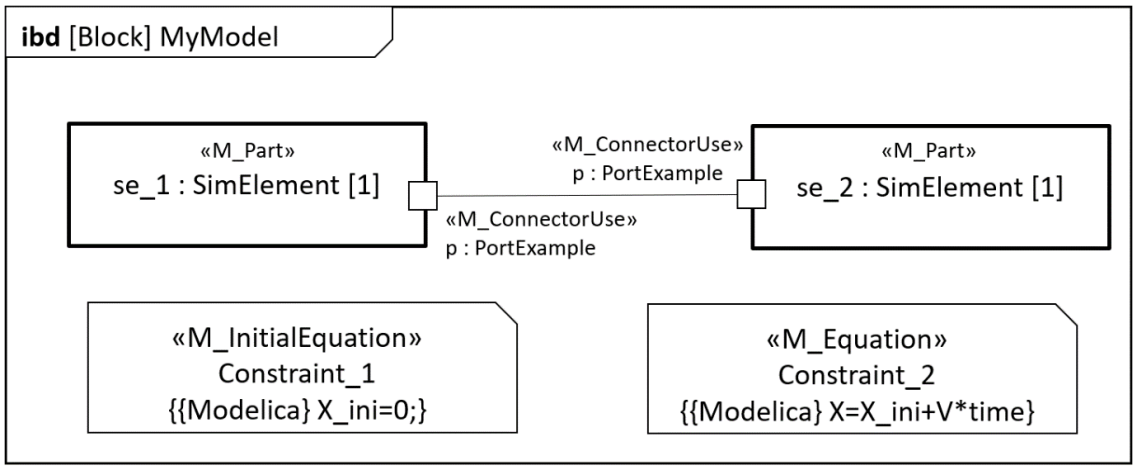


Figura 6.4. Ejemplo de transformación de ecuaciones y conexiones.

```
model MyModel
  Real X_ini, X, V;
   ExamplePackage.SimElement se_1;
   ExamplePackage.SimElement se_2;
initial equation
  X_ini=0;
equation
  X=X_ini+V*time;
  connect (se_1.p, se_2.p);
end MyModel;
```

Fin del modelo.

Al finalizar la definición del apartado de ecuaciones, es momento de terminar la definición del modelo de simulación (en el ejemplo, "end MyModel;") y, si es necesario, proseguir con la definición de los demás bloques, DataTypes o paquetes contenidos en el paquete raíz, en caso de que existan. Para ello, se repite el mismo procedimiento descrito anteriormente y, finalmente, se cierra la definición del paquete.

C. Transformación de instancias SysML

Además del modelado de sistemas de simulación, SysML se puede emplear para definir experimentos particulares de un modelo de simulación, para lo cual es necesario asignar valores a todos los parámetros definidos. El mecanismo que se propone para ello es definir elementos de tipo InstanceSpecification y estereotipados como «M_Experiment». Por tanto, se define un bucle para transformar los experimentos definidos en SysML a modelos ejecutables Modelica. Para ello, tras identificar todos los experimentos definidos, se procede a transformar cada uno de ellos. En primer lugar, se define el encabezado del modelo Modelica ("model") seguido del nombre que la InstanceSpecification. A continuación, se indica que este modelo Modelica extiende otro modelo, que será el clasificador (campo Classifier) al que está instanciando, y que necesariamente debe ser un bloque «M_Model». Para completar esta extensión, se indica entre paréntesis el conjunto de valores de los parámetros del modelo. Para ello, se navega por todos los slots definidos en la InstanceSpecification, indicando en el texto Modelica el parámetro y el valor del slot.

6.2.4. Definición de la aplicación de transformación

Una vez implementado el código EGL que define el procedimiento de transformación descrito anteriormente, es momento de abordar la implementación de un código EOL que define la aplicación ejecutable de transformación. Esta aplicación utiliza el código EGL y lo aplica sobre cualquier modelo SysML concreto disponible en el espacio de trabajo. Como resultado de la ejecución de esta aplicación, se genera automáticamente un nuevo fichero con el nombre y la extensión deseada, que contiene el texto resultante de la transformación, en este caso un modelo texto Modelica. Para ello, se implementa un código sencillo definido en EOL, cuyas partes se describen a continuación junto con un ejemplo ilustrativo.

- Nombre de transformación, definida como una regla (rule) en el lenguaje EOL. En el ejemplo tratado, se define la regla "SysML2Modelica" como:

```
rule SysML2Modelica
```

- Tipo de objeto origen y resultado de la transformación (transform ____:____) e inicio de la definición de los parámetros mediante un signo de llave "{". En el ejemplo, se indica la transformación de un modelo en otro modelo mediante la expresión:

```
transform Model: Model{
```

- Método empleado, que debe ser un código EGL en el que se define el procedimiento de transformación comentado en el apartado anterior. A continuación se ejemplifica la implementación de esta línea haciendo uso de un código EGL llamado "SysML2Modelica_template.egl".

```
template : "SysML2Modelica_template.egl"
```

- Nombre del fichero resultante. En el ejemplo desarrollado, se propone utilizar el mismo nombre que el modelo origen (Model.name), seguido por la cadena de caracteres "_Modelica.mo" para identificar rápidamente que se trata de un modelo expresado en Modelica, y con la extensión adecuada para ser abierto en un entorno de modelado como OpenModelica. Tras esta indicación, se cierra la definición de la regla con una llave "}".

```
target : Model.name + "_Modelica.mo" }
```

Además de definir el código EOL, Papyrus permite definir la configuración de cada ejecución a través de la interfaz gráfica de ventanas, mediante las cuales se selecciona el modelo origen sobre el que ejecutar dicha transformación, y el directorio en el que almacenar los ficheros resultantes, además de otros muchos parámetros y opciones. Conviene indicar que tanto el ejemplo de código EOL como la interfaz de configuración en Papyrus han sido empleados en las transformaciones desarrolladas en el caso de estudio presentado en el Capítulo 8.

6.3. Transformaciones implementadas

Con el fin de ejemplificar la transformación de modelos y validar los lenguajes y entornos seleccionados en la propuesta presentada anteriormente, se ha desarrollado una aplicación de transformación simple que permite obtener modelos textuales en Modelica a partir de modelos SysML de usuario definidos con elementos de la librería SysML. Además, para limitar la complejidad de la experiencia, se supone que se dispone de una librería en Modelica equivalente a la de SysML. En el caso ideal, las librerías Modelica se deberían obtener mediante transformación automática a partir de las librerías SysML, pero en este caso se han definido manualmente. Con este supuesto, la aplicación desarrollada permite la transformación automática de modelos mediante el mapeado de los elementos de ambas librerías, es decir, que aprovecha la correspondencia uno a uno entre elementos de las librerías de ambos lenguajes.

Para hacer uso de esta aplicación, es necesario tener en cuenta ciertas consideraciones adicionales. Por ejemplo, únicamente se pueden procesar modelos construidos a partir de elementos de librería, y es condición indispensable haber aplicado algunos estereotipos clave del perfil MMT2Modelica, tanto en la librería como en las nuevas construcciones definidas por el usuario. Además, las librerías SysML y Modelica deben tener la misma estructura de paquetes y utilizar la misma terminología para identificar los diferentes elementos, lo que también facilita la implementación de las transformaciones. Tras ejecutar la transformación del modelo y obtener código Modelica, es necesario abrir dicho fichero en un entorno de modelado Modelica, como OpenModelica, y disponer también de la mencionada librería, que es la base que permite validar y compilar el modelo.

La implementación completa del código EGL se recoge en el Anexo E, mientras que en el Capítulo 8 se presenta un caso de estudio en el que se puede comparar el modelo definido en SysML y el resultado de ejecutar este tipo de transformaciones automáticas.

A pesar de todas estas limitaciones y condicionantes comentados, la experiencia de desarrollar este tipo de transformación automática permite corroborar la adecuación de los lenguajes y entornos seleccionados en la propuesta, así como gran parte del algoritmo presentado en la sección anterior.

7. Librerías

Como se ha comentado en el capítulo anterior, la propuesta de automatización implementada en la actualidad (sección 6.3) se ha desarrollado bajo la premisa de disponer de una serie de librerías de elementos de modelado reutilizables, modeladas inicialmente en SysML y trasladadas, de forma manual o automática, a Modelica. Ambas librerías tienen la misma estructura de paquetes y emplean la misma terminología, de forma que se dispone de librerías estructuralmente equivalentes. Esta característica es condición indispensable para poder utilizar el proceso de mapeo como mecanismo de transformación.

A pesar de las similitudes, ambas librerías cuentan con evidentes diferencias. Por lo que respecta a las librerías definidas con SysML cabe indicar que en ellas se aplican los perfiles desarrollados en esta investigación, tanto los perfiles orientados a asegurar la consistencia de los modelos (Capítulo 5) como el perfil MMT2Modelica, propuesto para facilitar la transformación de los modelos SysML a Modelica (Capítulo 6). Además, considerando la utilización de las librerías SysML por parte de los usuarios y la estrategia de transformación planteada, los elementos de estas librerías SysML puede ser modeladas como "caja negra", en las que únicamente se definen sus interfaces y algunos parámetros, sin tener necesidad de conocer su estructura interna (variables, ecuaciones, etc). Sin embargo, sus elementos equivalentes en las librerías Modelica deben estar completamente definidos, porque es necesario para que los modelos se puedan compilar y ejecutar.

Además, cabe destacar que, a pesar de que la metodología propuesta en esta investigación es aplicable al modelado y análisis de diversos tipos de sistemas de fabricación y procesos, en el estado actual de la investigación el desarrollo de las librerías se ha limitado inicialmente al modelado para la simulación de sistemas lineales de ensamblado (Linear Assembly System - LAS), y concretamente para el análisis de calidad en artefactos de dos dimensiones (2D). Sin embargo, aunque las librerías desarrolladas se limiten a estos supuestos, esta propuesta es igualmente válida para otros tipos de análisis, como el caso 3D o la simulación de procesos de mecanizado, tal y como se pretende abordar en trabajos futuros.

Con el fin de estructurar estas librerías y siguiendo el planteamiento que se estableció en las experiencias previas con Modelica (Capítulo 3), se decidió estructurarlas en tres (sub)librerías. Dos de ellas se corresponden con las ya establecidas en la experiencia previa, una básica orientada al análisis de la productividad, y una segunda (extensión de la primera) en la que incorpora el análisis de la calidad geométrica. Por su parte, la tercera es la que incorpora los conceptos vinculados con el modelo TTRS, no

considerados en la experiencia inicial. Por lo tanto, las librerías desarrolladas, tanto en SysML como en Modelica, son:

- *Linear Assembly System Simulation (LAS_Sim)*. Librería básica para la simulación de sistemas de ensamble lineales centrada en el análisis del flujo logístico y la productividad con un comportamiento discreto de los recursos. Se trata de la librería desarrollada en las experiencias iniciales con Modelica ([30], [26] y Capítulo 3), modificada y enriquecida con la aplicación de los perfiles desarrollados.

- *TTRS_concepts*. Librería para el modelado de piezas y ensambles mecánicos en base a los conceptos de TTRS, incluyendo tanto la definición de RGEs y MGREs como las relaciones establecidas entre estas entidades. La librería Modelica se presentó en trabajos previos como [27], en los que no se contemplaba la integración con SysML.

- *Linear Assembly System and Quality Simulation (LASQ_Sim)*. Los elementos de esta librería especializan los elementos de la librería LAS_sim para incorporar los elementos estructurales necesarios para simular la variabilidad geométrica. Con este fin, además de hacer uso de elementos de la librería TTRS_concepts, también se definen nuevos elementos que dan soporte a este tipo de análisis.

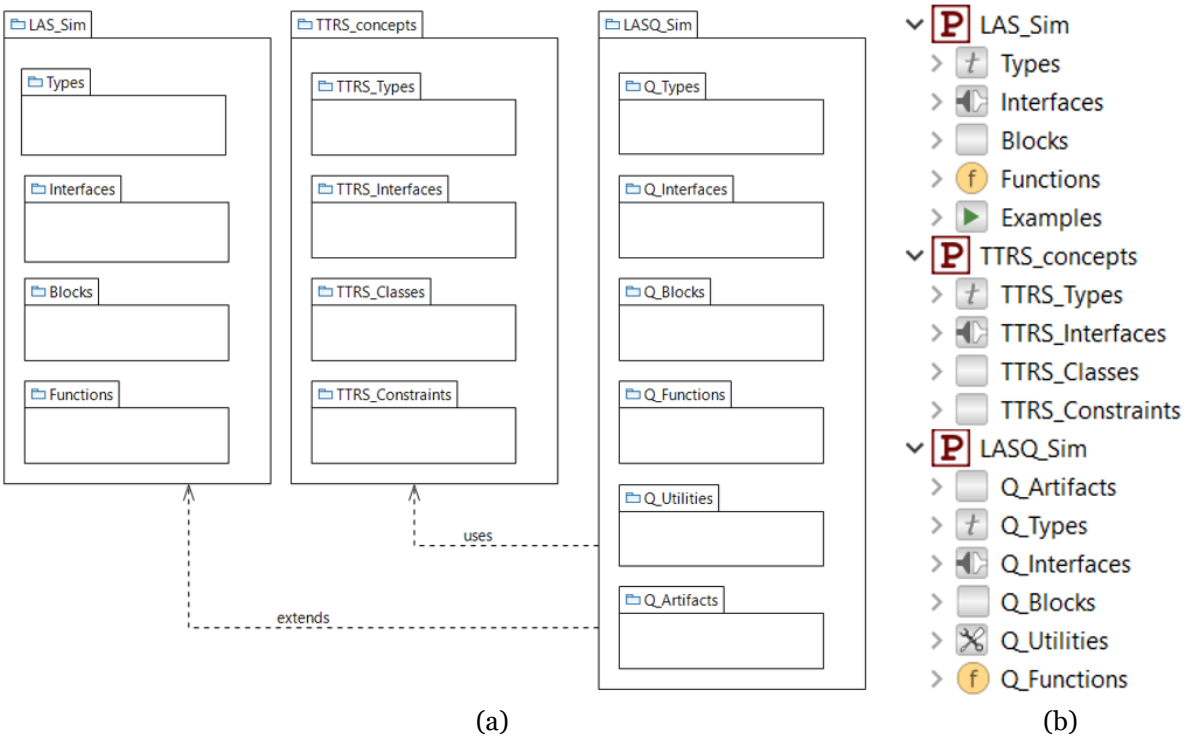


Figura 7.1. Paquetes de las librerías propuestas, implementadas en SysML (a) y Modelica (b).

La estructura de paquetes de cada una de las librerías propuestas, ilustrando tanto su modelado en SysML (diagrama de paquetes SysML) como en Modelica (árbol del explorador de librerías de OpenModelica), se muestra en la Figura 7.1. Siguiendo esta estructura de paquetes, el resto del capítulo se estructura en las correspondientes

secciones y apartados, en los que se comentan aspectos propios tanto del modelado en SysML como en Modelica.

7.1. Librería "Linear Assembly System Simulation" (LAS_Sim)

Como se representa en la Figura 7.1, esta librería se estructura en cuatro paquetes (tipos, interfaces, bloques y funciones), que se comentan a continuación.

7.1.1. Paquete "Types"

En este paquete se definen los tipos de datos manejados en la simulación. Como en este caso la simulación se limita al análisis de la productividad, estos tipos corresponden fundamentalmente a enumeraciones en las que se definen los estados posibles en los que puede estar un tipo de elemento, por ejemplo, una estación de trabajo, para la cual se define el tipo "A_status" como una enumeración que incluye los ítems: idle, measuring, pre_setting, setting, preparing, prepared, processing, blocked, repairing, maintaining. De esta manera, el bloque AssemblyWorkStation debe tener una propiedad de tipo A_status con el fin de representar de forma sencilla en qué estado se encuentra en cada instante. La Figura 7.2 representa la definición de este tipo como enumeración de SysML, estereotipada como «M_Type».

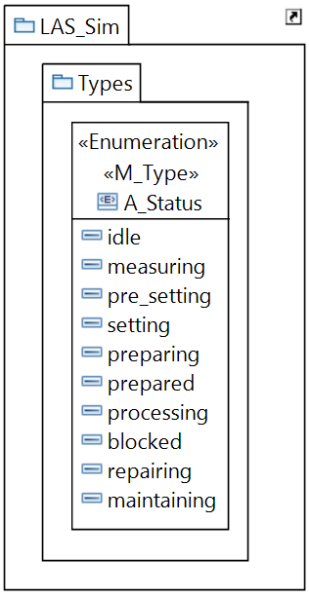


Figura 7.2. Definición de M_Types en SysML LAS_Sim.

Esta enumeración SysML se modela en Modelica como un "type", definido con el siguiente código.

```
package Types
  type A_Status = enumeration(idle, measuring, pre_setting, setting,
  preparing, prepared, processing, blocked, repairing, maintaining);
end Types;
```

7.1.2. Paquete “Interfaces”

Paquete que alberga los tipos de interfaces empleadas en la conexión de los diferentes tipos de componentes del sistema de simulación. Estas interfaces se modelan como puertos estereotipados como «M_Connector» en SysML, y como “connectors” en Modelica. Las principales interfaces consideradas son:

- *Assembly_Flow*. Da soporte a información sobre ensambles (como material) que se transmite de una estación a otra. Incluye propiedades que caracterizan el ensamble, como su identificador, las piezas que lo forman y su tipo, así como variables de tipo booleano con las que dar soporte a señales de comunicación bidireccional entre estaciones de trabajo. Esta comunicación permite coordinar el envío de las unidades de flujo, indicando cuándo una estación está lista para enviar el ensamble a la siguiente, y cuándo la siguiente estación está disponible para recibirlo, por lo que se estereotipan además como «FlowUnit».
- *Part_Flow*. Definición similar a la anterior, pero orientada en este caso a la caracterización de piezas (como material) (identificador, tipo, etc.), incluyendo las variables para la comunicación y sincronización de estaciones de trabajo. De nuevo, este tipo de puertos se estereotipa como «FlowUnit»
- *ControlData_Flow*. Da soporte al intercambio de datos entre el elemento que simula el sistema de control y los elementos que representan estaciones de trabajo. Este tipo de interfaz da soporte tanto a la monitorización de tiempos (tiempo procesando, bloqueado, reparando, midiendo, etc.) y estados de la estación, como a la comunicación de señales (variables booleanas) que el control lanza para activar determinadas acciones (reparar, ejecutar tarea de mantenimiento, etc.). Este tipo de puerto se estereotipa como «DataPort» en SysML.

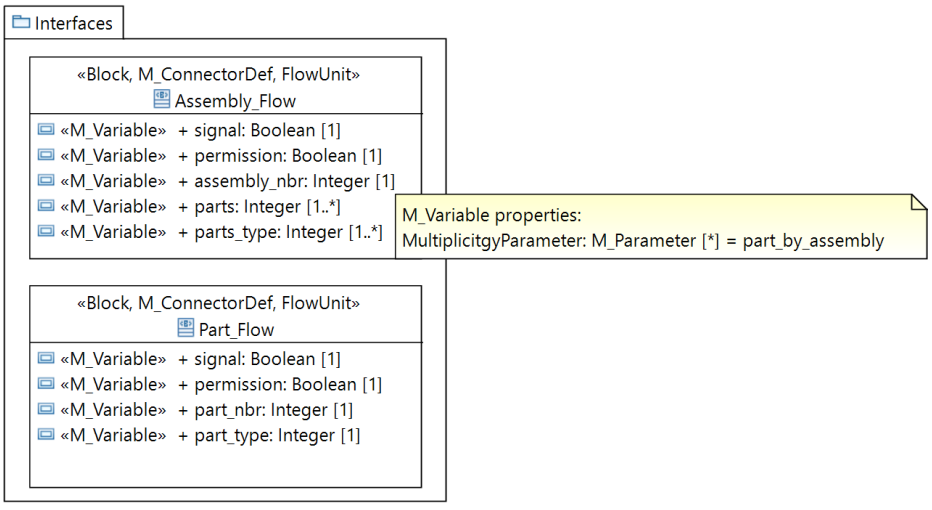


Figura 7.3. Puertos Assembly_Flow y Part_Flow.

- *EventPort (input y output)*. Interfaz simple de tipo booleano que sirve para transmitir únicamente una señal, generalmente vinculada a un evento. Este tipo de puerto también se estereotipa como «DataPort» en SysML.

A modo de ejemplo, la Figura 7.3 representa la definición de las interfaces puertos Assembly_Flow y Part_Flow como puertos en SysML.

Estos puertos SysML se implementan posteriormente en Modelica como "connector" definiendo el siguiente código.

```
package Interfaces
 import LAS_Sim.Types.*;
 connector Assembly_Flow
  discrete Boolean signal;
  discrete Boolean permission;
  discrete Integer assembly_nbr;
  discrete Integer parts[part_by_assembly];
  discrete Integer parts_type[part_by_assembly];
 end Assembly_Flow;
 connector Part_Flow
  discrete Boolean signal;
  discrete Boolean permission;
  discrete Integer part_nbr;
  discrete Integer part_type;
 end Part_Flow;
end Interfaces;
```

Estos ejemplos permiten comentar una de las grandes diferencias en el modelado SysML y Modelica, vinculada con el dimensionado de ciertas propiedades. En SysML es posible establecer la multiplicidad de un determinado atributo definiendo su valor o estableciendo ciertos intervalos, pero siempre mediante valores enteros (o * en el caso de valores indeterminados). En cambio, en Modelica es posible definir propiedades como vectores o matrices en los que el tamaño está determinado por otros parámetros del modelo. Para resolver esta diferencia, el perfil MMT2Modelica incluye un estereotipo abstracto llamado *«M_MultiplicityProperty»*, que se especializa en M_Variable, M_Parameter y M_Part para diferenciar el modelado de variables, parámetros y partes. Este estereotipo cuenta con una propiedad llamada "MultiplicityParameter", de tipo «M_Parameter», que apunta a una propiedad «M_Parameter» definida en el modelo, y cuyo valor se utiliza para definir el tamaño de la propiedad estereotipada con especializaciones de *«M_MultiplicityProperty»*. El valor de la propiedad MultiplicityParameter del estereotipo también puede ser nula, indicando que se considera válida la expresión de la multiplicidad como enteros. En el caso del Assembly_Flow expuesto anteriormente, las variables "signal", "permission" y "assembly_nbr" tienen vacio el campo MultiplicityParameter del estereotipo, indicando que se aplica la multiplicidad indicada (en este caso, de valor 1). En cambio, en las propiedades "parts" y "parts_type" el campo MultiplicityParameter del estereotipo apunta a una propiedad llamada part_by_assembly, estereotipada como «M_Parameter» y definida en otro elemento del modelo SysML (Data_Tables, descrito en el siguiente apartado). Estos aspectos relativos a la definición de propiedades de estereotipos aplicados no se pueden visualizar gráficamente en el diagrama SysML, y se muestra en la Figura 7.3 mediante un comentario que forma parte del diagrama.

7.1.3. Paquete “Blocks”

Se trata del paquete más importante desde la perspectiva del usuario, puesto que contiene los elementos principales para construir el modelo de simulación, si bien éstos están definidos empleando elementos de los otros paquetes. Son, por tanto, elementos definidos como bloques en SysML y como “models” en Modelica. De forma general, el usuario construye el modelo de simulación mediante usos de elementos de este paquete, estableciendo conexiones y dando valor a sus parámetros para definir un experimento concreto. Además, se definen algunos bloques abstractos (partial models en Modelica) para definir características estructurales comunes a varios elementos, como es el caso de la definición de puertos. La Figura 7.4 presenta un BDD con los principales bloques definidos en este paquete SysML, en el que se pueden observar las relaciones de especialización y la aplicación de estereotipos.

A continuación, se describen brevemente los principales elementos definidos en este paquete y se comentan algunas de sus principales características:

- *BaseAssemblyOnePort*. Bloque abstracto (“partial model” en Modelica) que cuenta con una interfaz (puerto en SysML y conector en Modelica) de tipo Assembly_Flow para la entrada/salida de datos referentes al flujo de ensambles.
- *BaseAssemblyTwoPorts*. Especialización de BaseAssemblyOnePort para incorporar una segunda interfaz de tipo Assembly_Flow.
- *BasePartOnePort*. Bloque abstracto (“partial model” en Modelica) que cuenta con una interfaz de tipo Part_Flow.
- *BasePartTwoPorts*. Especialización de BasePartOnePort para incorporar una segunda interfaz de tipo Part_Flow.
- *Part_Generator*. Especialización del BasePartOnePort encargada de generar los datos de cada pieza simulada y transmitirlos a una estación de trabajo a través de la interfaz de tipo Part_Flow. Únicamente necesita una interfaz debido a que estos bloques son el inicio del flujo de datos que representa el flujo de piezas.
- *Assembly_Generator*. Especialización del BaseAssemblyOnePort encargada de generar los datos de cada ensamble simulado y transmitirlos a una estación de trabajo a través de la interfaz de tipo Assembly_Flow. Únicamente necesita una interfaz debido a que estos bloques son el inicio del flujo de datos que representa el flujo de ensambles.
- *Part_Store*. Especialización del BasePartTwoPort para emular un almacén en el que se produce una espera (una cola) de piezas. Por ello, se modela como un elemento con dos interfaces, una para la recepción y otra para el envío de piezas hacia la siguiente estación.
- *Assembly_Store*. Especialización del BaseAssemblyTwoPort para emular un almacén en el que se produce una espera (una cola) de ensambles. Por ello, se modela como un elemento con dos interfaces, una para la recepción y otra para el envío de ensambles hacia la siguiente estación.
- *Assembly_Step*. Elemento que emula un recurso encargado de un proceso de ensamble. Para ello, especializa el BaseAssemblyTwoPorts, siendo un elemento

con dos interfaces, uno para la recepción y otro para el envío de ensambles hacia la siguiente estación.

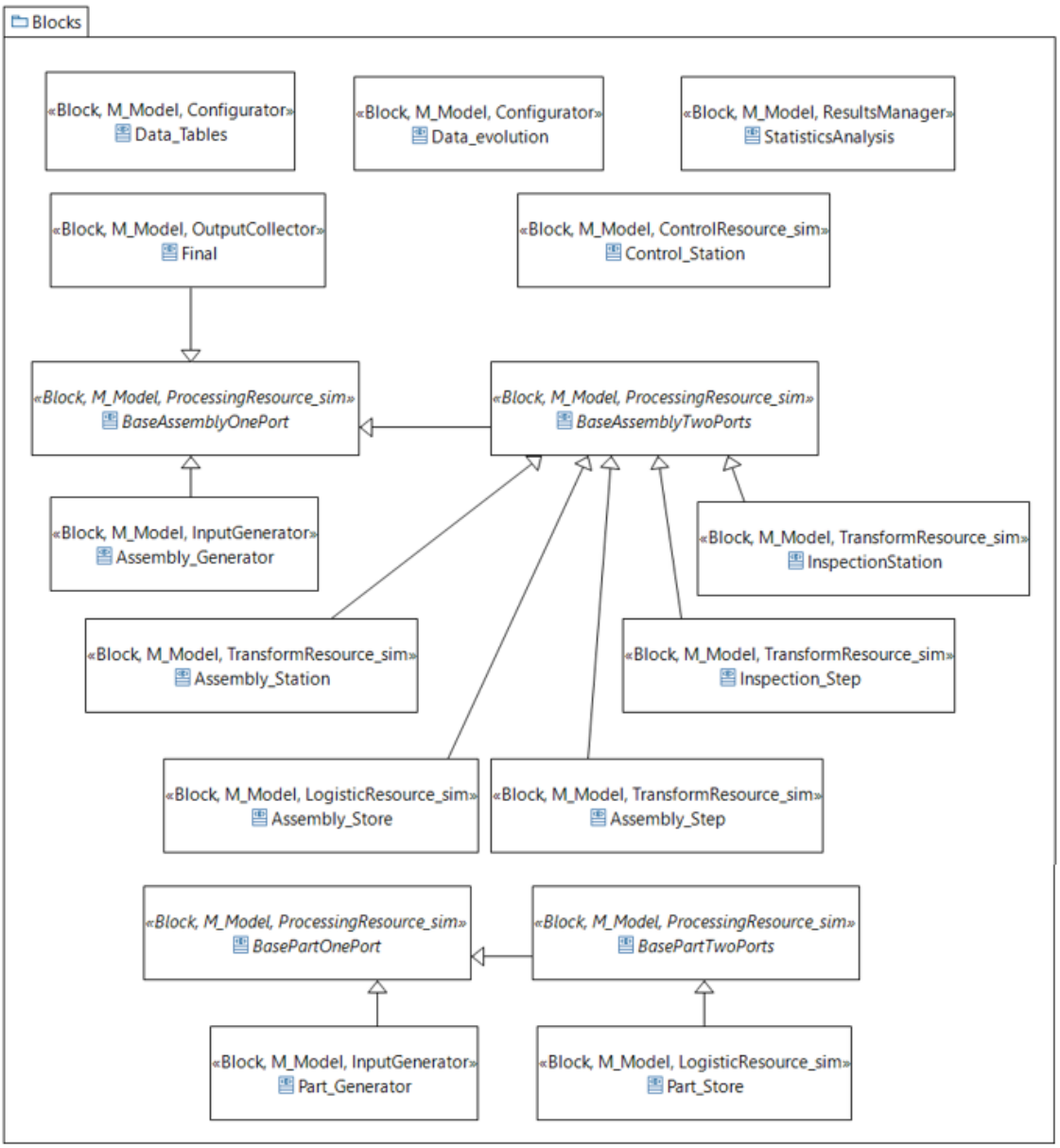


Figura 7.4. BDD del paquete Blocks de LAS_Sim.

- *Assembly_Station*. Elemento que emula una estación de ensamble completa, incluyendo un AssemblyStore y un PartStore para el almacenamiento de ensambles y piezas entrantes (productos en proceso), y el propio elemento de ensamble (AssemblyStep).
- *Inspection_Step*. Elemento que emula un recurso encargado de un proceso de inspección. Para ello, especializa el BaseAssemblyTwoPorts, siendo un bloque con dos puertos, uno para la recepción y otro para el envío de ensambles hacia la siguiente estación.
- *Inspection_Station*. Elemento que emula una estación de inspección completa, incluyendo un AssemblyStore para el almacenamiento de ensambles entrantes y el propio proceso de inspección (InspectionStep).

- *Data_tables*. Almacena los principales parámetros de la simulación y gestiona los datos para su utilización en los diferentes elementos del modelo, para lo cual hace uso de diversas funciones. Algunos de estos parámetros definen el comportamiento estocástico de algunas variables como por ejemplo el tiempo de proceso.
- *Data_evolution*. Especialización de Data_tables que permite incorporar algún tipo de comportamiento en las variables. Por ejemplo, se puede considerar que la media del tiempo de procesado se incrementa dentro de un turno de trabajo por el cansancio de los operarios.
- *Control_Station*. Representa el control central del sistema de fabricación simulado, soportando tanto la monitorización como la toma de decisiones. Para ello, cuenta con unos conectores de tipo ControlData_Flow, tantos como interacciones con otras estaciones de trabajo se consideren.
- *Statistic_Analysis*. Soporta el tratamiento de los datos de simulación para calcular los indicadores deseados, según los intereses del análisis.
- *Final*. Elemento que actúa como sumidero en el que termina un flujo logístico, fundamentalmente ensambles resultantes.

Con el fin de ejemplificar el modelado de estos elementos, a continuación, se presenta en detalle el elemento AssemblyStation, mostrando en primer lugar el modelado en SysML y, posteriormente, su implementación en Modelica. En la Figura 7.5 se visualiza un diagrama de definición de bloques (BDD) que contiene el bloque AssemblyStation como especialización del BaseAssemblyTwoPorts, lo que supone heredar dos puertos (inp_a y outp_a) que dan soporte a la entrada y salida de unidades de flujo. Además, el propio bloque AssemblyStation incorpora un tercer puerto (inp_c) que permite establecer conexiones con el sistema de control centralizado. Como se puede ver, más allá de los puertos, el bloque AssemblyStation está compuesto por los usos de cuatro bloques, que emulan respectivamente la generación de piezas a incorporar en el proceso de ensamble (Part_Generator), los almacenes, tanto para los ensambles entrantes como para piezas a incorporar (Assembly_Store y Part_Store), y la unidad procesadora en la que se ensambla el producto (Assembly_Step). Cada uno de estos bloques se define, a su vez, como especializaciones de otros bloques genéricos. Por último, es destacable la aplicación de estereotipos de los perfiles SysML4MSSim y MMT2Modelica. Por ejemplo, los estereotipos «TransformResource_sim» y «LogisticResource_sim» se aplican para identificar los bloques que emulan recursos transformadores (Assembly_Station, Assembly_Step) y logísticos (Part_Store, Assembly_Store) respectivamente, mientras que los puertos se estereotipan como «FU_Port» en el caso puertos de tipo Assembly_Flow o Part_Flow (bloques «FlowUnit») y como «DataPort» en el caso de los puertos de comunicación con el control. Por su parte, los estereotipos «M_Model» y «M_ConnectorDef» se aplican a bloques que definen modelos y conectores Modelica, mientras que el «M_ConnectorUse» se aplica sobre puertos que se transformarán en usos de conectores Modelica.

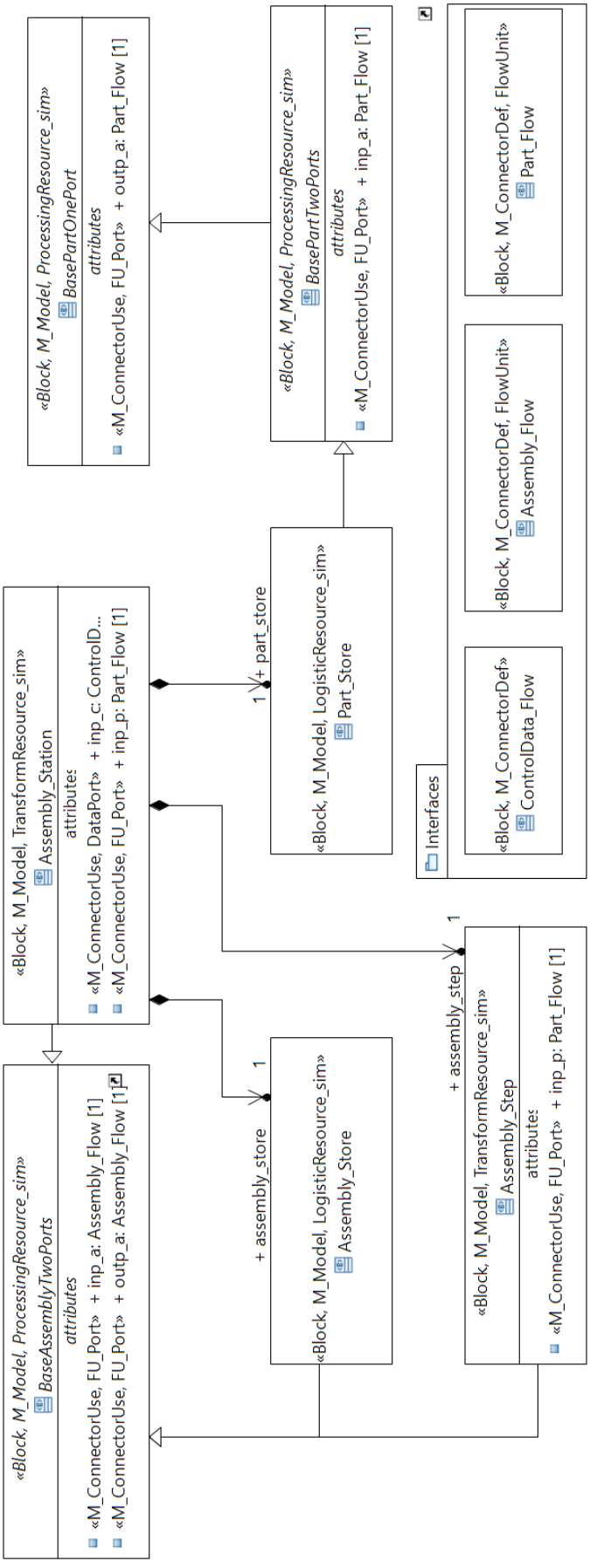


Figura 7.5. BDD de la definición del bloque Assembly_Station en SysML.

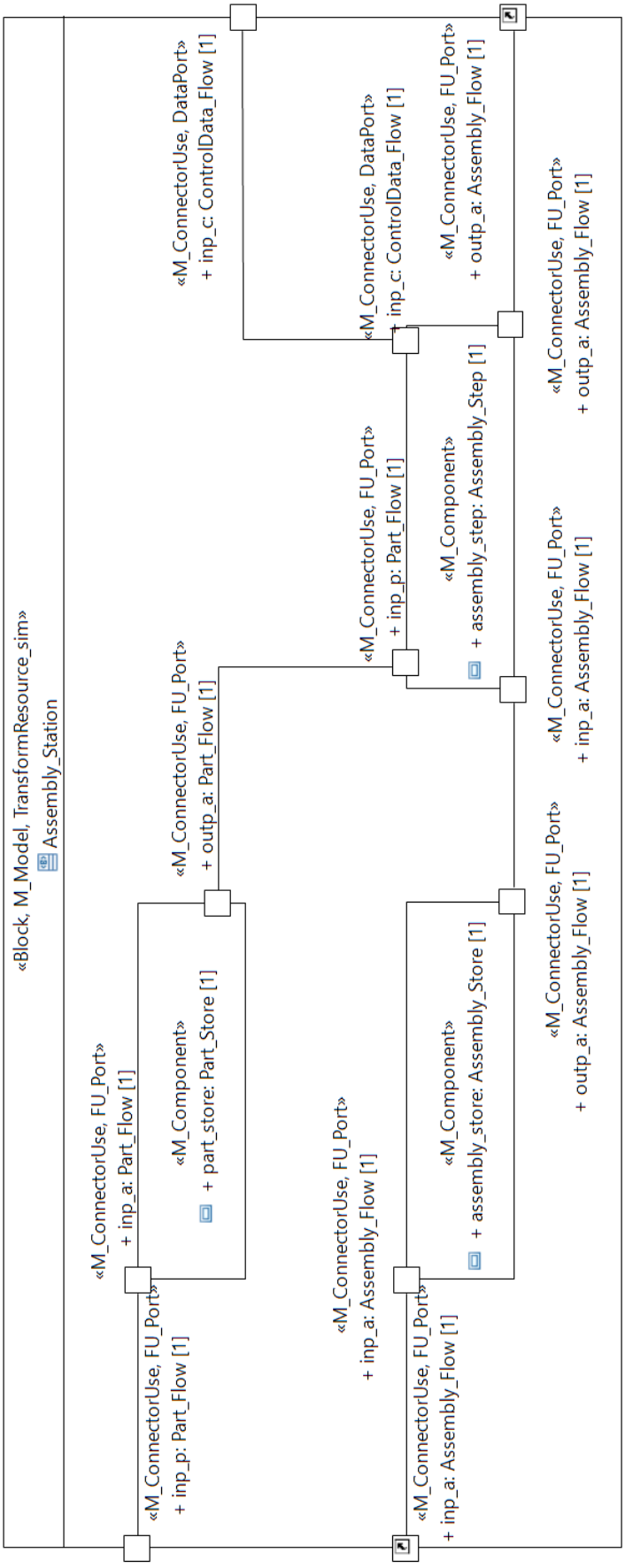


Figura 7.6. IBD de la definición del bloque AssemblyStation en SysML.

Además de la estructura representada en el BDD, también se deben definir las conexiones entre los diferentes componentes y puertos considerados en un diagrama interno de bloques (IBD), como el representado en la Figura 7.6, en el que se establecen los BindingConnectors que conectan los puertos propios del bloque contexto (AssemblyStation) con los puertos de cada una de las propiedades definidas, así como puertos de diferentes propiedades entre sí. Cabe recordar que estos conectores se estereotipan como «M_Connection», aunque no se represente este estereotipo en las figuras. Además, este diagrama permite ver la aplicación de los estereotipos en las diferentes propiedades, tanto en las partes («M_Connector») como en los puertos («M_ConnectorUse» y «DataPort»/«FU_Port», según el caso).

Además de los componentes y conexiones mostradas en las figuras anteriores, un elemento AssemblyStation requiere de numerosas variables para su simulación. Sin embargo, en el modelo SysML únicamente se han definido los parámetros que permiten al usuario definir las características concretas de cada instancia, mostrados en la Figura 7.7.

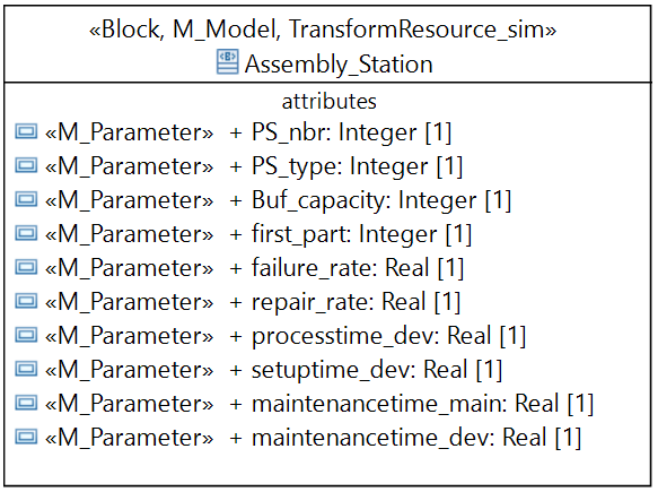


Figura 7.7. Propiedades «M_Parameter» del bloque AssemblyStation en SysML.

Comentada la definición de este bloque en SysML, es momento de abordar su implementación como elemento de la librería Modelica. Esta descripción se emplea, además, para ejemplificar y comentar brevemente algunos de los apartados típicos de un modelo Modelica, como son:

- Encabezado del modelo.

```
model AssemblyStation
```

- Extensión del BaseAssemblyTwoPort. En algunos modelos se incluye además la redefinición de algunas de las propiedades.

```
 extends BaseAssemblyTwoPort;
```

- Parámetros, en este caso, cuatro parámetros enteros y siete parámetros reales. Estos parámetros caracterizan la estación de trabajo con características concretas como el tipo o la capacidad de los almacenes, y otras empleadas para

caracterizar propiedades como los tiempos de procesos (de ensamble, de medición, de reparación, etc.) a través de un valor medio y/o su desviación típica.

```
parameter Integer PS_nbr = 1;
parameter Integer PS_type = 1;
parameter Integer Buf_capacity = 2;
parameter Integer first_part = 1;
parameter Real failure_rate, repair_rate, processtime_desv,
           meassuretime_desv, setuptime_desv, maintenancetime_mean,
           maintenancetime_desv;
```

- Variables. En este caso, se definen 10 variables, algunas de ellas con un comportamiento discreto, siendo cuatro de tipo Integer y seis de tipo Real. Estas variables se emplean para definir datos como el tiempo concreto de ejecución de cada operación del proceso, la ocupación del almacén en cada instante o el número de productos procesados, entre otros.

```
discrete Integer stored_assemblies;
discrete Integer stored_parts;
Real time_processing;
Real time_blocked;
Real time_idle;
Real time_repairing;
Real time_setting;
discrete Integer ass_in_process;
discrete Integer ass_processed;
Real service_time, repair_time, setup_time;
```

- Componentes y conectores (puertos), definidos mediante constructores en los que se crean instancias y se asignan valores a sus parámetros. En este caso, tanto el conector como los cuatro componentes considerados se definen como elementos reemplazables, es decir, que si se crea una extensión del modelo AssemblyStation estos elementos pueden ser sustituidos por especializaciones que incorporen otros datos o cálculos, como se verá más adelante en la definición de la librería LASQ_Sim.

```
replaceable LAS_Sim.Interfaces.Control_Station inp_c;
replaceable LAS_Sim.Blocks.Assembly_Store assembly_Store (capacity =
  Buf_capacity);
replaceable LAS_Sim.Blocks.Part_Store part_Store(capacity = Buf_capacity);
replaceable LAS_Sim.Blocks.Assembly_Step assembly_Step(AW_nbr = PS_nbr,
  AW_type = PS_type, failure_rate = failure_rate, repair_rate =
  repair_rate, processtime_desv = processtime_desv, meassuretime_desv =
  meassuretime_desv, maintenancetime_mean = maintenancetime_mean,
  maintenancetime_desv = maintenancetime_desv, setuptime_desv =
  setuptime_desv);
replaceable LAS_Sim.Blocks.Part_Generator part_Generator (AW_nbr = PS_nbr,
  AW_type = PS_type, first_part = first_part);
```

- Ecuaciones iniciales, en las que se establece el valor de algunas variables en el instante inicial. En este caso, no se han definido expresiones de este tipo.

- Ecuaciones, expresiones matemáticas que se validan a lo largo del tiempo simulado. En el caso de la AssemblyStation, se relacionan variables del modelo con variables propias de algunos de sus componentes. Dentro del apartado de

ecuaciones también se definen las conexiones entre las diversas interfaces, tanto del modelo como de sus componentes internos. Estas conexiones se implementan con la expresión connect().

```
equation
    stored_assemblies = assembly_Store.load;
    stored_parts = part_Store.load;
    time_processing = Assembly_Step.processing_time;
    time_blocked = Assembly_Step.blocked_time;
    time_idle = Assembly_Step.idle_time;
    time_repairing = Assembly_Step.repairing_time;
    time_setting = Assembly_Step.setting_time;
    ass_in_process = Assembly_Step.ass_in_process;
    ass_processed = Assembly_Step.processed;
    service_time = Assembly_Step.servTime;
    repair_time = Assembly_Step.repair_time;
    setup_time = Assembly_Step.setup_time;
    inp_c.ass_store_load = assembly_Store.m_load;
    inp_c.part_store_load = part_Store.m_load;

    connect(inp_a, assembly_Store.inp_a);
    connect(part_Generator.outp_p, part_Store.inp_p);
    connect(assembly_Store.outp_a, assembly_Step.inp_a);
    connect(part_Store.outp_p, assembly_Step.inp_p);
    connect(assembly_Step.outp_a, outp_a);
    connect(inp_c, assembly_Step.inp_c);
```

- Fin del modelo AssemblyStation.

```
end AssemblyStation;
```

Cabe comentar que los modelos textuales de Modelica incluyen además una gran cantidad de anotaciones que, entre otras cosas, sirven para definir la representación gráfica de los componentes, conexiones, iconos, etc., pero la descripción de este tipo de definiciones se ha obviado para simplificar el relato.

7.1.4. Paquete “Functions”

Este paquete contiene la definición de las diversas funciones Modelica utilizadas en los bloques anteriormente descritos. Las funciones de Modelica son básicamente construcciones que realizan cálculos (expresiones causales) expresados como algoritmos que calculan unos valores de salida (outputs) a partir de unos parámetros de entrada (inputs). El orden en el que se definen los inputs y outputs en la función indica, respectivamente, el orden en el que se deben aportar los datos de entrada en la llamada a dicha función, y el orden en que la función devolverá los datos de salida. Además, conviene recordar que las funciones no se instancian, sino que describen procedimientos que se ejecutan en determinados instantes en los que son llamados por algún modelo.

Como se ha comentado en el Capítulo 6, SysML no cuenta con construcciones específicas para dar soporte a la definición de funciones, tal y como son entendidas en Modelica. Por este motivo, se ha decidido modelar las funciones como

FunctionBehavior, caracterizadas por una expresión opaca que contiene la definición completa de la función, que contempla tanto sus entradas y salidas como el algoritmo. Al modelarse como expresiones opacas, opción que imposibilita la comprobación de su consistencia en SysML, este apartado se limitará a describir las características propias de la implementación en lenguaje Modelica.

La mayoría de las funciones definidas en este paquete dan soporte a las funciones propias del formalismo DEVS, adaptadas para dar soporte al comportamiento de diferentes bloques (Assembly_Generator, Assembly_Step, Assembly_Store, etc.). Otras funciones, en cambio, dan soporte a cálculos estadísticos que, por emplearse en repetidas ocasiones, se definen en una función para ser reutilizables. La Figura 7.8 representa mediante un BDD el paquete Functions modelado en SysML, que contempla diversas FunctionBehavior que pueden contener como expresiones opacas la implementación Modelica de la función.

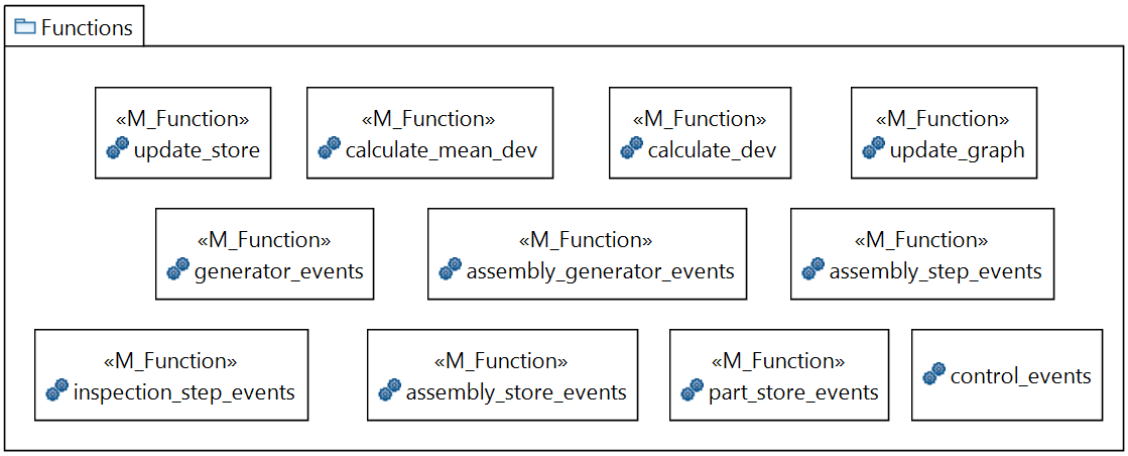


Figura 7.8. Elementos del paquete Functions de la librería LAS_Sim en SysML.

Sin embargo, para este tipo de elementos es muy útil describir su definición completa en Modelica. Tomando como ejemplo la implementación de la función "Calculate_mean_dev", que se encarga de calcular el valor medio y la desviación típica de un conjunto de valores, a continuación, se definen las principales partes del código Modelica de esta función.

- Encabezado de la función

```
function calculate_mean_dev
```

- Entradas (input), salidas (output) y variables. En este caso, las entradas son el conjunto de datos y la cantidad de datos, mientras que las salidas son el valor calculado de la media y la desviación típica. En caso de no querer restringir el tamaño de vectores o matrices se emplea el signo ":" para indicar que no está definido. En ocasiones, las funciones requieren de variables intermedias, en cuyo caso se definen en el apartado "protected", de manera que ningún otro elemento puede acceder a ellas ni se pueden visualizar en los resultados de la simulación, como es el caso de la variable real "ss" (sum of squares).

```
input Real data[:];
input Integer quantity;
output Real mean;
output Real dev;
protected
 Real ss;
```

- Algoritmo. En este caso, se calcula la media de las características (almacen), el sumatorio de los cuadrados de la diferencia de cada valor y la media (sc), y la raíz cuadrada de este resultado.

```
algorithm
 mean:= sum(data) / quantity;
 ss := sum((data[i] - mean) ^ 2 for i in 1:quantity);
 desv := sqrt(ss / quantity);
```

- Final de la función.

```
end calculate_mean_dev;
```

7.2. Librería “TTRS_concepts”

La librería TTRS_concepts ha sido diseñada y desarrollada para definir los elementos y relaciones básicas TTRS que dan soporte al modelado de geometrías nominales y desviadas. En el caso de las geometrías desviadas, sus valores pueden ser reales o simulados de acuerdo a las tolerancias especificadas. Para ello, no solo se definen elementos basados en conceptos como MGRE o RGE, sino que además se definen las relaciones matemáticas entre estos elementos, en función de los casos propuestos en TTRS según la combinación de clases de invarianza. Esta librería se estructura en cuatro paquetes que se describen a continuación, comentando con mayor detalle los casos de los MGRE linear y planar, así como las posibles restricciones establecidas entre ellos, unos tipos básicos que pueden servir de ejemplo para un mejor acercamiento a esta librería y sus contenidos.

7.2.1. Paquete “TTRS_Types”

Este paquete contiene la definición de los tipos complejos necesarios en el modelado TTRS, y que se modelan como DataTypes y Enumerations en SysML y que corresponden a “records” y “types” en Modelica respectivamente. Se crean tres DataTypes para definir los tres RGEs: punto, línea y plano. Los RGE_Line y RGE_Plane están definidos en ambos casos por un punto (contenido en la línea o en el plano) y un vector (director en el caso de la línea y normal en el caso del plano), mientras que el RGE_Point únicamente se define con su posición. Otro tipo definido es la enumeración Ref_types, que permite definir si una referencia es de "orientación" o "posición". La Figura 7.9 muestra un BDD con la definición de estos tipos en SysML.

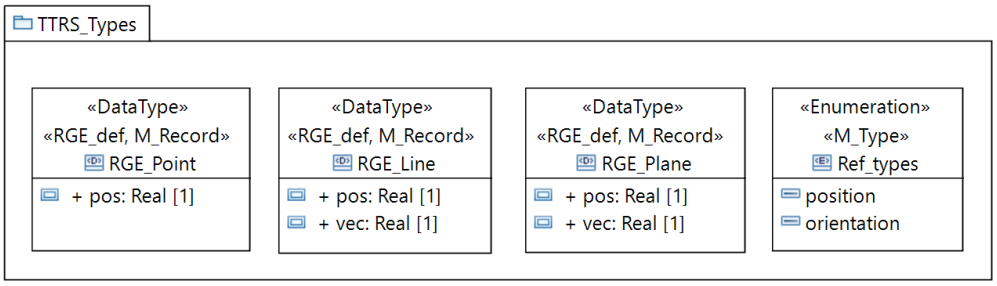


Figura 7.9. Elementos del paquete TTRS_Types de la librería TTRS_concepts definida en SysML.

Este paquete está definido en Modelica mediante el siguiente código.

```
package TTRS_Types
 type Ref_type = enumeration(position, orientation);
 record RGE_Point
    Real pos[3];
 end RGE_Point;
 record RGE_Line
   Real pos[3];
   Real vec[3];
 end RGE_Line;
 record RGE_Plane
   Real pos[3];
   Real vec[3];
 end RGE_Plane;
end TTRS_Types;
```

7.2.2. Paquete “TTRS_Interfaces”

En este paquete se definen los tipos utilizados para especificar puertos geométricos en SysML (o conectores en Modelica). Estos puertos están definidos básicamente haciendo uso de RGEs (definidos en el paquete anterior). Aunque se pueden crear puertos que contengan un único RGE para representar una geometría nominal o desviada (por ejemplo, los bloques N_RGE_Pl y D_RGE_Pl de la Figura 7.10), los puertos más utilizados habitualmente en el análisis están compuestos por dos RGE, uno para establecer la definición nominal de la geometría y otro para la desviada. Tomando como ejemplo los RGE planos, la Figura 7.10 muestra la forma en que se definen estos puertos en SysML.

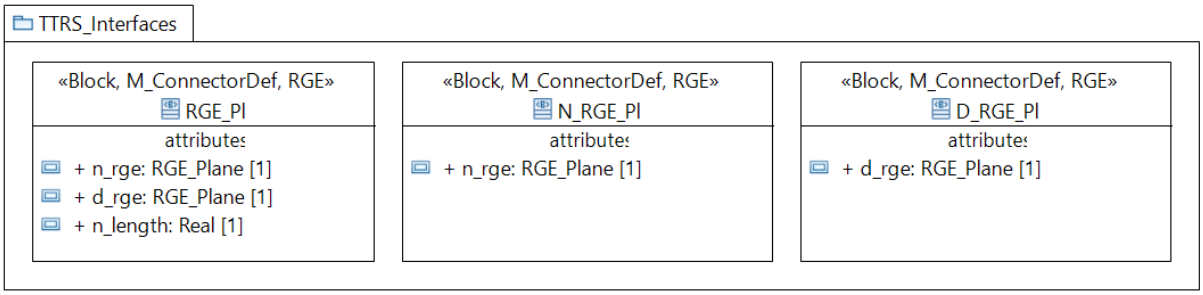


Figura 7.10. Definición de puertos de tipo RGE plano en el paquete TTRS_Interfaces de la librería TTRS_concepts.

Estos mismos elementos se expresan en la librería Modelica mediante el siguiente código.

```
package TTRS_Interfaces
  connector RGE_Pl
    RGE_Plane n_rge;
    Real n_length;
    RGE_Plane d_rge;
  end RGE_Pl;
  connector N_RGE_Pl
    RGE_Plane n_rge;
  end N_RGE_Pl;
  connector D_RGE_Pl
    RGE_Plane d_rge;
  end D_RGE_Pl;
end TTRS_Interfaces;
```

7.2.3. Paquete “TTRS_Constraints”

El paquete TTRS_Constraints contiene el modelado de las 13 restricciones básicas entre superficies que se pueden establecer atendiendo a sus clases de invarianza, es decir, según sus respectivos RGEs. En el estado actual de desarrollo de la librería, únicamente se han desarrollado las restricciones aplicables al caso 2D. Como ejemplo, se describe el modelado de la restricción C7, que establece la relación entre dos planos no paralelos y que, por tanto, forman un ángulo. Los dos RGEs relacionados por la C7 son de tipo plano y se representan mediante dos puertos de tipo RGE_Pl (tipo de puerto definido en el paquete TTRS_Interfaces). Además, otro dato característico de la restricción C7 es el ángulo que forman ambos planos. Dicho ángulo puede ser un parámetro impuesto o un ángulo calculado a partir de la orientación relativa de los planos nominales, y se utiliza como restricción entre las geometrías desviadas. Para diferenciar estos dos usos alternativos de la C7, se define el parámetro booleano “calculate”, mediante el que se indica si se desea calcular el ángulo a partir de los RGEs o se desea imponer un valor concreto. En la Figura 7.11 se presenta el modelado de esta restricción C7 de TTRS en SysML.

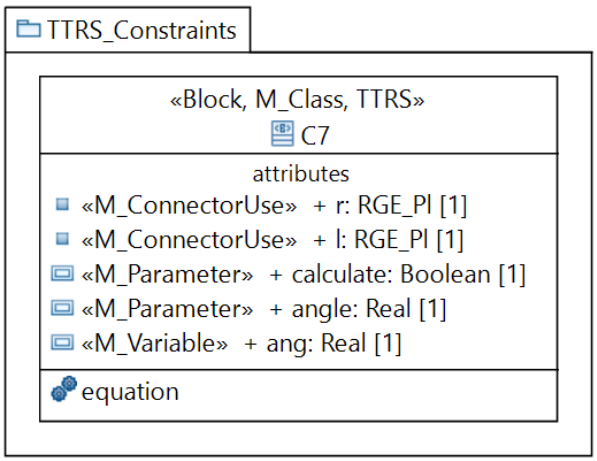


Figura 7.11. Modelado de la restricción C7 de TTRS en SysML.

Esta restricción C7 de TTRS se modela en Modelica mediante el código que se detalla a continuación, en el que se puede comprobar cómo el cálculo está condicionado al valor del booleano “calculate”, de manera que si es verdadero se calcula el valor del “ang” a partir de los RGE nominales y, en caso de ser falso, se emplea el valor definido en el parámetro “angle”.

```
model C7
  parameter Boolean calculate = true;
  parameter Real angle(start = 0);
  TTRS_concepts.TTRS_Interfaces.RGE_Pl r;
   TTRS_concepts.TTRS_Interfaces.RGE_Pl l;
  Real ang;
equation
 if calculate then
  cos(ang) = l.n_rge.vec * r.n_rge.vec;
 else
  ang = angle;
 end if;
 l.d_rge.vec * r.d_rge.vec = cos(ang);
end C7;
```

7.2.4. Paquete “TTRS_Classes”

En el paquete TTRS_Classes se definen clases que dan soporte a la definición tanto de MGREs como de zonas de tolerancia definidas en base a TTRS. Con el fin de ejemplificar estas clases, a continuación, se comenta un ejemplo de cada tipo.

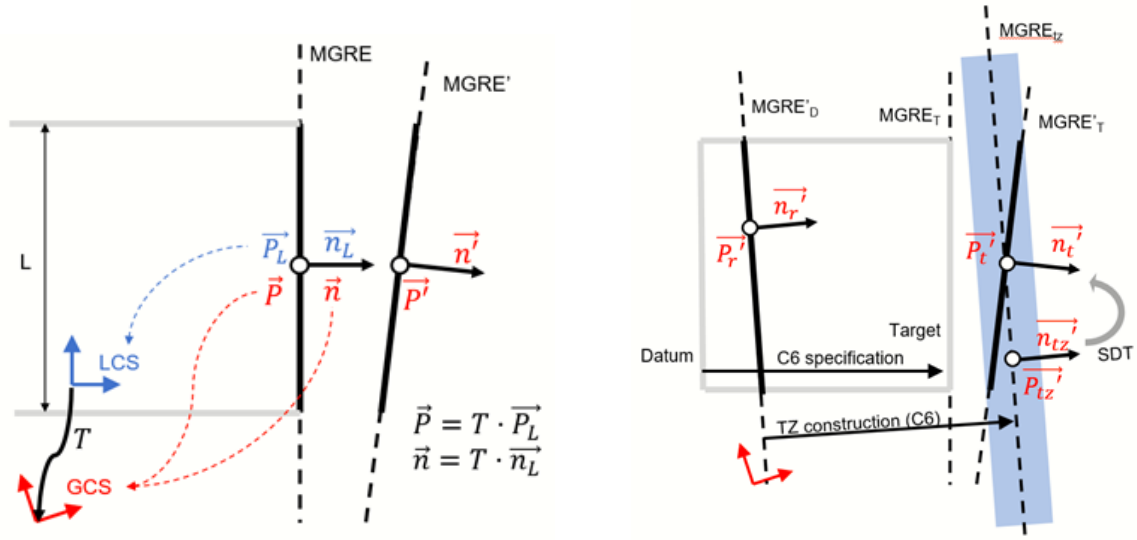


Figura 7.12. Definición nominal y desviada de un MGRE plano [27]

Por una parte, un MGRE se define en SysML como un bloque con puertos de tipo RGE (alguno del paquete TTRS_Interfaces), que definen los datos de la geometría nominal y desviada, todo ello expresado en coordenadas globales del ensamble para poder establecer relaciones de ensamble entre MGREs de diferentes artefactos. Un bloque MGRE también contiene propiedades parte con la definición de la geometría expresada en el sistema local de la pieza, una definición más propia de la especificación de piezas. Sin embargo, ambas propiedades (puertos y partes) deben estar relacionadas mediante la matriz de transformación correspondiente (característica de la pieza), que permite la transformación de los datos expresados en el sistema local de la pieza a datos expresados en un sistema global del ensamble. Cabe indicar, además, que los datos relativos a la geometría desviada están definidos en función de la zona de tolerancia establecida, generalmente construida a partir de otras geometrías (desviadas) empleadas como referencias. Para ilustrar este tipo de elementos y relaciones, la Figura

7.12 representa el caso de un MGRE plano (con su definición nominal y desviada) y su relación con la zona de tolerancia plana definida a partir de un datum plano.

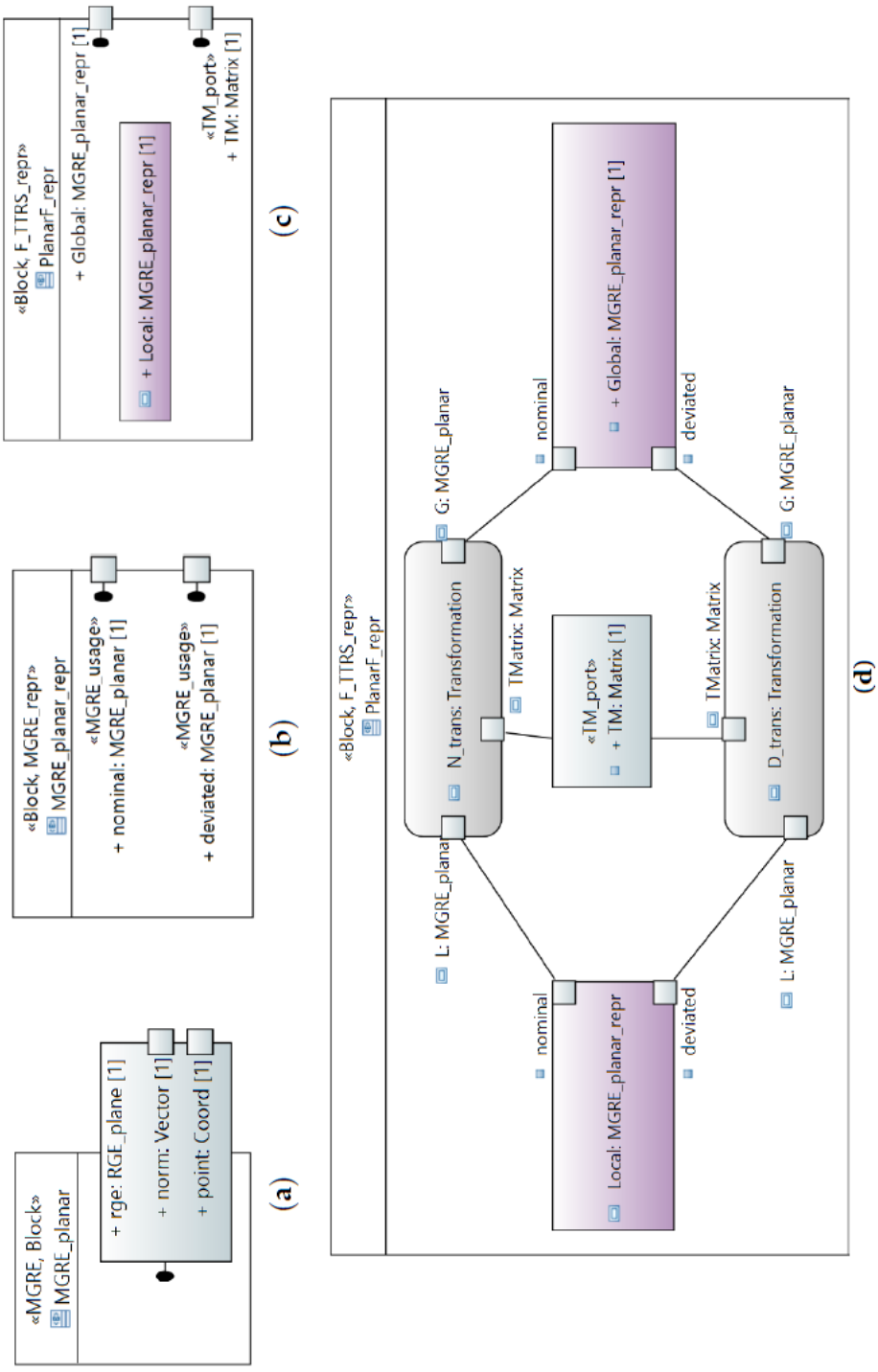


Figura 7.13. Representación TTRS de un feature plano: (a) IBD del bloque MGRE_planar; (b) IBD del bloque MGRE_planar_repr; (c) IBD del bloque PlanarF_repr; (d) PAR del bloque PlanarF_repr. [31]

A modo de ejemplo, la Figura 7.13 muestra varios diagramas referidos a la definición del bloque PlanarF_Repr, representación basada en TTRS de un feature plano. Como se muestra en el IBD de la Figura 7.13-C, un PlanarF_Rep posee dos propiedades de tipo MGRE_planar_repr (definición local y global) y un puerto "TM_port" en el que se

obtienen los datos de la matriz de transformación necesaria para pasar de un sistema de referencia a otro, tal y como muestra en el diagrama PAR (Figura 7.13-D). Por su parte, un MGRE_planar_rep está formado por dos propiedades de tipo MGRE_Planar con los datos de la geometría nominal y desviada (Figura 7.13-B). Finalmente, el IBD en la Figura 7.13-A muestra la definición de un MGRE_Planar, compuesto por un puerto de comportamiento de tipo RGE_Plane, que posee dos puertos anidados correspondientes al punto y al vector normal al plano. Para un mayor detalle sobre estas construcciones en SysML se propone la consulta de [31].

Por otra parte, la definición de zonas de tolerancia en base a TTRS ha sido objeto de estudio a lo largo de esta investigación, desarrollando diversas librerías con diferente nivel de detalle. Una de las primeras propuestas se comenta en [31], un trabajo en el que además de presentar el perfil SysML4TA también se muestra el desarrollo de librerías SysML, incluyendo elementos reutilizables para el modelado de zonas de tolerancia. Sin embargo, atendiendo a la estrategia de modelado y transformaciones propuesta, y diferenciando entre los elementos que el usuario debe ser capaz de utilizar y editar, frente a aquellos otros elementos protegidos a los que el usuario no debe tener acceso, se definió una nueva versión de esta librería. En esta segunda versión de la librería SysML se limita el contenido y se caracteriza cada elemento únicamente con los componentes y parámetros que deben ser accesibles y editables para el modelador.

Para ejemplificar estas cuestiones, a continuación, se presenta el modelado del bloque TPlana_1rP_P, que representa una zona de tolerancia plana definida a partir de un plano de referencia, por ejemplo, para una especificación de paralelismo. El BDD de la Figura 7.14 muestra únicamente las propiedades estereotipadas como «M_Parameter» de este bloque, mientras que en el IBD de la Figura 7.15, se representan sus partes, puertos y conexiones internas.

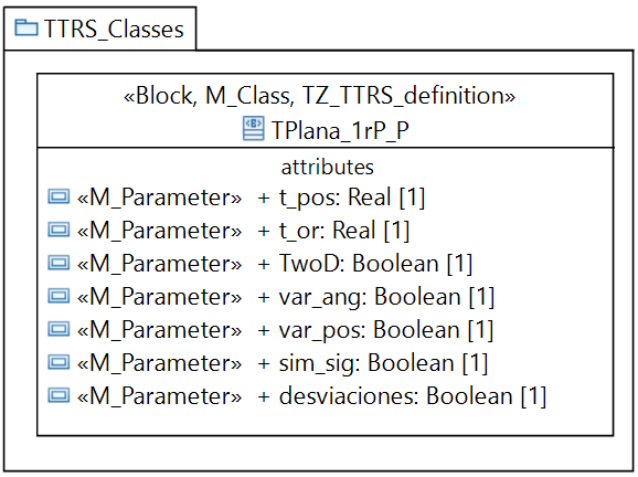


Figura 7.14. BDD del bloque TPlana_1rP_P en SysML.

Como se puede observar en la Figura 7.15, esta zona de tolerancia define dos puertos de tipo RGE plano (RGE_Pl), uno como referencia (ref) y otro como target (tar). A partir de la referencia, se calcula la posición y desviación del target mediante restricciones como “pos:C6_” y “calc_dev:C6_dev”, almacenando el resultado de cálculos intermedios en otros componentes del bloque contexto.

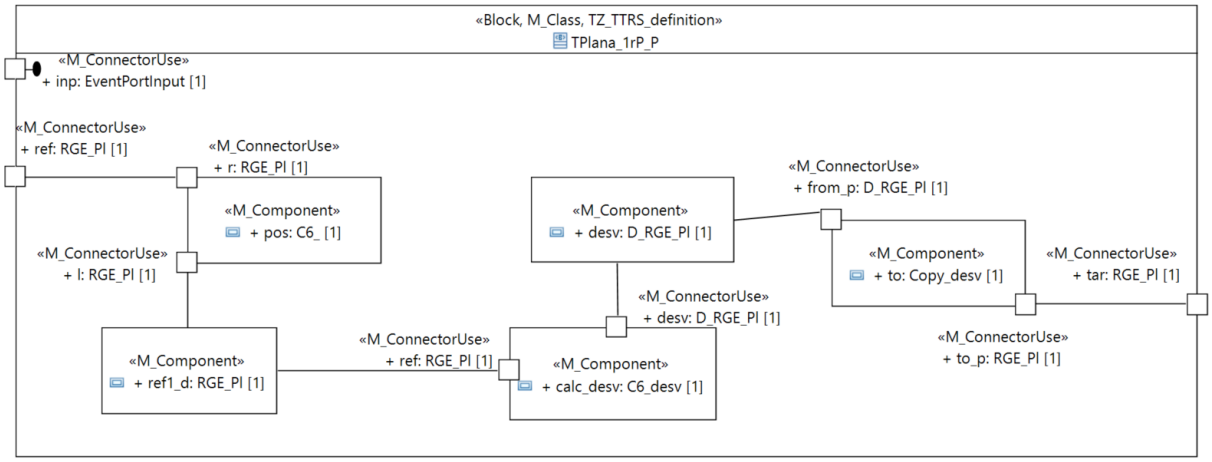


Figura 7.15. IBD del bloque TPlana_1rP_P en SysML.

A partir de este modelado en SysML, a continuación, se describe el modelado del elemento TPlana_1rP_P en Modelica. Dado que es una clase de cierta complejidad, a continuación, se presenta de forma fragmentada, diferenciando algunas de sus partes principales.

- Encabezado del modelo TPlana_1rP_P y definición de parámetros, tanto referidos a las tolerancias establecidas como al tipo de análisis simulado. Además, también se definen algunas variables empleadas en los cálculos del modelo.

```
model TPlana_1rP_P
    parameter Real t_pos = 0;
    parameter Real t_or =;
    parameter Boolean TwoD = true;
    parameter Boolean var_ang = true;
    parameter Boolean var_pos = true;
    parameter Boolean sim_sig = false
    Real new_tor(start = 0) if sim_sig;
    Real new_tpos(start = 0) if sim_sig;
    parameter Boolean desviaciones = false;
    Real ada if not desviaciones;
    Real add if not desviaciones;
    inner Real da, dd;
    Real dist_n, lambd;
```

- Definición de componentes y conectores. En este caso, se definen usos del RGE_Pl tanto para modelar los planos de referencia (ref) y target (tar), así como otros componentes (dev y ref1_d) necesarios para dar soporte a la restricción C6 (pos, calc_dev, to). También se define un conector para recibir el evento que desencadena el cálculo (inp).

```
    TTRS_concepts.TTRS_Interfaces.RGE_Pl ref;
    TTRS_concepts.TTRS_Interfaces.RGE_Pl tar;
    TTRS_concepts.TTRS_Interfaces.RGE_Pl ref1_d;
    TTRS_concepts.TTRS_Constraints.C6_ pos(calcular = false);
    TTRS_concepts.TTRS_Constraints.C6_dev calc_desv;
    TTRS_concepts.TTRS_Constraints.Copy_dev to(
    typep = TTRS_concepts.TTRS_Constraints.Copy_dev.ref_p.Plane);
    TTRS_concepts.TTRS_Interfaces.Desv_Pl dev;
```

```
    TTRS_concepts.TTRS_Types.EventPortInput inp if sim_sig;
    Random.Seed seed(start = {23, 87, 187}) if sim_sig;
```

- Definición de ecuaciones, que realizan los cálculos necesarios para obtener la desviación de la orientación y la posición del target en función de los parámetros establecidos.

```
equation
    if sim_sig then
        when inp then
             (seed, new_tor, new_tpos) = generate(pre(seed), atan(t_or /
             tar.n_length) / 3, (t_pos - t_or) / 6);
        end when;
        da = new_tor;
        dd = new_tpos;
        if not desviaciones then
             ada = da;
             add = dd;
        end if;
    else
        if not desviaciones then
             tan(ada) = t_or / tar.n_length;
             2 * add = t_pos - t_or;
             da = if var_ang then ada else -ada;
             dd = if var_pos then add else -add;
        end if;
    end if;

    (tar.n_rge.pos - ref.n_rge.pos) * ref.n_rge.vec = dist_n;
    ref1_d.n_rge = tar.n_rge;
    ref1_d.n_length = tar.n_length;
    norm(ref1_d.d_rge.vec) = 1;
    if TwoD then
      {0, 0, 1} * ref1_d.d_rge.vec = 0;
    end if;
    ref1_d.d_rge.pos = tar.n_rge.pos + lambd * tar.n_rge.vec;
    pos.dist = dist_n;
```

- Definición de conexiones entre los diferentes componentes y conectores definidos, y fin de la definición del modelo.

```
    connect(ref, pos.r);
    connect(pos.l, ref1_d) ;
    connect(ref1_d, calc_desv.ref);
    connect(calc_desv.desv, desv);
    connect(desv, to.from_p);
    connect(to.to_p, tar);
end TPlana_1rP_P;
```

7.3. Librería “Linear Assembly System and Quality Simulation” (LASQ_Sim)

La librería que se presenta en esta sección contiene especializaciones de las construcciones definidas en la librería LAS_Sim (sección 7.1), incorporando todos aquellos datos y cálculos necesarios para el análisis de la calidad geométrica en

procesos multietapa, para lo cual se hace uso de elementos definidos en la librería TTRS_concepts (sección 7.2). Por lo tanto, la librería LASQ_Sim se emplea en la construcción de modelos de simulación que, además del análisis de la productividad y la simulación del flujo de materiales, también permiten el análisis de la calidad geométrica basado en el modelado TTRS de determinados artefactos, principalmente ensambles de proceso que representan amarres de productos y utillajes. Cabe indicar que la letra Q, referida a Quality, se emplea en los nombres de los elementos definidos en esta librería como elemento diferenciador frente a términos similares de la librería LAS_Sim.

De forma análoga a la estructura de la librería LAS_Sim, la librería LASQ_Sim cuenta con cuatro (sub)paquetes dedicados a definir los nuevos tipos, interfaces, bloques y funciones. A modo de síntesis, los nuevos bloques se definen como especializaciones de los bloques de LAS_Sim en las que se redefinen algunas de sus propiedades, entre ellas los puertos (empleando nuevas interfaces que incluyen datos de calidad), y se añaden nuevos componentes, como es el caso de los modelos de artefactos para el análisis de la calidad. Para ello, a estos cuatro paquetes básicos se les suman además otros dos: Q_Artifacts, que contiene los elementos para el modelado en base a TTRS de artefactos del ámbito de la fabricación; y Q_Utilities, que define elementos para el tratamiento de los datos geométricos y su propagación por el modelo de simulación.

7.3.1. Paquete “Q_Types”

Paquete dedicado a la definición de dos nuevos tipos de datos que dan soporte a la definición de posiciones y de desviaciones respectivamente, y que se emplean, entre otras cosas, en la definición de los puertos descritos en el apartado siguiente. La Figura 7.16 presenta la definición de estos DataType en SysML.

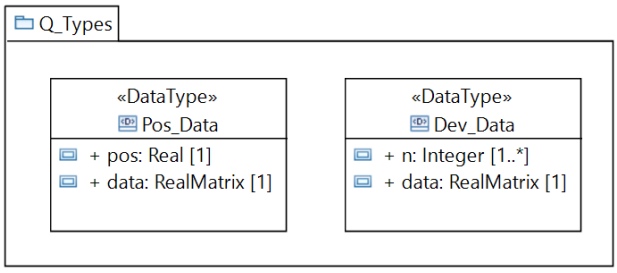


Figura 7.16. Contenido del paquete Q_Types de LASQ_Sim.

Estos tipos de datos se definen en Modelica como records mediante el siguiente código.

```
package Q_Types
  record Desv_Data
    Integer n[max_parts];
    Real data[n_desv_max, 2];
  end Desv_Data;
  record Pos_Data
    Integer pos;
    Real data[max_parts, 3];
  end Pos_Data;
end Q_Types;
```

7.3.2. Paquete “Q_Interfaces”

Este paquete incluye las especializaciones de las clases definidas en el paquete Interfaces de LAS_Sim, incorporando los tipos de datos que dan soporte a la información sobre las posiciones y desviaciones que se transmite de unas estaciones de trabajo a otras. La Figura 7.17 representa gráficamente la definición de estos puertos en SysML, mostrando las propiedades de los bloques base y las propiedades añadidas en cada una de las especializaciones.

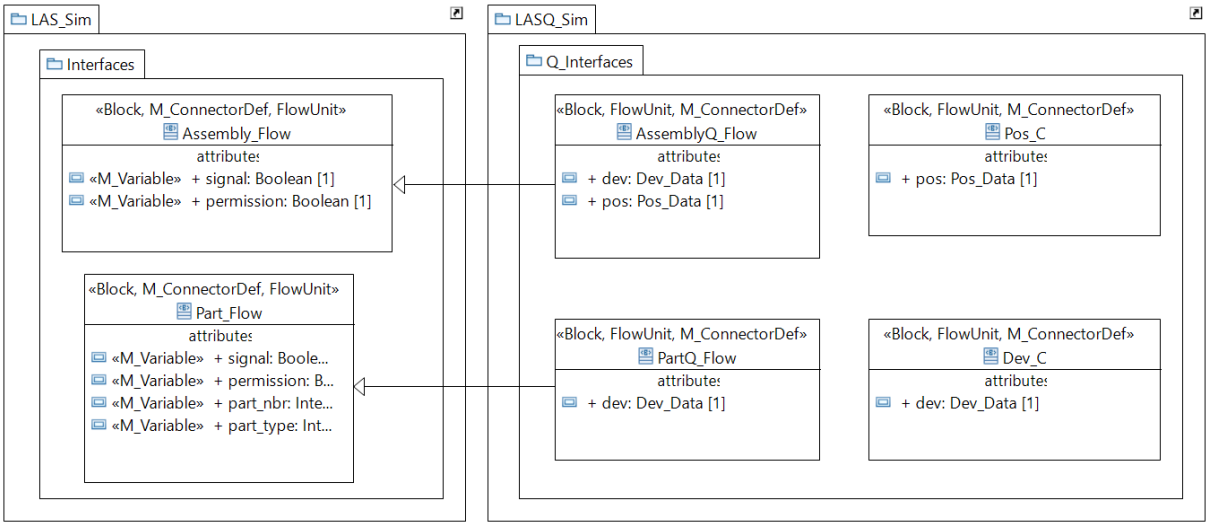


Figura 7.17. Elementos del paquete Q_Interfaces de LASQ_Sim en SysML.

Estas nuevas interfaces se definen como conectores Modelica mediante el siguiente código.

```
package Q_Interfaces
 connector AssemblyQ_Flow
   extends LAS_Sim.Interfaces.Assembly_Flow;
   Types.Dev_Data dev;
   Types.Pos_Data pos;
 end AssemblyQ_Flow;
 connector PartQ_Flow
   extends LAS_Sim.Interfaces.Part_Flow;
   Types.Dev_Data dev;
 end PartQ_Flow;
 connector Dev_C = Types.Dev_Data;
 connector Pos_C = Types.Pos_Data;
end Q_Interfaces;
```

7.3.3. Paquete “Q_Blocks”

Paquete en el que se definen diversas especializaciones de las clases descritas en el paquete Blocks de LAS_Sim, incorporando los datos y cálculos necesarios para soportar el flujo de las desviaciones geométricas de los artefactos simulados. Estas especializaciones corresponden a los generadores de piezas y ensambles (Part_Generator_Q y Assembly_Generator_Q), a los almacenes de piezas y ensambles

(Part_Store_Q y Assembly_Generator_Q) y a las estaciones de ensamble (Assembly_Step_Q y Assembly_Station_Q). En todas las especializaciones se redefinen los puertos de las clases del LAS_Sim por los tipos de puerto comentados en el paquete Q_Interfaces. Además, algunos bloques también aplican el mecanismo de redefinición para reemplazar otras propiedades, como es el caso de la Assembly_Station_Q, cuya propiedad assembly_Step (de tipo Assembly_Step) pasa a ser de tipo Assembly_Step_Q (bloque definido en este paquete). La Figura 7.18 muestra un BDD con los bloques definidos en este paquete, representando también las relaciones de especialización con bloques de LAS_Sim.

Además del mecanismo de redefinición de propiedades, empleado en todas estas especializaciones, la clase Assembly_Step_Q incorpora mayores diferencias respecto a su generalización Assembly_Step de la librería LAS_Sim, motivo por el cual a continuación se centra el apartado en describir la definición de esta clase.

La clase Assembly_Step_Q incorpora diversas propiedades adicionales respecto al Assembly_Step. Una de las más relevantes es la propiedad "process_assembly", de tipo ProcessAssembly, un bloque que define el ensamble de proceso característico de dicha etapa y permite calcular las desviaciones y posiciones del producto resultante en base a los artefactos entrantes y a los utillajes empleados en el amarre, como se detalla en la descripción del paquete Q_Artifacts. Esta propiedad se define como reemplazable, ya que a la hora de definir el modelo de simulación, cada uso del Assembly_Step_Q debe emplear la definición de un process_assembly diferente (especializaciones de Process_Assembly).

Adicionalmente, Assembly_Step_Q incorpora otras propiedades para el tratamiento de los datos geométricos de entrada y salida, dando soporte a la propagación de los mismos. Estas propiedades están tipeadas por bloques del paquete Q_Utilities, que será descrito más adelante. La Figura 7.19 muestra un BDD con la definición del Assembly_Step_Q, incluyendo la redefinición de algunas propiedades y la definición de otras adicionales. Por su parte, la Figura 7.20 presenta un IBD de este mismo bloque, comentando a continuación el significado y funcionalidad de cada componente.

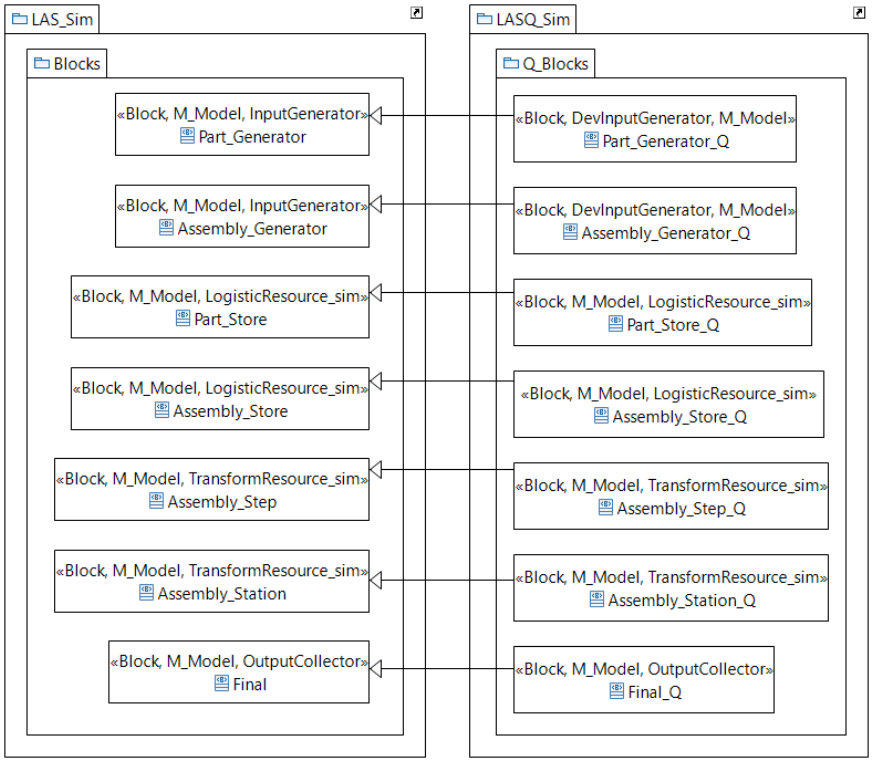


Figura 7.18. BDD bloques del paquete Q_Blocks, de la librería LASQ_Sim.

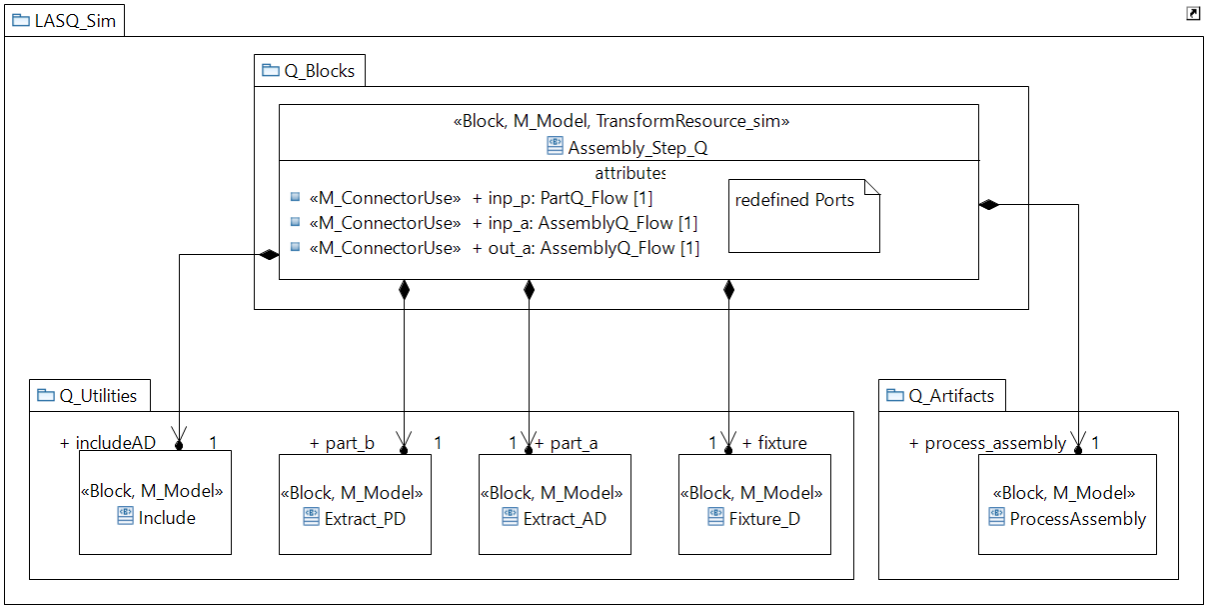


Figura 7.19. BDD del bloque Assembly_Step_Q.

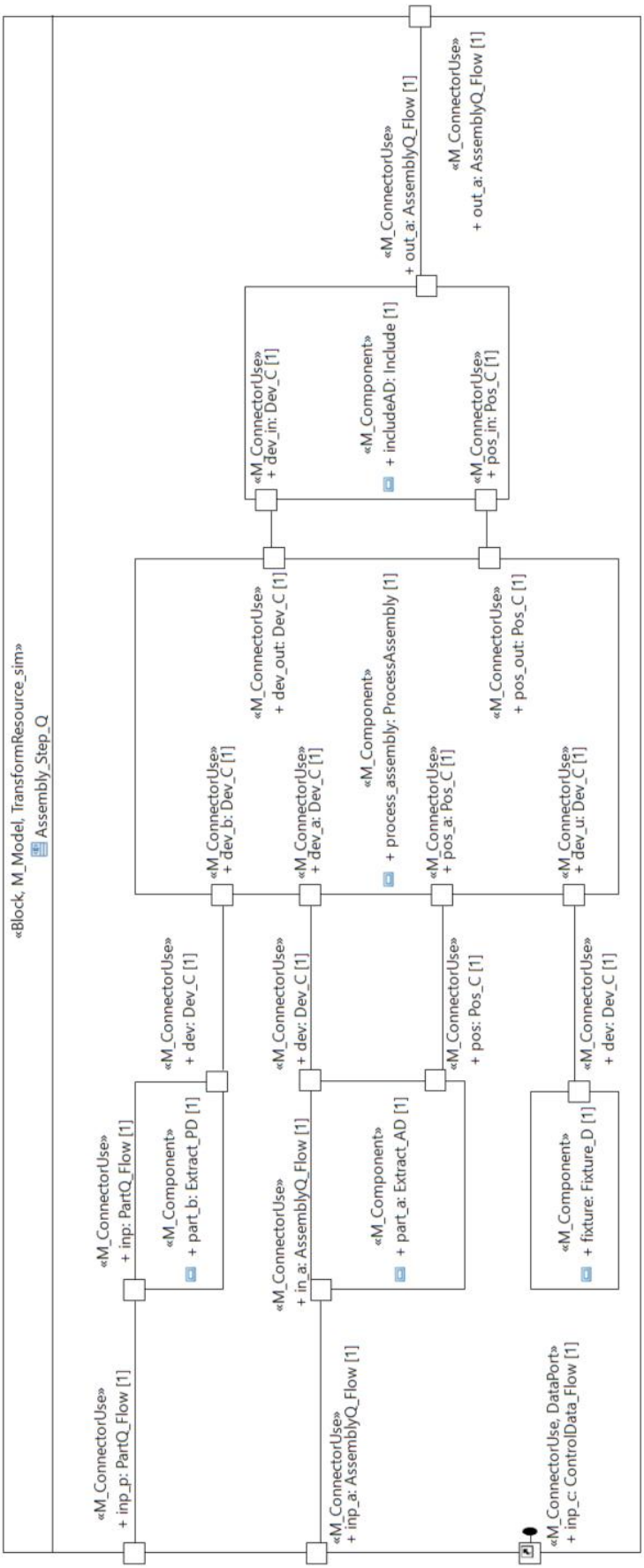


Figura 7.20. IBD del bloque Assembly_Step_Q.

El modelado de este elemento en Modelica se presenta a continuación, un código en el que se ejemplifican tanto los mecanismos de extensión y redeclaración de componentes, como la definición de componentes reemplazables. Además, también se definen las nuevas propiedades y conexiones del bloque.

```
model Assembly_Step_Q
 extends LAS_Sim.Blocks.Assembly_Step(
   redeclare LASQ_Sim.Q_Interfaces.Part_G inp_p,
   redeclare LASQ_Sim.Q_Interfaces.AssemblyQ_Flow inp_a,
   redeclare LASQ_Sim.Q_Interfaces.AssemblyQ_Flow outp_a);

 Q_Utilities.Extract_PD part_b;
 Q_Utilities.Extract_AD part_a;
 Q_Utilities.Fixture_D fixture;
 replaceable Q_Artifacts.ProcessAssembly process_assembly;
 Q_Utilities.IncludeAD includeAD;

equation
 connect(inp_p, part_b.inp);
 connect(inp_a, part_a.ina);
 connect(part_b.dev, process_assembly.dev_b);
 connect(part_a.pos, process_assembly.pos_a);
 connect(part_a.dev, process_assembly.dev_a);
 connect(fixture.dev, process_assembly.dev_u);
 connect(includeAD.outp_a, outp_a);
 connect(process_assembly.pos_out, includeAD.pos_in);
 connect(process_assembly.dev_out, includeAD.dev_in);
end Assembly_Step_Q;
```

Además de las mencionadas especializaciones, este paquete también incluye el bloque LASQ_SimulationModel, que define los elementos básicos de un modelo de simulación, de manera que el usuario puede utilizar esta definición como base sobre la que definir su propio modelo de simulación, redeclarando algunas propiedades o añadiendo otras. La estructura del bloque LASQ_SimulationModel se presenta en el BDD de las Figura 7.21 y el IBD de la Figura 7.22, comentando a continuación algunas de sus principales propiedades.

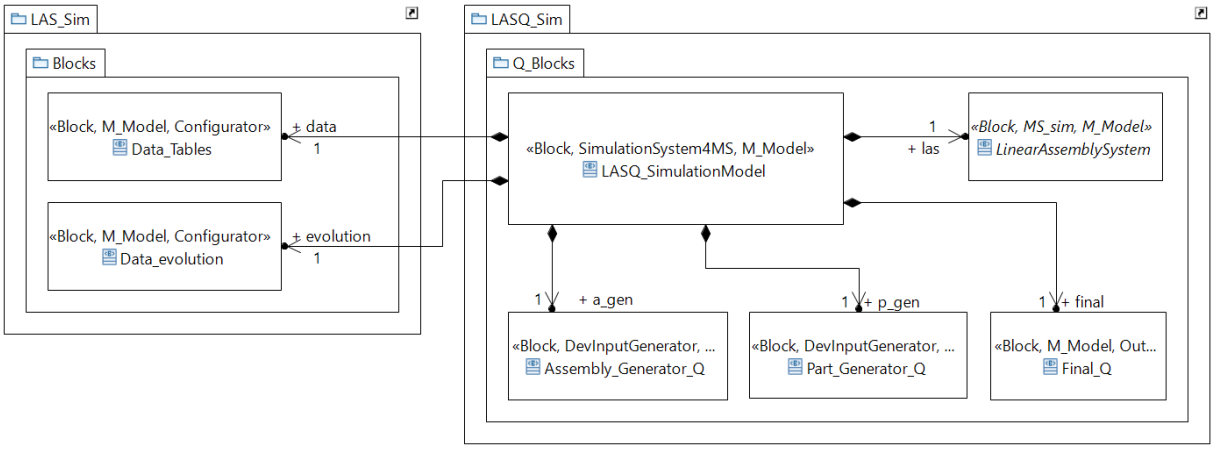


Figura 7.21. BDD del bloque LASQ_SimulationModel.

Como se observa en los diagramas, el LASQ_SimulationModel está estereotipado como un «SimulationSystem4MS», que indica que se trata de un modelo de simulación

de un sistema de fabricación, y como «M_Model», que señala que es susceptible de transformarse en un modelo Modelica. Este bloque está formado por diversas partes (estereotipadas como «M_Model»), que son usos de bloques definidos en las librerías, concretamente de: los Data_Tables y Data_Evolution de la librería LAS_Sim para soportar la configuración del modelo de simulación; los Assembly_Generator_Q y Part_Generator_Q de LASQ_Sim como generadores de ensambles y piezas respectivamente; el Final_Q como final del flujo de ensambles; y el bloque abstracto LinearAssemblySystem. Este último bloque deberá ser reemplazado por el modelo particular del sistema de ensamble a simular, definido por el usuario, si bien este bloque abstracto cuenta con las interfaces necesarias para conectarse con el resto de componentes considerados. En el Capítulo 8 se ejemplifica el uso del bloque LASQ_SimulationModel en un caso de estudio.

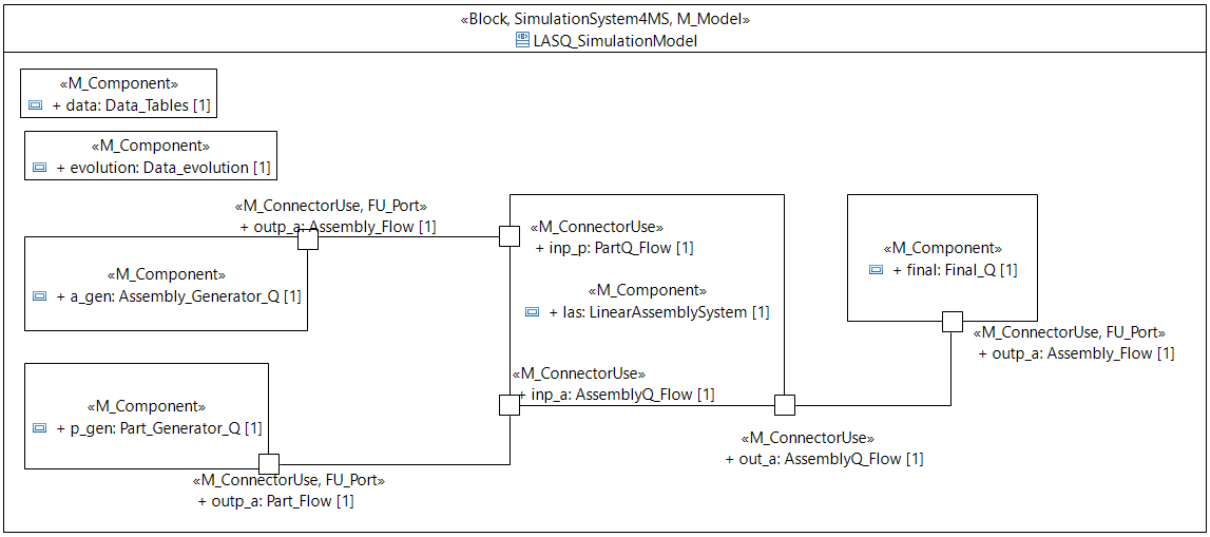


Figura 7.22. IBD del bloque LASQ_SimulationModel.

7.3.4. Paquete "Q_Functions"

De forma similar a lo comentado en el paquete Functions de la librería LAS_Sim, en la librería LASQ_Sim también se dedica un paquete a definir un conjunto de funciones, en concreto, funciones del formalismo DEVS aplicadas al modelado del comportamiento de los bloques Assembly_Step_Q, Assembly_Store_Q y Part_Store_Q. La Figura 7.23 representa en un BDD los tres elementos de tipo FunctionBehavior definidos en este paquete de la librería SysML.

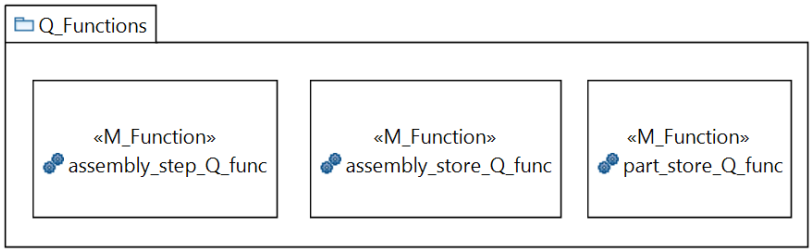


Figura 7.23. Elementos del paquete Functions de la librería LASQ_Sim modelados en SysML.

Con el fin de ejemplificar su modelado en Modelica, a continuación, se presenta el código que define la función "assembly_step_Q_func". En este código, además de definir las entradas y salidas de la función, se implementa un algoritmo que se encarga de definir los datos de salida, tanto de posición como de las desviaciones del ensamble resultante de la etapa, únicamente en determinadas condiciones, por ejemplo, si se cumple el tiempo de procesado (evento temporal "tEvent" mientras el estado es "processing") y la siguiente estación admite nuevas órdenes, o si, estando la estación bloqueada (estado "blocked"), se autoriza el envío del ensamble a la siguiente etapa (evento "aEvent).

```
function assembly_Step_Q_func
 input Boolean tEvent;
 input Boolean aEvent;
 input A_Status status;
 input Boolean allow;
 input Real dev_data[:, :];
 input Integer dev_n[:];
 input Real pos_data[:, :];
 input Integer pos_pos;
 input Real out_dev_data[:, :];
 input Integer out_dev_n[:];
 input Real out_pos_data[:, :];
 input Integer out_pos_pos;
 output Real new_out_dev_data[:, :];
 output Integer new_out_dev_n[:];
 output Real new_out_pos_data[:, :];
 output Integer new_out_pos_pos;
algorithm
 new_out_dev_data := out_dev_data;
 new_out_dev_n := out_dev_n;
 new_out_pos_data := out_pos_data;
 new_out_pos_pos := out_pos_pos;
 if tEvent then
   if status == A_Status.processing then
     if allow then
       new_out_dev_data := dev_data;
       new_out_dev_n := dev_n;
       new_out_pos_data := pos_data;
       new_out_pos_pos := pos_pos;
     end if;
   end if;
 end if;
 if aEvent then
   if status == A_Status.blocked then
     new_out_dev_data := dev_data;
     new_out_dev_n := dev_n;
     new_out_pos_data := pos_data;
     new_out_pos_pos := pos_pos;
   end if;
 end if;
end assembly_Step_Q_func;
```

7.3.5. Paquete "Q_Artifacts"

Este paquete contiene las clases que se emplean en los modelos de simulación para representar diversos artefactos (piezas y ensambles) y su geometría, definida con los

elementos de la librería TTRS_Concepts. A continuación, se comentan brevemente las clases principales consideradas.

- *Part*. Modelo de una pieza que define su geometría, tanto la nominal como la desviada, incluyendo la definición de las tolerancias geométricas entre los elementos geométricos definidos. Este modelo puede ser utilizado con dos orientaciones: a) generar valores aleatorios en sus desviaciones, dentro de los límites impuestos por las tolerancias; o b) utilizar unas desviaciones transmitidas por una interfaz de entrada. Para diferenciar estas alternativas se definen dos parámetros booleanos ("simulate" y "deviations"). Además, se define el número de geometrías de cada tipo y se crean tantos componentes de cada tipo como sea necesario. También se crean las interfaces a través de las cuales fluyen los datos de las desviaciones y de los eventos, utilizando tipos definidos en el paquete Q_Interfaces. Por último, se definen las variables relacionadas con la matriz de rotación y traslación de la pieza, necesaria para determinar la localización de la pieza dando cumplimiento a las relaciones de ensamble con otros artefactos.
- *FixedAssembly*. Modelo que representa un ensamble que, por tener todos los grados de libertad restringidos, se comporta como un sólido rígido y, por tanto, puede tratarse como una pieza. En este tipo de construcción, se definen como componentes las diversas piezas (parts) y se interpretan sus posiciones relativas fijadas para construir un único vector de posiciones mediante componentes de tipo BreakUp y Joint, que serán comentados en el paquete Utilities. Por tanto, este tipo de elemento se caracteriza por almacenar las desviaciones de cada pieza expresadas en su sistema de referencia local, y definir las localizaciones (posición y orientación) de cada pieza en el ensamble, tal y como se representa en la Figura 7.24.
- *Assembly*. Modelo de un ensamble formado por tres elementos: un FixedAssembly A de N piezas (aunque N puede ser 1), una Part B que representa la pieza que se incorpora en dicha etapa al ensamble A, y una Part U que representa un utillaje. Este modelo se define como parcial puesto que las especializaciones propias de cada caso de estudio incluirán la definición de las relaciones de ensamble entre geometrías, pero a nivel de elemento de librería únicamente establecen como propiedades los puertos a través de los cuales se transmiten los datos de posición y desviación de cada artefacto.
- *ProcessAssembly*. Modelo que utiliza un elemento de tipo StageAssembly para resolver el ensamble de proceso en base a las posiciones y desviaciones consideradas. Posteriormente, a partir de las nuevas posiciones y desviaciones de los artefactos A y B, se combinan los datos para definir la posición y desviaciones del ensamble resultante A+B. Esta transformación de los datos se realiza mediante un componente de tipo JointPA_PB, un modelo del paquete Utilities descrito más adelante.

Con el fin de ejemplificar el modelado de alguno de estos elementos, la Figura 7.25 muestra un IBD del bloque ProcessAssembly, en el que se puede observar cómo la información de la posición/desviaciones de las piezas iniciales (A y B) y del utillaje se transmite al componente de tipo Assembly, en el que se realizan los cálculos para obtener la nueva posición y desviaciones de las piezas A y B ya ensambladas. A

continuación, gracias al uso del bloque JoinPA_PB que será comentado más adelante, se combina esta información de cada pieza en dos elementos de tipo Pos_C y Dev_C, que recogen las posiciones y desviaciones características del ensamble AB.

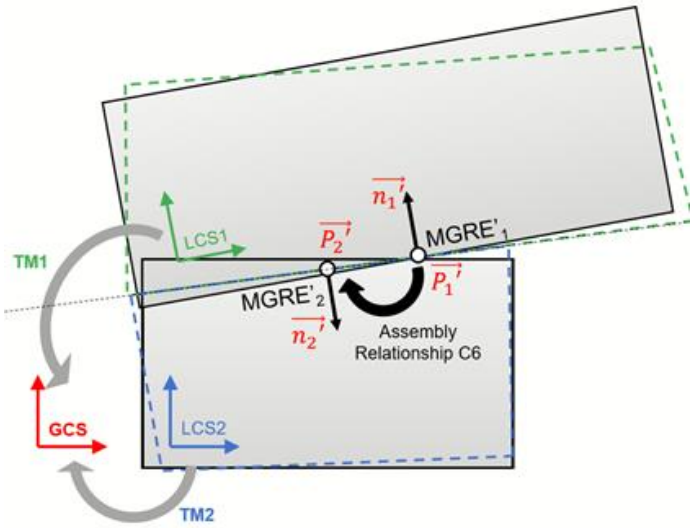


Figura 7.24. Construcción de un ensamble en un sistema de coordenadas globales. [27]

El código Modelica con el que se ha modelado este elemento ProcessAssembly se muestra a continuación.

```
model ProcessAssembly
 import LAS_Sim.*;
 parameter Integer n_parts = 1;
 parameter Real pos_fixture[2] = {0, 0};
 LASQ_Sim.Q_Interfaces.Dev_C dev_u;
 LASQ_Sim.Q_Interfaces.Dev_C dev_a;
 LASQ_Sim.Q_Interfaces.Dev_C dev_b;
 LASQ_Sim.Q_Interfaces.Dev_C dev_out;
 LASQ_Sim.Q_Interfaces.Pos_C pos_out;
 LASQ_Sim.Q_Interfaces.Pos_C pos_a;
 LASQ_Sim.Q_Utilities.JoinPA_PB joinPAPB;
 replaceable LASQ_Sim.Q_Artifacts.Assembly assembly;
equation
 assembly.pos_u.data[1, :] = {pos_utillaje[1], pos_utillaje[2], 0};
 assembly.pos_u.data[2:max_parts, :] = zeros(max_parts - 1, 3);
 assembly.pos_u.pos = 1;
 connect(joinPAPB.dev_ab, dev_out);
 connect(joinPAPB.pos_ab, pos_out);
 connect(dev_u, assembly.dev_u);
 connect(dev_a, assembly.dev_a);
 connect(pos_a, assembly.pos_a);
 connect(dev_b, assembly.dev_b);
 connect(assembly.pos_b_out, joinPAPB.pos_b);
 connect(assembly.dev_b_out, joinPAPB.dev_b);
 connect(assembly.pos_a_out, joinPAPB.pos_a);
 connect(assembly.dev_a_out, joinPAPB.dev_a);
end ProcessAssembly;
```

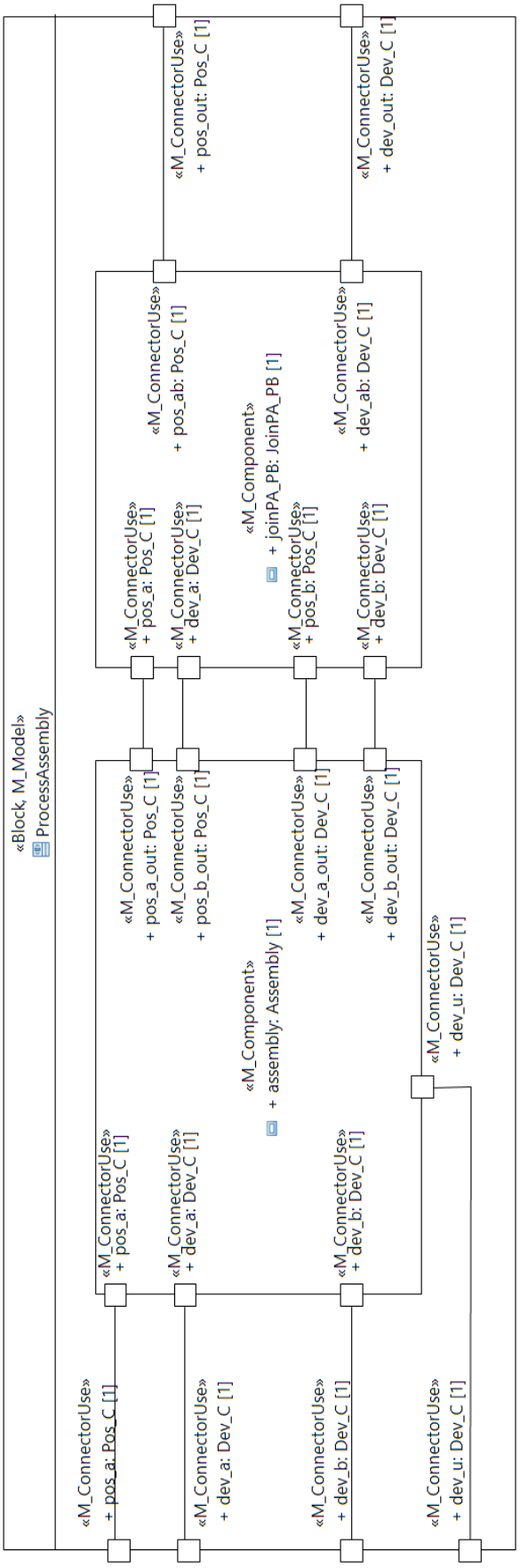


Figura 7.25. Definición en SysML del bloque ProcessAssembly.

Además de las clases genéricas propuestas, este paquete de librería también incluye algunos bloques que especializan los anteriores y que sirven para modelar casos particulares. En este punto, interesa describir brevemente dos de estos bloques, ya que serán utilizados durante el caso de estudio presentado en el Capítulo 8.

- *BasePart_with_hole*. Especialización del bloque Part que representa una pieza 2D con un contorno rectangular (caracterizado por una altura y anchura) y un agujero. Además de las cinco geometrías (cuatro caras planas y un agujero), se definen las especificaciones dimensionales y geométricas propias del esquema de acotado considerado: perpendicularidad de la cara izquierda respecto de la base, posicionamiento de la cara superior respecto de la inferior, posicionamiento de la cara derecha respecto de la izquierda, y posicionamiento del agujero respecto de la cara inferior y la cara izquierda, en este orden. Además, ofrece el eje del agujero y el plano inferior como puertos para establecer relaciones de ensamble con otros artefactos.

- *Fixture_4cyl*. Especialización del bloque Part que representa un utillaje específico dedicado a posicionar y orientar varios artefactos siguiendo un determinado proceso de ensamblaje. Para ello posee cuatro pines cilíndricos, posicionados respecto de dos caras ortogonales entre sí y que se ofrecen como puertos para establecer los contactos con otros artefactos.

Como en este capítulo se pretende ofrecer una visión general de las librerías, se ha optado por evitar excesivos detalles en la descripción de estos bloques particulares, que serán descritos con mayor detalle en el caso de estudio en el que se utilizan (Capítulo 8), dónde el contexto del caso propuesto facilita la interpretación de estos bloques.

7.3.6. Paquete “Utilities”

Este paquete agrupa diferentes bloques que dan soporte a diferentes cálculos y funcionalidades, fundamentalmente relacionados con el tratamiento de los datos manejados en las interfaces para dar soporte a una correcta propagación de los mismos. A continuación, se enumeran las clases propuestas, comentando brevemente su funcionalidad:

- Extract_PD. Extracción de las desviaciones de una pieza, es decir, la variable “dev” de un PartQ_Flow. Para ello, se define un puerto de entrada de tipo PartQ_Flow y un puerto de salida de tipo Dev_C.
- Extract_AD. Extracción de las desviaciones de un ensamble, es decir, la variable “dev” de un AssemblyQ_Flow. Para ello, se define un puerto de entrada de tipo AssemblyQ_Flow y un puerto de salida de tipo Dev_C.
- Fixture_D. Definición de las desviaciones de un utillaje, que se ofrecen a su entorno a través de un puerto de tipo Dev_C.
- JoinPA_PB. Integración de los datos de posición y desviaciones de dos piezas. Para ello se definen cuatro puertos de entrada (posiciones y desviaciones de las piezas A y B) y dos puertos de salida (posición y desviación del artefacto resultante AB).

Figura 7.26. Elementos del paquete Q_Utilities de la librería LASQ_Sim modelados en SysML.

- BreackUp. Fragmentación de los vectores de posición y desviaciones de un ensamble en diversos vectores de ambos tipos, tantos como piezas tenga el ensamble, para tratar independientemente los datos de cada uno de sus componentes.
- Join. Integración de los datos de posición de diversas piezas en un único vector.
- IncludeAD. Integración de los datos de posición y desviaciones de un ensamble, es decir, que a partir de las entradas de tipo Dev_C y Pos_C calcula una salida de tipo AssemblyQ_Flow.

La Figura 7.26 muestra un BDD con los elementos anteriormente mencionados modelados en SysML, indicando los puertos, variables y parámetros que los definen.

Para ejemplificar el modelado de estos elementos en Modelica, a continuación, se muestra el código de la definición del elemento Extract_PD, en el que se definen como propiedades parte las interfaces de tipo PartQ_Flow (pieza entrante) y Dev_C (desviaciones a transmitir a otros bloques), así como una variable booleana de tipo output (evento de llegada de pieza). La ecuación define que cuando llegue una pieza (propiedad booleana positiva) se igualen las desviaciones del puerto de entrada y de salida.

```
model Extract_PD
  LASQ_Sim.Q_Interfaces.PartQ_Flow inp;
  LASQ_Sim.Q_Interfaces.Dev_C dev;
  outer Boolean input_part_Event;
equation
  when {input_part_Event} then
    dev.data = inp.dev.data;
    dev.n = inp.dev.n;
  end when;
end Extract_PD;
```

8. Validación de la propuesta: caso de estudio

En este capítulo se presenta un caso de estudio desarrollado con el fin de poner a prueba la propuesta metodológica y el uso de los recursos de modelado (perfiles y librerías) creados a lo largo de la investigación. Aunque en el caso se aborda el análisis 2D de un proceso de ensamble multietapa, la sencillez del caso no debe entenderse como una limitación en la validación de la propuesta, porque ésta es escalable y aplicable al análisis de casos más complejos, que aborden estudios con geometrías 3D o que simulen otro tipo de procesos, como puede ser el mecanizado. A continuación, se comentan los principales aspectos de interés vinculados con la puesta en práctica de las diferentes etapas del procedimiento propuesto (apartado 4.3.1), haciendo especial mención a los resultados obtenidos en cada una de ellas. Estos contenidos se agrupan en cinco secciones, que son:

- *Representación del sistema referente*. Presentación del caso a través del modelo de especificación del sistema referente (tarea A).
- *Requisitos de los experimentos*. Establecimiento de los requisitos para los experimentos a desarrollar, que permitan el análisis de la productividad y la calidad geométrica del sistema referente (tarea B).
- *Modelado en SysML*. Modelado del sistema de simulación (tarea C) a partir de las librerías SysML presentadas en el Capítulo 7, así como la definición de los experimentos (tarea D) mediante la instanciación del modelo de simulación.
- *Transformación SysML-Modelica*. Ejecución de la aplicación de transformación automática (tarea E) descrita en el Capítulo 6, creando el modelo textual y ejecutable Modelica a partir del modelo SysML y de las librerías disponibles.
- *Resultados de la simulación*. Tras la compilación y ejecución del modelo de simulación (tarea F), se analizan los resultados obtenidos.

8.1. Representación del sistema referente

El caso se centra en el análisis de la fabricación del producto presentado en la Figura 8.1. Se trata de un bastidor fabricado a partir de la unión de cuatro piezas metálicas, cada una de las cuales cuenta con uno o más agujeros destinados a alojar los ejes de una serie de engranajes. La fabricación del bastidor comporta la unión por soldadura de las cuatro piezas, cuya fabricación individual no se considera en el análisis. Por su

parte, cabe indicar que las características funcionales más relevantes son las distancias entre los centros de los agujeros, pero la simulación se centrará en aquellas que pertenecen a piezas diferentes (representadas en rojo en la Figura 8.1). Se trata de una simulación en la que entran en juego factores como: a) las especificaciones GD&T de las piezas, b) el esquema de montaje/la forma de sujeción de las piezas, y c) las especificaciones GD&T del utillaje. Por lo tanto, el modelo de simulación deberá contemplar las desviaciones de cada una de las piezas y de los utillajes, limitadas por las tolerancias especificadas para cada elemento, y dar soporte al flujo de estos datos por las diferentes etapas del sistema simulado, modificando estas características geométricas y simulando la propagación de errores.

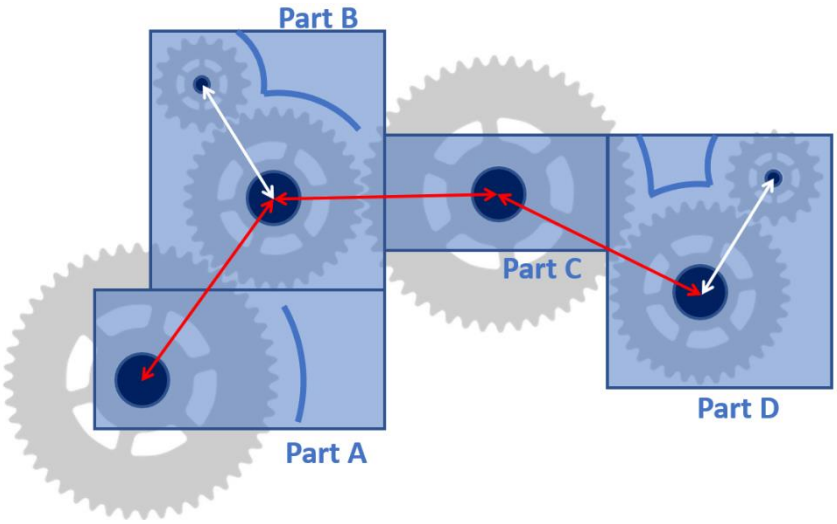


Figura 8.1. Representación del producto y de las condiciones funcionales consideradas.

Tal y como se establece en el procedimiento expuesto en la metodología, el modelado del sistema referente permite definir el conjunto de información que sirve como base para el modelo del sistema de simulación. A pesar de que en este documento el modelado de especificación (diseño) del sistema referente y su integración con el modelo de simulación no han sido objeto de una especial atención, en esta sección se presentan algunos de los diagramas SysML relativos al sistema referente del caso, denominado MyPPR por ser un sistema integrado de producto, proceso y recurso. En esta tarea de modelado se ha supuesto que SysML estándar posee suficientes construcciones y capacidades para especificar este sistema al nivel que se ha tratado, por lo que este modelo no tiene aplicado ninguno de los perfiles propuestos, orientados al modelado del sistema de simulación. La integración de especificación y simulación mediante la aplicación de lenguajes más desarrollados (extendiendo los propuestos) es una línea de investigación que se pretende explotar en trabajos futuro, aunque se han realizado algunas experiencias en el caso del perfil SysML4TA (relación entre especificación basada en features y representación TTRS).

La Figura 8.2 muestra un diagrama de definición de bloques (BDD) en el que se representa la estructura general del bloque MyPPR, que está formado por un sistema de fabricación y un producto con su plan de procesos genérico (Generic Process Plan - GPP). El sistema de fabricación considerado es una línea de ensamble multietapa (Linear Assembly System - LAS), denominado “MyLAS”, mientras que el producto es modelado como un ensamble denominado “MyProd”. Por su parte, el plan de procesos

se modela como una actividad llamada MyNPP y definida en MyProd. A lo largo de la sección se describen estos tres componentes.

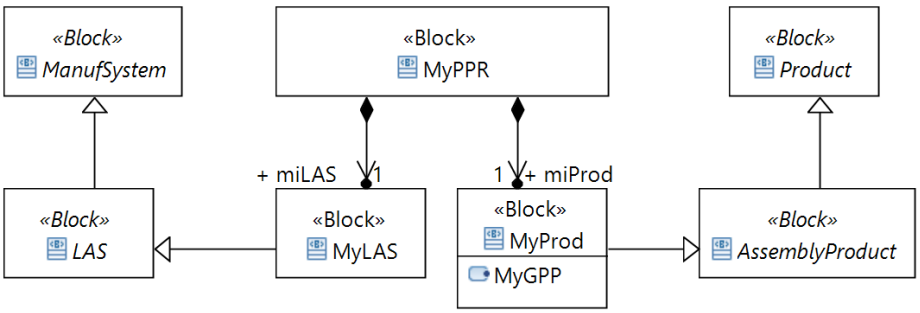


Figura 8.2. Modelo de especificación del sistema PPR (bloque MyPPR).

8.1.1. Modelado del proceso

Aunque el principio de solución del producto y proceso de fabricación generalmente se abordan en paralelo, en esta sección se presenta en primer lugar el plan de procesos genérico (MyGPP), representado mediante el diagrama de actividades de la Figura 8.3. En este diagrama se muestran los diferentes procesos considerados, tanto procesos de ensamblado como colas, y el flujo de materiales entre los procesos. Esta representación permite mostrar además que se parte de cuatro componentes (parámetros Part_A, Part_B, Part_C y Part_D), que fluyen por las acciones (flujo de objetos) y se van ensamblando, incorporando una pieza al subensamble en cada etapa hasta obtener el producto final (Assembly_ABCD). Cabe señalar además que esta representación del plan de procesos genérico puede enriquecerse en etapas más avanzadas del diseño, incorporando la asignación de acciones a recursos mediante el uso de AllocateActivityPartitions de SysML, dando soporte al modelado del plan de procesos nativo.

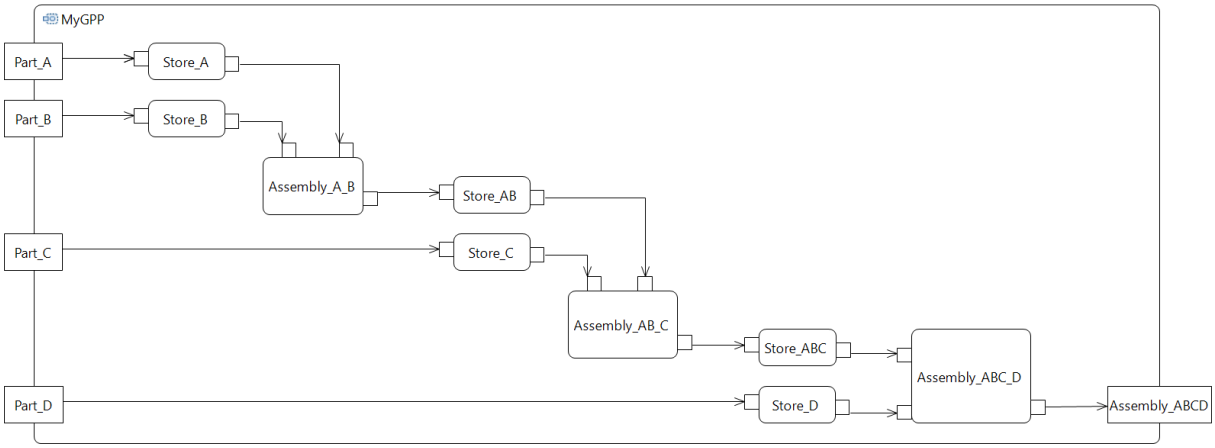


Figura 8.3. Diagrama de actividades de MyNPP, especificación del plan de proceso genérico.

8.1.2. Modelado del producto

A partir de la primera aproximación al caso que ofrece el GPP descrito, es momento de centrar el discurso en el producto, en este caso un ensamble formado por cuatro piezas metálicas que se unen entre sí mediante soldadura. Pero el modelado del producto no

debe limitarse a la configuración del producto final, porque a lo largo del proceso de fabricación el producto va a adquirir otras configuraciones que interaccionan con los recursos del sistema de fabricación. Como en este caso el proceso contempla una secuencia de tres etapas de ensamblado y soldadura, la especificación del producto debe incluir la definición tanto de las piezas individuales (Part_A, Part_B, Part_C y Part_D) como los subensambles intermedios (SubAssem_AB y SubAssem_ABC) y el ensamble final (Assembly_ABCD). Esta estructura del artefacto, que se muestra en la Figura 8.4, se corresponde con el principio de solución que deriva del GPP, si bien el MyProd podría incluir otros principios de solución, así como información relativa a otros puntos de vista del producto, adicionalmente a la de ensamble.

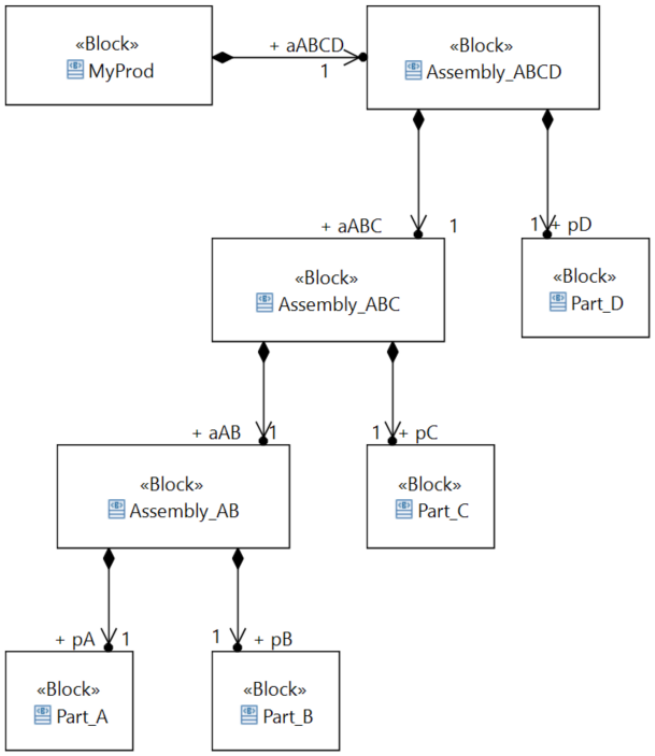


Figura 8.4. BDD de la estructura del producto especificado (bloque MiProd).

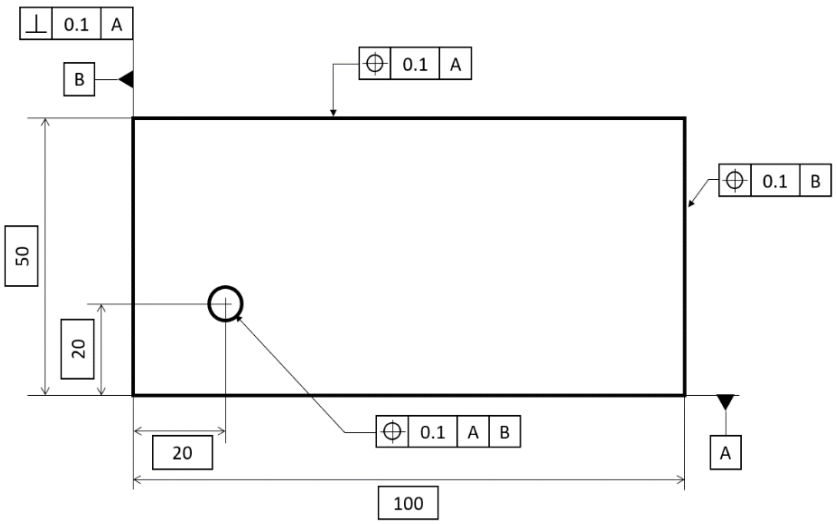


Figura 8.5. Acotado funcional de Part_A.

Por lo que respecta al modelado de las piezas individuales, es necesario trasladar la información relativa al esquema de acotado. Conviene indicar que, con el fin de simplificar el caso desarrollado, se ha empleado un esquema similar para todas las piezas, que se ilustra en la Figura 8.5 con el acotado funcional de diseño de la PartA. Como se observa, la principal superficie de referencia es la superficie inferior o base (A). La superficie de la izquierda tiene una especificación de perpendicularidad respecto de la base (referencia A), y constituye la segunda referencia (B) para otras especificaciones geométricas. Por su parte, la cara superior se posiciona respecto de A, y la derecha respecto de B. En cuanto a los agujeros considerados, todos ellos se posicionan respecto de A y B en ese orden. En cuanto a las dimensiones, se define un ancho y una altura para la pieza, así como las distancias del agujero a las referencias A y B.

Los datums A y B identificados en la Figura 8.5 son los que definen el sistema de referencia local de cada una de las piezas, que es utilizado para indicar la posición y orientación de cada una de las cuatro piezas en el ensamble, tal y como se muestra en la Figura 8.6.

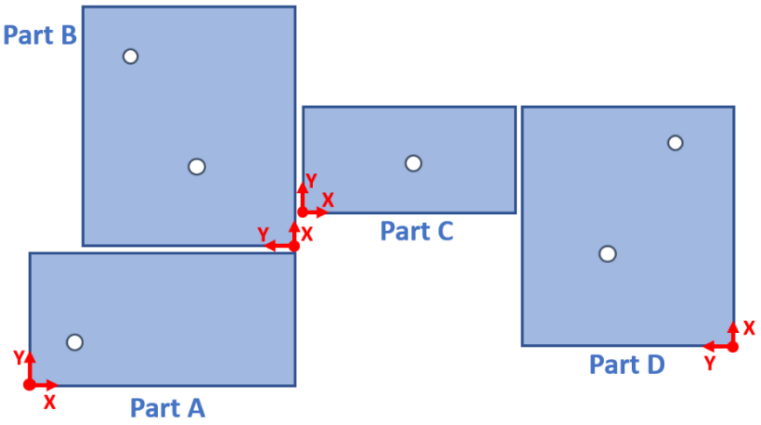


Figura 8.6. Posición y orientación de piezas en el ensamble final.

8.1.3. Modelado de los recursos

Una vez que se han definido el producto y el plan de procesos (genérico), es momento de abordar el modelado de los recursos de fabricación. Este apartado se divide en dos puntos para abordar de manera diferenciada el modelado de ensambles de proceso, utilizados en cada etapa de ensamblado, y el modelado de la línea de ensamble.

A. Modelado de ensambles de proceso

En cada una de las etapas de unión por soldadura es necesario definir un ensamble de proceso, en el que se describen las relaciones entre los artefactos producto/s y los utillajes que permiten posicionar y orientar adecuadamente los componentes a unir mediante soldadura. Para ello, se ha decidido emplear un único tipo de utillaje que está dotado de cuatro pines cilíndricos localizados en posiciones estratégicas que variarán en función de la etapa. Tal y como se muestra en la Figura 8.7, dos de estos pines (p1 y p3) son centradores, y los otros dos (p2 y p4) son de apoyo. Así, la posición relativa de los centradores p1 y p3 coincide con la de los ejes de dos engranajes (mostrado en rojo

en la figura), determinando la posición de los artefactos a ensamblar (limita su desplazamiento en X e Y). Por su parte, los pines de apoyo contactan con una cara plana (mostrada en rojo en la figura) de cada artefacto para fijar su orientación (limitar el giro en Z).

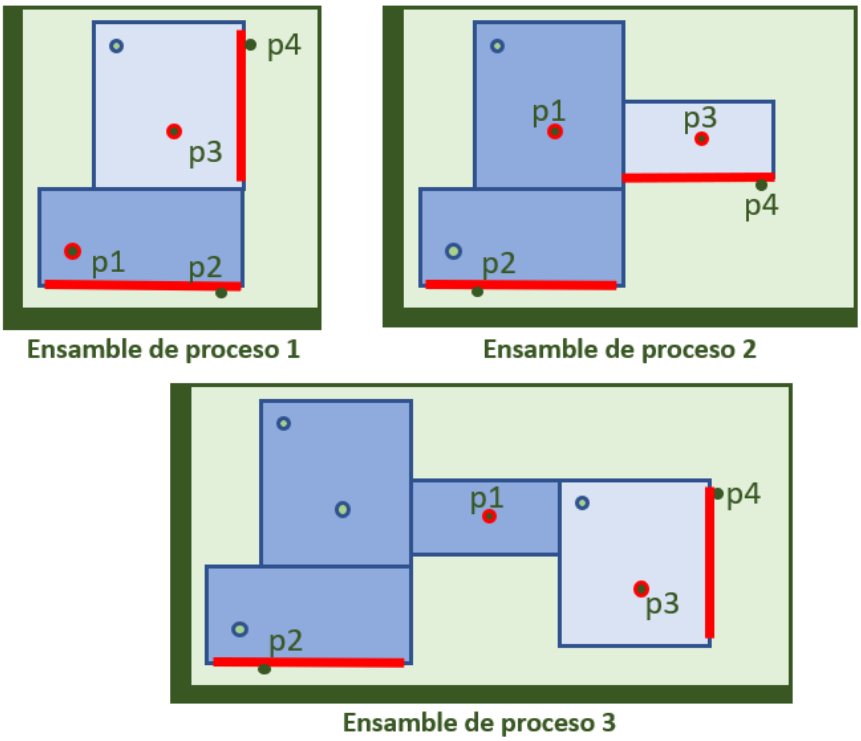


Figura 8.7. Ensamble de proceso correspondiente a cada una de las etapas consideradas.

Más allá de ilustrar gráficamente estas soluciones adoptadas en el caso, este apartado presenta el modelado en SysML de artefactos, incluidos los ensambles de proceso en los que se utilizan los modelos del producto y de los utillajes. Cabe indicar que los artefactos simples (piezas) se modelan como bloques con propiedades parte que representan features, mientras que los artefactos compuestos (ensambles) se modelan como bloques que poseen propiedades parte que representan sus componentes (piezas o subensambles), estableciendo relaciones entre ellos. Para poder establecer conexiones entre los artefactos, se definen puertos que representan los features de ensamble, es decir, aquellas geometrías que se emplean en las relaciones de ensamble. Cabe señalar una diferencia importante a la hora de modelar los ensambles de proceso, y es que los elementos que representan los artefactos a ensamblar no se han modelado como propiedades parte (relación de composición), sino como referencias (relación de agregación). Esta decisión se basa en que el ensamble de proceso no es poseedor de estos artefactos.

Para ilustrar estos aspectos del modelado de artefactos en SysML, la Figura 8.8 muestra el IBD del ensamble de proceso para la segunda etapa. En este diagrama no solo se representan las propiedades, referencias y relaciones de ensamble (conectores), sino que además se puede ver cómo algunos puertos de partes internas se conectan con puertos del bloque poseedor en lo que se conoce como delegación de puertos. En este caso, la delegación de puertos permite utilizar geometrías de piezas concretas en las

relaciones de ensamble del subensamble al que pertenecen. Es importante señalar que en la delegación de puertos se emplean BindingConnectors, que equivalen a una igualdad entre puertos, pero en las relaciones de ensamble los conectores deben estar tipeados por asociaciones o bloques asociativos que representen el tipo de contacto o relación geométrica existente entre los features conectados.

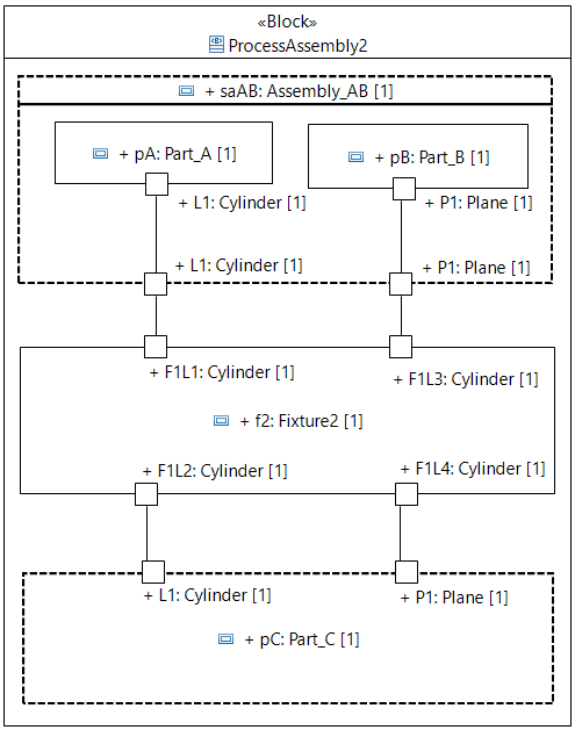


Figura 8.8. Especificación del ensamble de proceso PA_AB_C_U2.

B. Modelado de la línea de ensamblado

Atendiendo a la secuencia de montaje del producto, se diseña una línea (MyLAS) compuesta por tres estaciones de trabajo (ws1, ws2 y ws3) del mismo tipo (WorkStation). Cada estación de trabajo cuenta con un equipo de soldadura (wEquipment de tipo WeldingEquipment) y con dos almacenes previos, uno para los productos en proceso (subensambles) provenientes de otras etapas (aStore de tipo AssemblyStore) y otro para las piezas que se incorporan al ensamble en dicha etapa (pStore de tipo PartStore). Estos almacenes permiten regular el flujo de materiales, evitando tiempos ociosos y bloqueos en las estaciones. La Figura 8.9 presenta un BDD con la estructura general del bloque MyLAS.

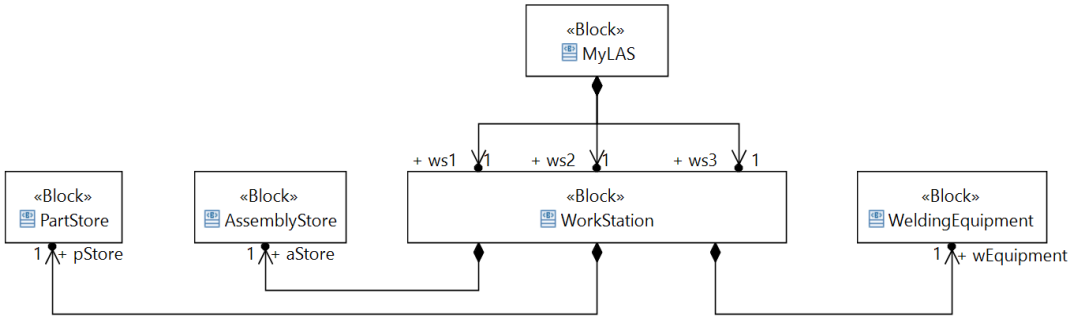


Figura 8.9. Especificación de la línea de ensamble (bloque MyLAS).

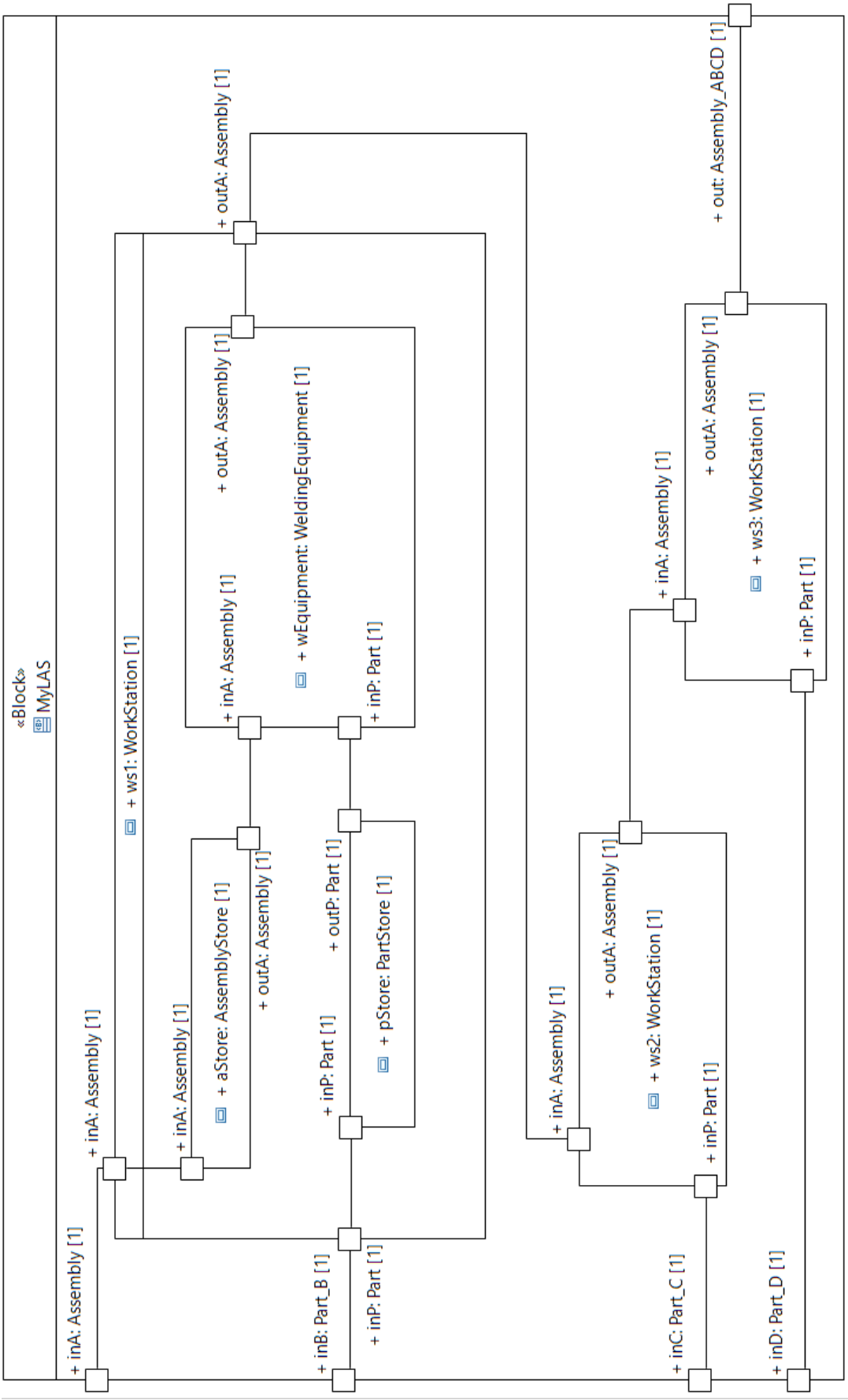


Figura 8.10. Estructura interna de la línea de ensamble especificada (bloque MyLAS).

Siguiendo con la definición del bloque MyLAS, la Figura 8.10 muestra en un IBD su estructura interna, en el que se observan tanto los puertos de entrada de piezas y el puerto de salida de ensambles como los conectores que permiten el flujo de material. Además, para ofrecer mayor detalle, el diagrama muestra la estructura interna de una de las estaciones (idéntica en las dos estaciones restantes), permitiendo visualizar los almacenes y el equipo de soldadura que componen la estación, así como los flujos internos. El diagrama muestra la interfaz del bloque con su entorno, por ejemplo, para representar el suministro de materiales por proveedores, o el envío de los productos finales a clientes.

8.2. Requisitos de los experimentos

Para analizar el sistema referente descrito, se desea desarrollar un experimento mediante simulación que permita calcular la productividad del sistema, considerando también la calidad geométrica de los productos procesados. Con este objetivo, se establecen una serie de requisitos para el experimento a desarrollar.

Por una parte, se definen requisitos funcionales, orientados a concretar qué se espera obtener del experimento. En este caso, se pretende obtener medidas básicas sobre el proceso de fabricación simulado, como es el *throughput* (unidades de productos fabricados por unidad de tiempo) y la tasa de fallo (porcentaje de productos no conformes con las especificaciones). A partir de estas medidas básicas, se deben obtener otros indicadores como la eficacia global del equipo.

Por otra parte, y aunque resulte evidente en el contexto de validar la metodología propuesta, se impone la definición del modelo de simulación y el experimento con SysML, con el fin de verificar su consistencia. La posterior ejecución de las transformaciones automáticas permitirá obtener el experimento ejecutable como código Modelica. En cuanto a los resultados obtenidos, el experimento debe permitir obtener gráficos en los que se visualice la evolución temporal de las medidas e indicadores, así como representaciones gráficas esquemáticas de los ensambles de proceso de cada etapa, mostrando las desviaciones de los artefactos procesados.

8.3. Modelado SysML del sistema de simulación

Una vez que se ha modelado el sistema referente y los requisitos impuestos sobre el sistema de simulación, ya se puede definir en SysML el modelo de simulación para el caso propuesto y de un experimento con el que ejemplificar la ejecución la simulación y el análisis de resultados. Para ello se define el paquete Case_of_Study, estereotipado como «M_Package», que contendrá todos los elementos definidos por el usuario a partir de las librerías disponibles. La estructura de este paquete se representa en el diagrama de paquetes mostrado en la Figura 8.11, en el que se muestran los principales elementos contenidos en: a) el MySimulationModel, un bloque al que se le aplican los estereotipos «SimulationSystem4MS» «M_Model» y que define el modelo de simulación; b) el Experiment_1, una InstanceSpecification a la que se le aplica el estereotipo «M_Experiment», y que es una instancia del bloque MySimulationModel

para definir el experimento, y c) tres subpaquetes, estereotipados como «MPackage» ("Artifacts", "Generators" y "WorkStations"), en los que se organizan elementos de diferentes tipos definidos por el usuario y que se utilizan para definir MySimulationModel. Como se puede observar en la Figura 8.11, a todos los elementos (paquetes, bloques, etc.) se les aplican uno o varios estereotipos de los perfiles propuestos en los Capítulos 5 y 6 con el objetivo de verificar la consistencia (perfiles SysML4MMSim, SysML4TA y SysML4DPSim) y facilitar las transformaciones automáticas a Modelica (perfil MTT2Modelica), aunque en algunos diagramas presentados a continuación no se muestren explícitamente todos los estereotipos para facilitar la legibilidad del diagrama.

Con el fin de enfatizar en la diferencia entre el modelado del sistema de simulación y su instanciación en experimentos, la sección se divide en dos apartados que abordan de forma separada estas cuestiones.

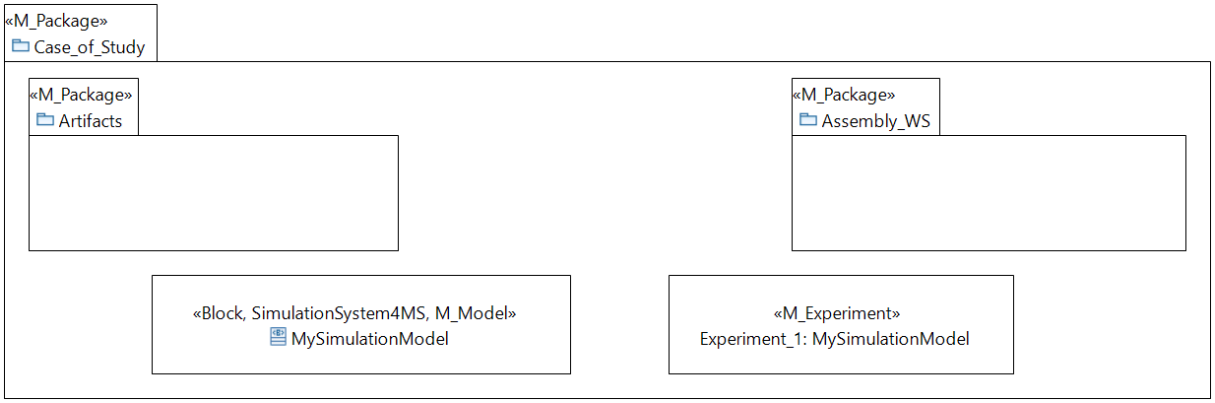


Figura 8.11. Estructura del paquete Case_of_Study.

8.3.1. Modelado del sistema de simulación

Como se ha comentado en el apartado 7.3.3, la librería LASQ_Sim incluye la definición del bloque LASQ_SimulationModel, que define un sistema de simulación básico para el análisis de líneas de ensamble multietapa. Por tanto, aprovechando esta construcción, se define el modelo de simulación del caso como una especialización del LASQ_SimulationModel, llamada MySimulationModel. Esta especialización no se limita a añadir propiedades de diferentes tipos (dos generadores de piezas adicionales, por ejemplo), redefiniendo la propiedad "las", cuya nueva definición apunta al sistema de fabricación simulado definido por el usuario (MyLAS_sim), que se describirá a continuación. La estructura básica del modelo MySimulationModel se representa en el BDD de la Figura 8.12.

Entrando en detalle en la definición del MyLAS_sim, la Figura 8.12 indica que, más allá de las interfaces heredadas de la generalización LinearAssemblySystem, este bloque añade dos interfaces de tipo PartQ_Flow (del paquete Q_Interfaces de la librería LASQ_Sim), un uso del bloque Control_Station (del paquete Blocks de la librería LAS_Sim), y tres usos (ws1, ws2 y ws3) del bloque Assembly_Station_Q (del paquete Q_Blocks de la librería LASQ_Sim).

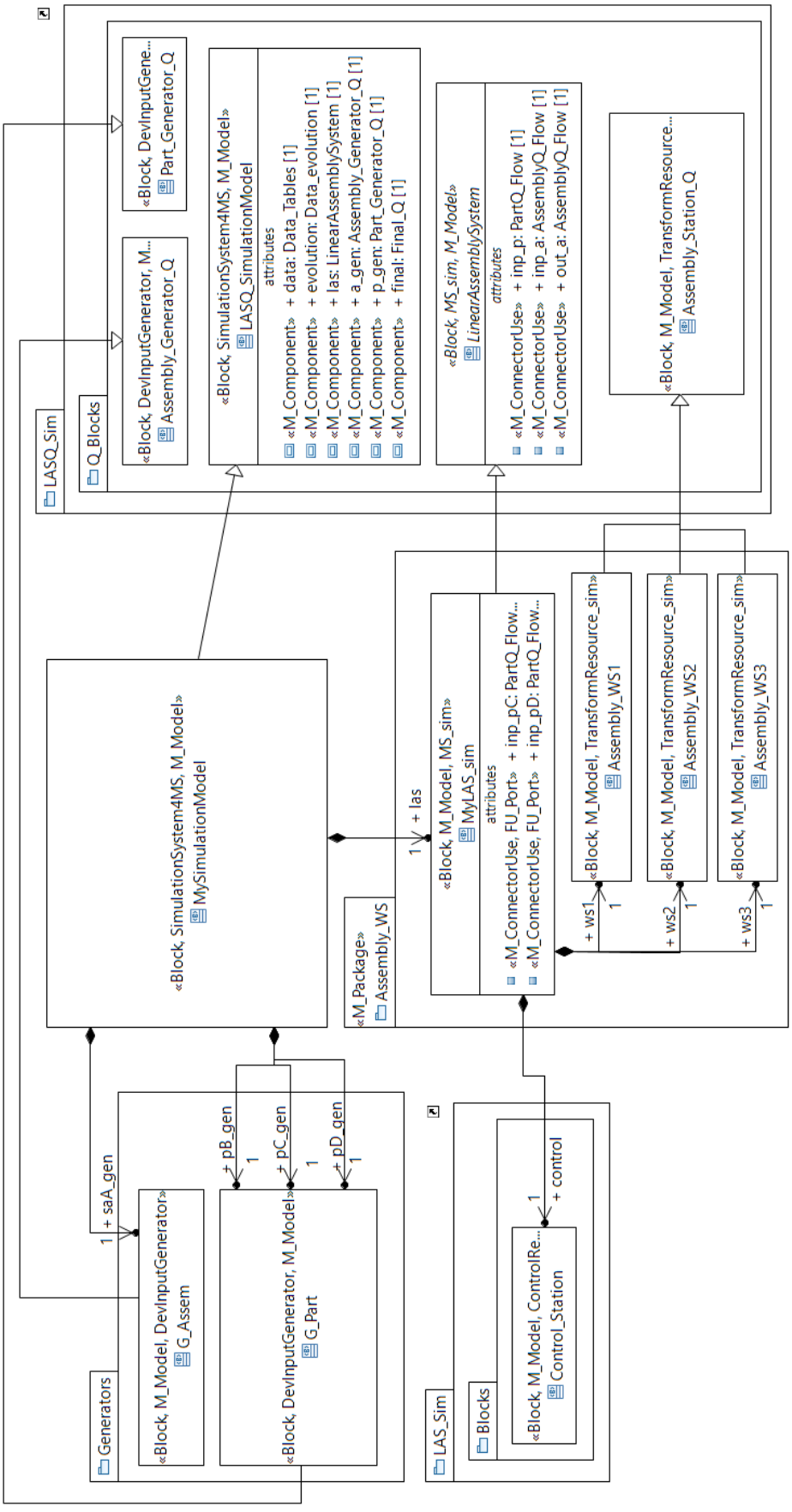


Figura 8.12. BDD del sistema de simulación (bloque MySimulationModel).

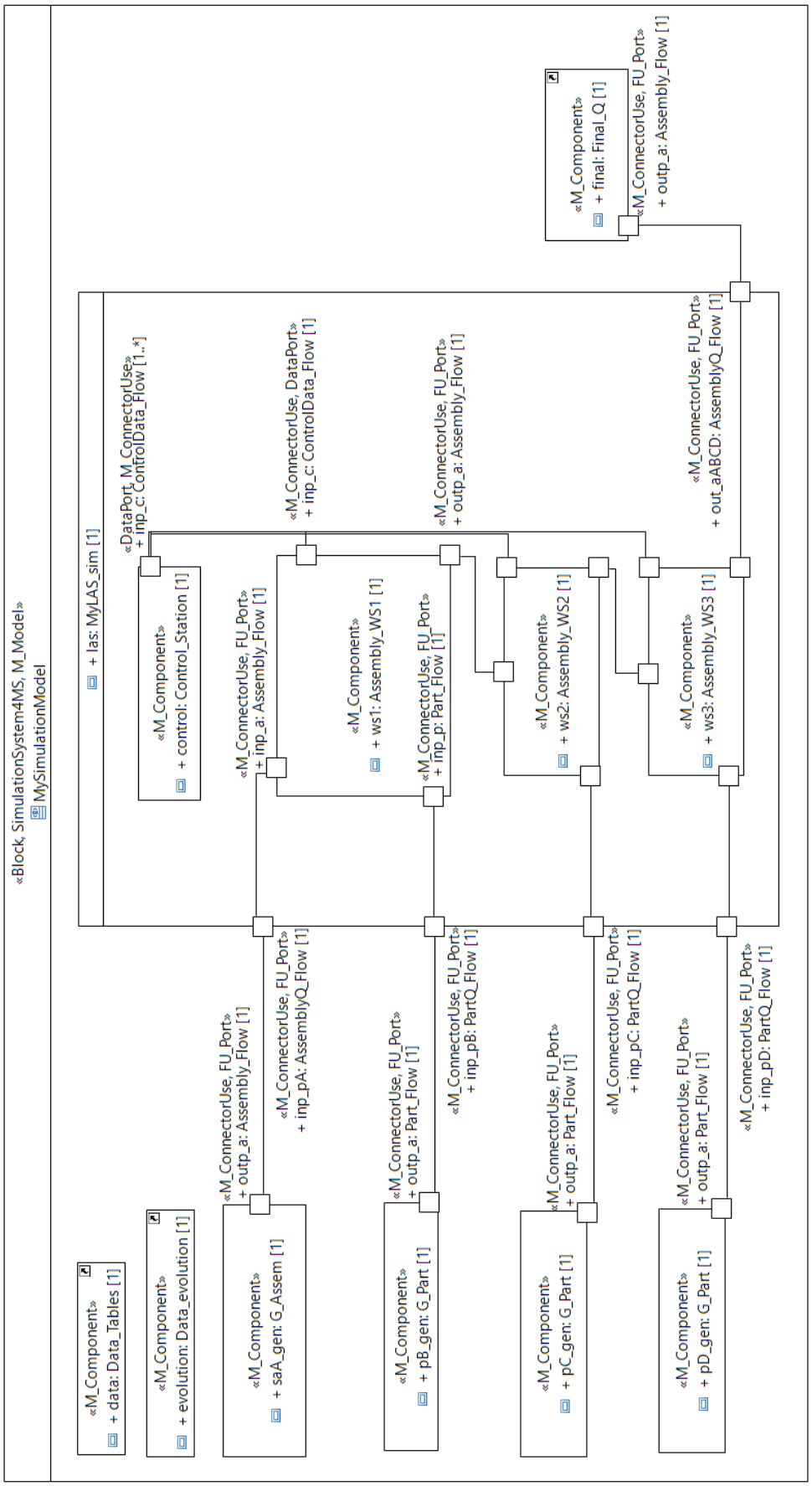


Figura 8.13. IBD del sistema de simulación.

El IBD de la Figura 8.13 representa la estructura interna del bloque MySimulationModel, mostrando el contenido de la propiedad "las" (de tipo MyLAS_sim). Como se puede apreciar en este diagrama, los puertos Assembly_Flow_Q y Part_Flow_Q de los generadores (a_gen, p_gen, pC_gen y pD_gen) se conectan con puertos de la parte "las", transmitiendo los datos del ensamble inicial y de las piezas. Poniendo el foco en la estructura interna de MyLAS_sim, se puede observar que las tres estaciones se conectan entre sí, transmitiendo los datos del ensamble resultante de una estación hacia la siguiente, hasta que la última de ellas (ws3) conecta su puerto de salida de tipo Assembly_Flow_Q con el componente "final", que es el final del flujo de ensambles. Por último, también cabe mencionar que las conexiones entre la unidad de control y cada una de las estaciones de ensamble definidas en MyLAS_sim permiten un flujo bidireccional de datos, tanto para la captura de datos y monitorización del sistema, como para la toma de decisiones desde el control central, produciendo cambios en los recursos (mantenimiento, ajuste, etc.).

A la hora de comentar cada una de estas propiedades, es necesario hacer una distinción entre aquellas propiedades que son directamente usos de elemento de librería (paquetes LAS_Sim y LASQ_Sim de la Figura 8.13) y aquellas cuyos tipos corresponden a bloques nuevos, definidos por el usuario mediante usos de elementos de librería. En el caso del uso directo de elementos de librería, su definición está disponible en Modelica y por tanto se tratan como cajas negras en SysML, estableciendo conexiones entre los usos, pero sin detallar su estructura interna. Pero el usuario debe crear otros bloques, incluyendo el propio MySimulationModel, en los que establecer completamente su definición, incluyendo la estructura interna.

Estos bloques-usuario incluyen: a) artefactos (ensambles de proceso, ensambles producto, etc.); b) generadores específicos de piezas y ensambles (sin son necesarios); y c) estaciones de trabajo concretas. Estos elementos se organizan en los tres paquetes representados en la Figura 8.13, que serán descritos a lo largo de los siguientes puntos.

A. Paquete "Artifacts"

Este paquete contiene todos los bloques SysML definidos por el usuario para modelar aquellos artefactos emulados que son necesarios para el análisis de la calidad geométrica, ya sean piezas y ensambles de las diferentes configuraciones del producto (Figura 8.3), o utillajes y los ensambles de proceso característicos de cada etapa. Con el fin de organizar el discurso, se describen en primer lugar los artefactos más complejos y se comenta su descomposición hasta llegar a los usos de elementos de librería.

En primer lugar, se comenta el modelado de los ensambles de proceso en la simulación. Con el fin de unificar las características de los tres ensambles de proceso desarrollados, se considera que todos ellos estarán formados por un utillaje, una pieza y un (sub)ensamble. A pesar de que en la primera etapa realmente se ensamblan dos piezas, en el modelo de simulación se define un subensamble que únicamente contiene la pieza

A, lo que da uniformidad al tratamiento de los ensambles de proceso y, como se verá más adelante, al modelado de las estaciones.

Con el fin de ilustrar la definición de ensambles de proceso, la Figura 8.14 presenta un diagrama BDD en el que se muestra la estructura del bloque ProcessAssembly1, que define el ensamble de proceso de la primera etapa. ProcessAssembly1 tiene tres propiedades parte que describen los artefactos que lo forman: el utillaje (f1, de tipo Fixture_4cyl), el subensamble entrante (saA, de tipo SubAssem_A) y la pieza a incorporar (pB, de tipo BasePart_with_hole). Los bloques que tipean estas propiedades serán comentados más adelante. Además, ProcessAssembly1 también tiene otras propiedades parte que definen las relaciones de ensamble entre el utillaje y cada una de los elementos a ensamblar. En este caso, son usos de bloques de la librería TTRS_concepts, concretamente los bloques C9 y C11 del paquete TTRS_Constraints que establecen el paralelismo entre un eje y un plano (C11), para simular el contacto de un pin con una cara plana, y coincidencia de ejes (C9), para simular el acoplamiento entre un pin centrador y un agujero del producto.

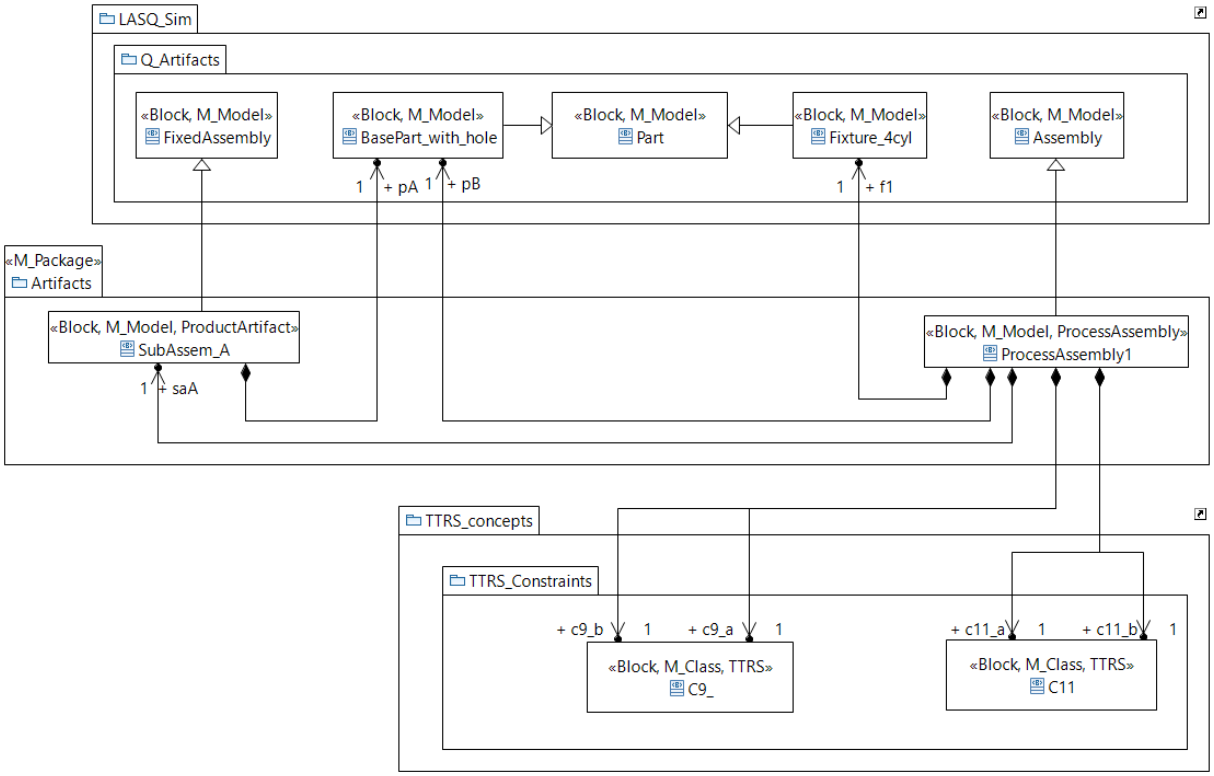


Figura 8.14. Artefactos creados para la definición del ensamble de proceso 1.

Además de establecer estas relaciones de composición, el usuario también debe definir las conexiones internas del bloque, que se representan en el IBD de la Figura 8.15. Cabe mencionar que algunos puertos del bloque contexto se utilizan para trasladar a cada artefacto los datos de sus desviaciones y posiciones previas al ensamblado, mientras que otros puertos transmiten las posiciones resultantes del ensamble y las desviaciones correspondientes para ser propagados hacia las siguientes etapas.

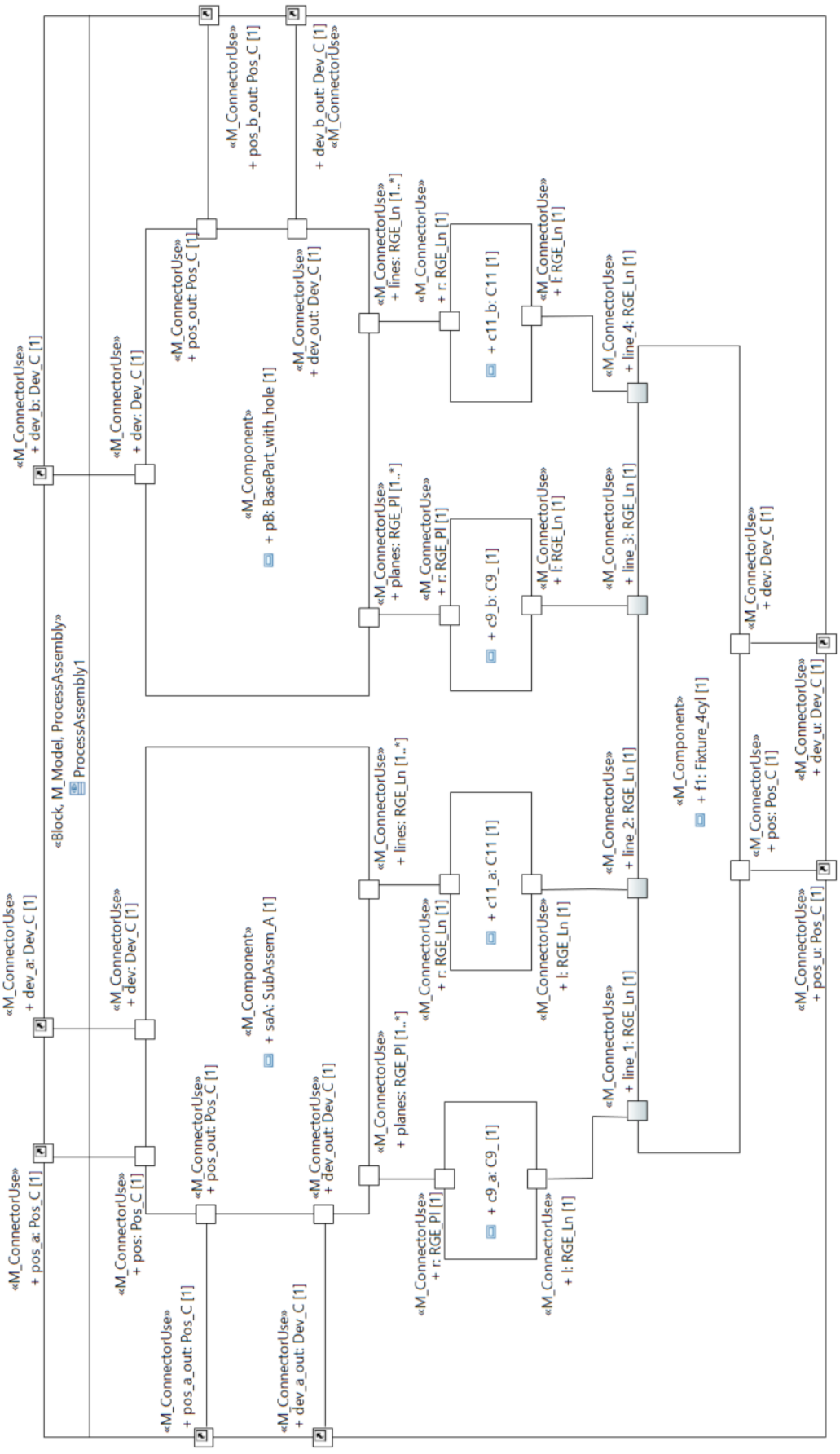


Figura 8.15. IBD del ensamble de proceso 1.

Como se ha comentado, en el ensamble de proceso se incluyen tanto piezas (utillaje y pieza a incorporar) como (sub)ensambles. Éstos últimos se modelan de forma similar al ensamble de proceso, incluyendo propiedades parte para modelar sus componentes y las relaciones de ensamble, por lo que no se le dedica mayor explicación. Cabe indicar que se han modelado cuatro ensambles en el modeado del producto (SubAssem_A, SubAssem_AB, SubAsem_ABC y SubAsem_ABCD), que corresponden a diversos estados a lo largo del proceso, y que se definen como bloques que especializan del elemento de librería FixedAssembly.

Es momento de abordar el modelado de las piezas emuladas, para las cuales únicamente se define su contorno y el agujero que se utiliza para su posicionamiento. Esta decisión permite reducir el modelado de piezas a un único tipo, un bloque que contiene un MGRE cilíndrico y cuatro MGRE plano, así como las restricciones TTRS que las relacionan. La instanciación de este tipo de pieza emulada, asignando valores diferentes a los parámetros del cilindro y del plano, permitirá definir las cuatro variantes (A, B, C y D) a partir de un único bloque. Cabe indicar que, atendiendo a las características de estas piezas, se hace uso del elemento BasePart_with_hole de la librería LASQ_Sim (apartado 7.3.5), que sirve para modelar el tipo de pieza propuesto en lugar de modelar las piezas desde cero. El bloque BasePart_with_hole cuenta con diversos parámetros que permiten al usuario definir tanto aspectos geométricos (dimensiones, posiciones, tolerancias, etc.) como otras cuestiones, como son las vinculadas con la representación de la pieza en las animaciones generadas. Lo mismo ocurre con el modelado de los utillajes de las tres etapas, que se modelan haciendo uso de un elemento de la librería LASQ_Sim llamado Fixture_4cyl, que contiene la definición de una pieza con cuatro cilindros, cuyas posiciones se definen mediante parámetros del bloque. Para ilustrar estos elementos de librería, en la Figura 8.16 se presenta un BDD en el que se visualizan ambos bloques con sus propiedades «M_Parameter».

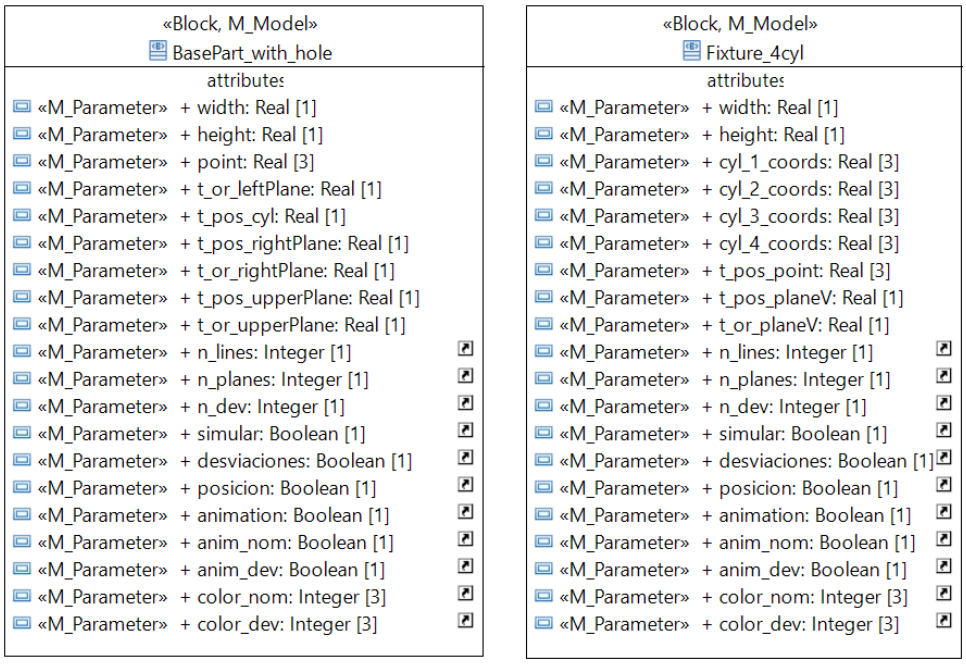


Figura 8.16. BDD del bloque de librería BasePart_with_hole.

B. Paquete “Generators”

En este paquete se definen los diferentes generadores de piezas y ensambles emulados. Su definición es muy sencilla, y consiste en especializar el bloque correspondiente de la librería LASQ_Sim y añadirle una propiedad parte que define el artefacto (pieza o ensamble) a simular. Como se ha comentado anteriormente, las piezas de partida consideradas en este caso comparten todas ellas una misma definición, por lo que también se reduce el número de generadores diferentes a definir. En concreto, se han modelado dos generadores, llamadas G_Part y G_Assem, que especializan a los bloques Part_Generator_Q y Assembly_Generator (librería LASQ_Sim) respectivamente, y añaden una propiedad parte de tipo BasePart_with_hole. Para ilustrar esta descripción, se presenta en la Figura 8.17 un BDD con las relaciones de especialización y composición haciendo uso de elementos de librería.

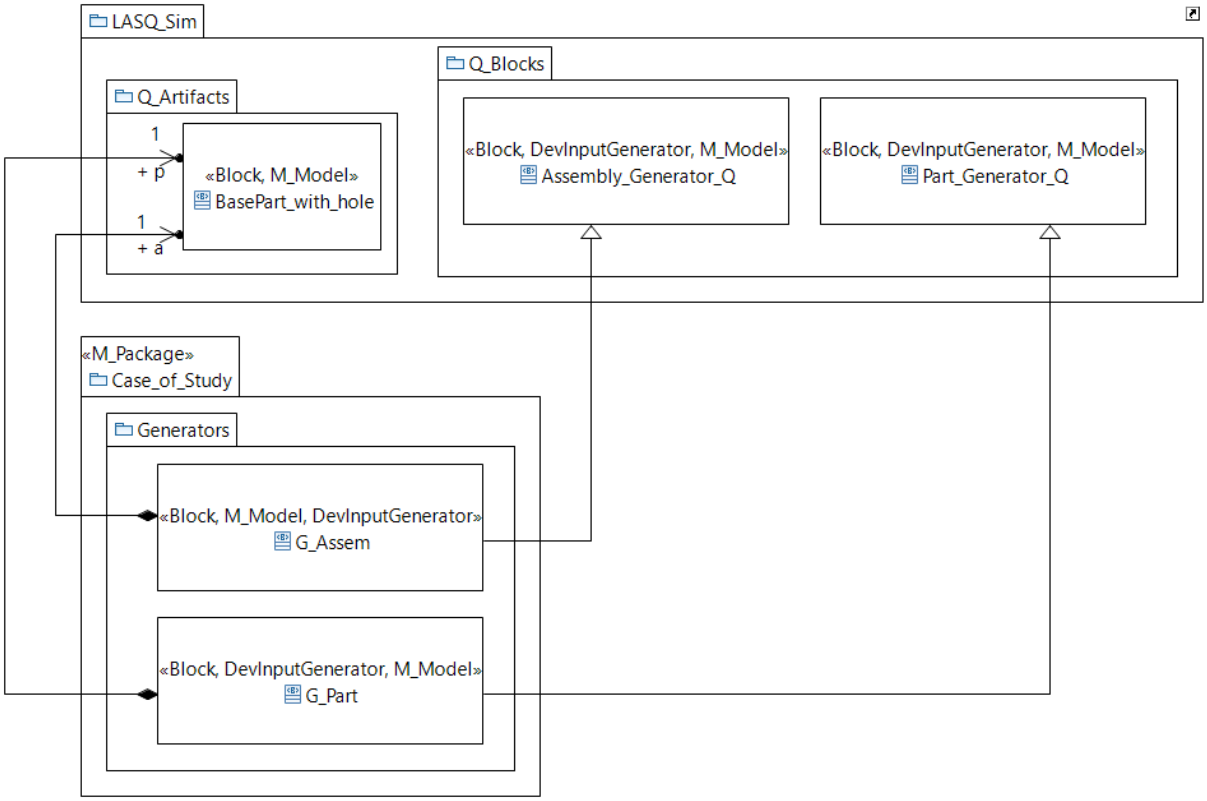


Figura 8.17. Definición de generadores de piezas y ensambles.

C. Paquete “Assembly_WS”

Este paquete contiene bloques que especializan elementos de librería, redefiniendo determinadas propiedades parte para utilizar los bloques que contienen la información específica del caso desarrollado. Por una parte, se crean tres especializaciones del bloque ProcessAssembly, una para cada etapa, en las que se redefine la propiedad “assembly”. Esta nueva definición de la propiedad apunta a un bloque del paquete Artifacts, que define el ensamble de proceso de la etapa. También se definen tres especializaciones del bloque Assembly_Step_Q, en las que se redefine la propiedad parte “process_assembly” apuntando a una de las especializaciones de

ProcessAssembly comentadas anteriormente. Finalmente, también se especializa el bloque Assembly_Station_Q tres veces, redefiniendo en cada caso la propiedad "assembly_step" por la respectiva especialización de Assembly_Step_Q. Para ilustrar los elementos de este paquete, en la Figura 8.18 se presenta la definición de los bloques definidos para la primera etapa del proceso simulado, definidos como especializaciones de elemento de librería.

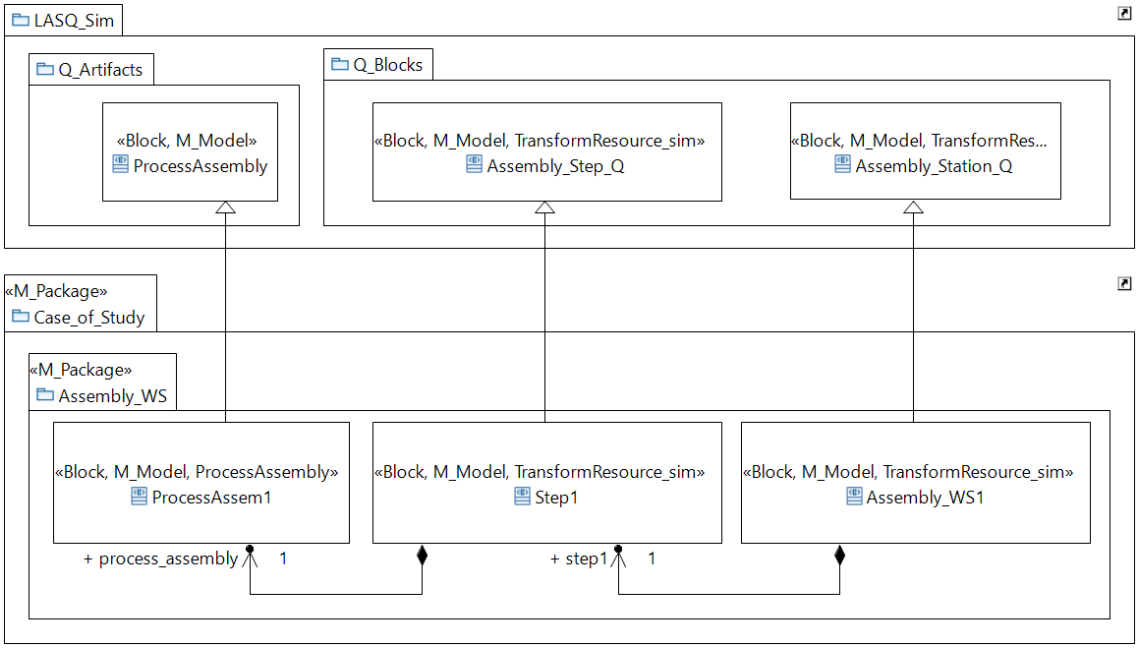


Figura 8.18. Bloques del paquete Assembly_WS para la simulación de la primera etapa de ensamblado.

Llegados a este punto, ya se ha definido estructuralmente el modelo de simulación. Sin embargo, para definir experimentos compilables y ejecutables es necesario asignar valores a todos los parámetros definidos en el modelo Esta cuestión será abordada en el siguiente apartado.

8.3.2. Modelado de experimentos

Tras la definición del modelo de simulación, es momento de concretar los experimentos a ejecutar. Para ello, se propone definir una InstanceEspecification por cada experimento planteado. Esta InstanceEspecification se estereotipa como «M_Experiment» para indicar que se trata de un experimento y orientar las transformaciones.

Mediante la InstanceSpecification se concretan las propiedades de un bloque definiendo Slots. En esta experiencia, cada Slot asigna un valor a cada propiedad, ya sean propiedades valor (indicando su valor) o propiedades parte, en cuyo caso un InstanceValue apunta a otra InstanceSpecification. Esto se muestra en la Figura 8.19, en la que se representa la InstanceSpecification Experiment_1, único experimento desarrollado en este caso. Experiment_1 es una instancia del bloque MySimulation en la que a cada una de sus propiedades le corresponde otra InstanceSpecification de las comentadas a continuación. En concreto, se crean instancias para las propiedades de

configuración (data y evolution), los cuatro generadores (a_gen, p_gen, pC_gen y pD_gen) y la línea de ensamble simulada (las).

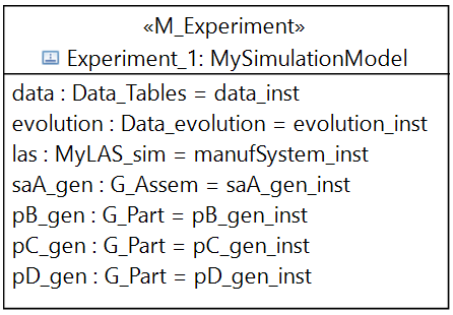


Figura 8.19. Definición del experimento.

La Figura 8.20 muestra la "data_inst", la InstanceSpecification del bloque Data_Tables que define los valores de sus parámetros, expresados como valores reales o cadenas de caracteres empleadas para definir vectores según el formato de Modelica, puesto que SysML tiene ciertas limitaciones con las multiplicidades de más de una dimensión (vectores y matrices).

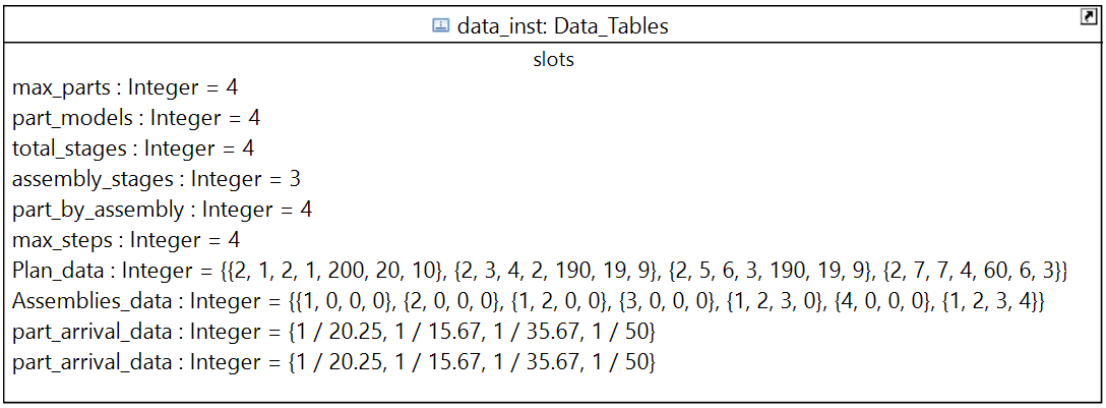


Figura 8.20. Instanciación de la propiedad data (data_inst).

La Figura 8.21 representa las instancias de cuatro generadores definidos en el modelo de simulación, uno de ellos de ensambles (bloque G_Assem) y los otros de piezas (bloque G_Part). Cabe destacar que se instancia tres veces un mismo bloque (G_Part) para asignar diferentes valores a sus parámetros y a la propiedad parte "p", que indica el tipo de pieza que genera cada uno y que, obviamente, apunta a instancias diferentes en cada caso.

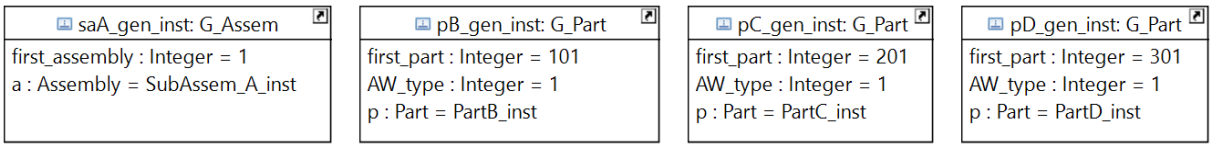


Figura 8.21. Instanciación de los generadores de piezas y ensambles.

En lo referido a la instanciación de estaciones de trabajo, estos elementos no cuentan solo con parámetros propios para identificar, por ejemplo, tasas de fallo, frecuencia del mantenimiento, etc., sino que también requiere de la instanciación de algunas de sus

partes, especialmente las vinculadas con el análisis de la calidad (por ejemplo, assembly_Step). En la figura 8.22 se presentan las InstanceSpecification referidas a estos elementos para el caso de la primera etapa.

Pero sin duda, las instancias más relevantes de cara al análisis son las InstanceSpecification de las piezas del producto y de los utillajes, ya que son los elementos constructivos básicos a partir de los cuales se construye el análisis unietapa y, mediante el flujo de datos, el análisis multietapa. Establecer los valores concretos de estas geometrías será un factor determinante en la creación de experimentos. La Figura 8.23(a) recoge una instancia del bloque de librería BasePart_with_hole que define los valores para la PartA, mientras que en la Figura 8.23(b) se muestra la instancia del bloque de librería Fixture_4cyl con los valores para el Fixture_1.

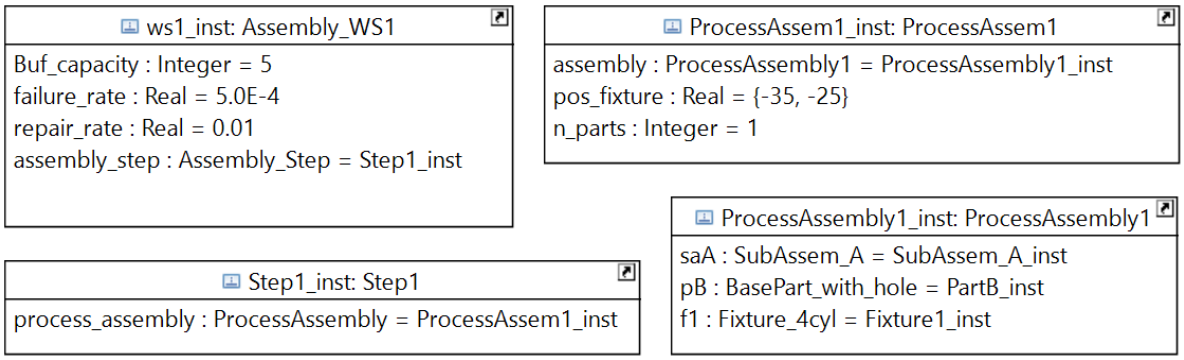
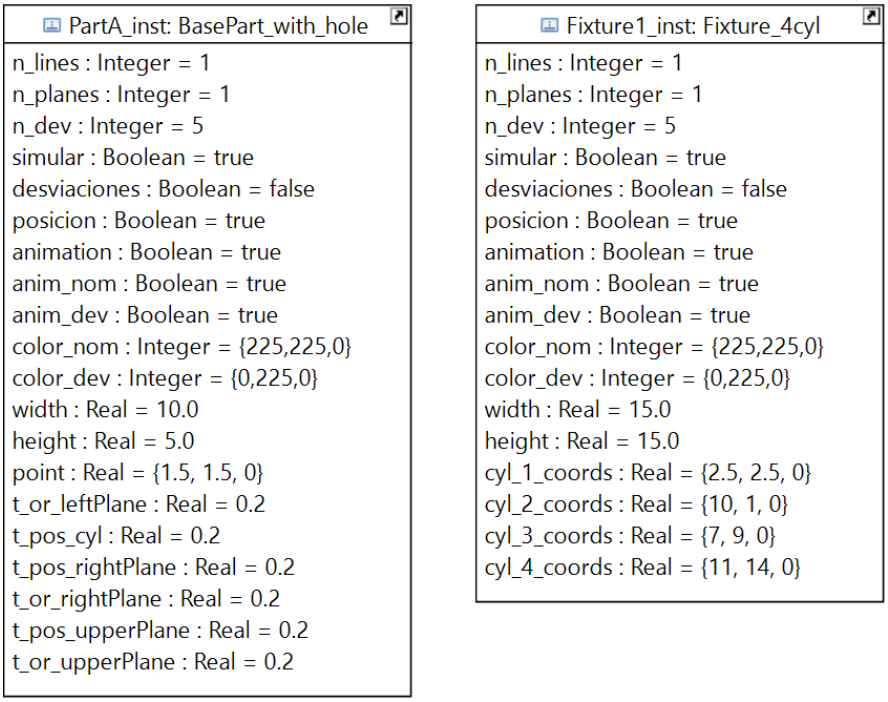


Figura 8.22. Instanciación de estaciones de trabajo y sus ensambles de proceso.



(a) (b)

Figura 8.23. Instanciación de artefactos tipo pieza: a) PartA_inst, b) Fixture1_inst.

8.4. Transformación SysML-Modelica

Fruto de la ejecución de la transformación automática se ha obtenido un fichero texto ".mo" que contiene la definición en lenguaje Modelica del modelo descrito anteriormente con SysML. La Figura 8.24 muestra una captura de pantalla del entorno OpenModelica en el que se visualiza parte del código obtenido. Además, en el árbol de librerías se muestran las librerías desarrolladas en Modelica y la estructura de paquetes del modelo textual.

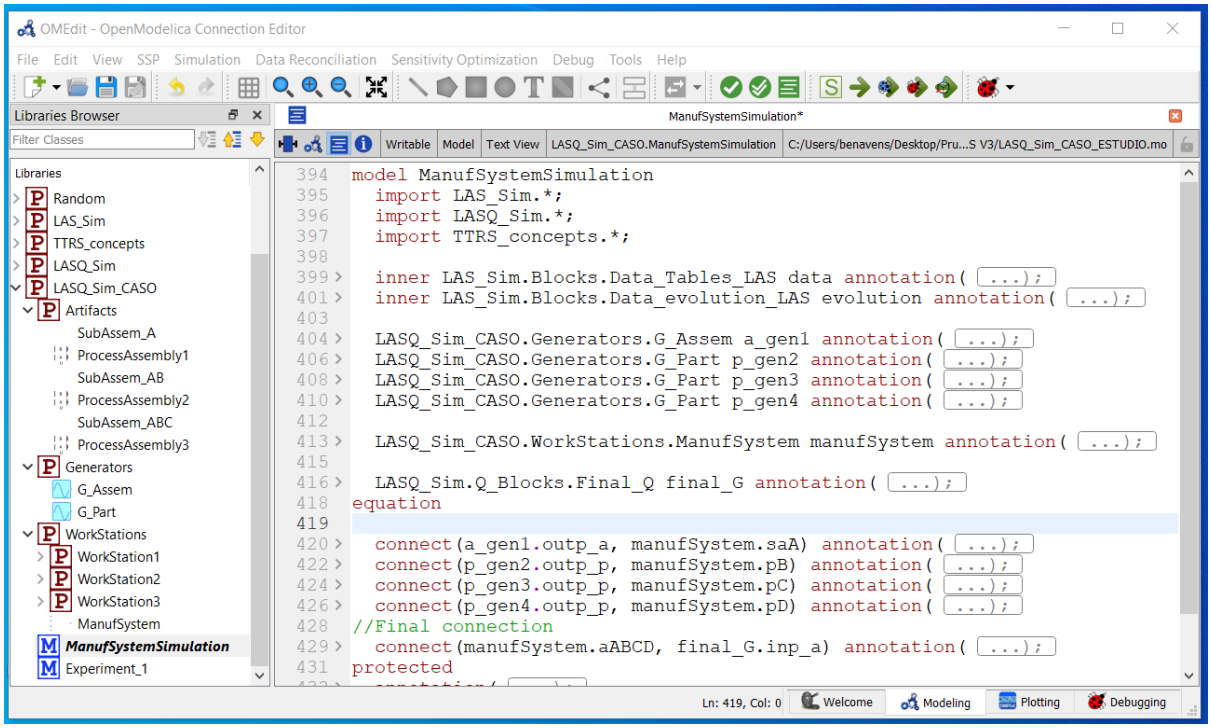


Figura 8.24. Modelo resultado de la transformación. Captura de pantalla de OpenModelica.

Para ilustrar el resultado de esta transformación, a continuación, se presenta en dos apartados el modelo de simulación y el experimento.

8.4.1. Modelo de simulación Modelica

En este apartado se describe el modelo de simulación del caso propuesto, que se presenta fragmentado y comentado para una mejor comprensión.

- Encabezado del modelo e importación de otros paquetes

```
model ThreeAssemblyStations
 import LAS_Sim.*;
 import LASQ_Sim.*;
 import TTRS_concepts.*;
```

- Componentes del modelo

```
 inner LAS_Sim.Blocks.Data_Tables_LAS data;
 inner LAS_Sim.Blocks.Data_evolution_LAS evolution;
```

```
 LASQ_Sim_CASO.Generators.G_Assem a_gen1;
 LASQ_Sim_CASO.Generators.G_Part p_gen2;
 LASQ_Sim_CASO.Generators.G_Part p_gen3;
 LASQ_Sim_CASO.Generators.G_Part p_gen4;
 LASQ_Sim_CASO.ManufSystem manufSystem;
 LASQ_Sim.Blocks.Final_Q final_G;
```

- Ecuaciones (limitado a establecer conexiones)

```
equation
 connect(a_gen1.outp_a, manufSystem.saA);
 connect(p_gen2.outp_p, manufSystem.pB);
 connect(p_gen3.outp_p, manufSystem.pC);
 connect(p_gen4.outp_p, manufSystem.pD);
 connect(manufSystem.aABCD, final_G.inp_a);
```

- Final del modelo

```
end ThreeAssemblyStations;
```

8.4.2. Experimento ejecutable

La transformación de experimentos da como resultado un modelo Modelica que contiene un uso del modelo de simulación (ManufSystemSimulation), dando valores a todos los parámetros del modelo y de sus partes internas. A continuación, se muestra un fragmento del modelo obtenido como resultado de la transformación del experimento propuesto para este caso.

```
model Experiment_1
ManufSystemSimulation m(data.Assemblies_data={{1, 0, 0, 0},{2, 0, 0, 0},
{1, 2, 0, 0}, {3, 0, 0, 0}, {1, 2, 3, 0},{4, 0, 0, 0},{1, 2, 3, 4}},
data.Plan_data={{2, 1, 2, 1, 200, 20, 10},{2, 3, 4, 2, 190, 19, 9},{2, 5,
6, 3, 190, 19, 9},{2, 7, 7, 4, 60, 6, 3}},data.assembly_stages=3,
data.part_arrival_data={1/20.25, 1/15.67, 1/35.67, 1/50},data.
part_by_assembly=4,data.part_models = 4,data.total_stages=4,
evolution.part_arrival_behabior=false,evolution.process_time_behabior=false
,a_gen1.t_x=0, a_gen1.t_y=0,a_gen1.p.height=5,a_gen1.p.width=10,
a_gen1.p.point={1.5, 1.5, 0},a_gen1.first_assembly=1,p_gen2.t_x=0,
p_gen2.t_y=0,p_gen2.p.height=8,p_gen2.p.width=9,p_gen2.p.point={3, 4, 0},
p_gen2.first_part=101, p_gen3.t_x=0, p_gen3.t_y=0, p_gen3.p.height=4,
p_gen3.p.width=8,p_gen3.p.point={4, 2, 0},p_gen3.first_part=201,
p_gen4.t_x=0,p_gen4.t_y=0,p_gen4.p.height=8,p_gen4.p.width=9,p_gen4.
p.point={3.5, 5, 0},p_gen4.first_part=301,manufSystem.ws1.buf_capacity=5,
manufSystem.ws1.failure_rate=0.0005,manufSystem.ws1.repair_rate=0.01,
manufSystem.ws1.assembly_step.process_assembly. pos_fixture={-35, -25},
manufSystem.ws1.assembly_step.process_assembly.n_parts=1,manufSystem.ws1.
assembly_step.process_assembly.assembly.f1.cyl_1_coords={2.5,2.5,0},
manufSystem.ws1.assembly_step.process_assembly.assembly.f1.
cyl_1_coords={10,1,0},manufSystem.ws1.assembly_step.process_assembly.
assembly.f1.cyl_1_coords={7,9,0},manufSystem.ws1.assembly_step.
process_assembly.assembly.f1.cyl_1_coords={11,14,0},manufSystem.ws2.
buf_capacity=5,manufSystem.ws2.failure_rate=0.0005,manufSystem.ws2.
repair_rate=0.01,manufSystem.ws2.assembly_step.process_assembly.
pos_fixture={-10,-5},manufSystem.ws2.assembly_step.process_assembly.
n_parts=2,manufSystem.ws2.assembly_step.process_assembly.assembly.f1.
cyl_1_coords={7,9,0},manufSystem.ws2.assembly_step.process_assembly.
assembly.f1.cyl_1_coords={2,1,0},manufSystem.ws2.assembly_step.
process_assembly.assembly.f1.cyl_1_coords={15,9,0},manufSystem.ws2.
```

```
assembly_step.process_assembly.assembly.f1.cyl_1_coords={18,7,0},
manufSystem.ws3.buf_capacity=5,manufSystem.ws3.failure_rate=0.0005,
manufSystem.ws3.repair_rate=0.01,manufSystem.ws3.assembly_step.
process_assembly. pos_fixture={15,-5},manufSystem.ws3.assembly_step.
process_assembly.n_parts=3,manufSystem.ws2.assembly_step.process_assembly.
assembly.f1.cyl_1_coords={15,9,0}, manufSystem.ws2.assembly_step.
process_assembly.assembly.f1.cyl_1_coords={2,1,0},manufSystem.ws2.
assembly_step.process_assembly.assembly.f1. cyl_1_coords={22,5.5,0},
manufSystem.ws2.assembly_step.process_assembly.assembly.f1.cyl_1_coords=
{27, 10, 0});
end Experiment_1;
```

8.5. Resultados de la simulación

La ejecución del experimento planteado (Experiment_1) permite obtener múltiples resultados, tanto en relación a las medias relacionadas con la productividad como a la calidad geométrica, similar a las comentadas en el Capítulo 3. A pesar de los numerosos análisis que se pueden plantear en torno a los resultados obtenidos, este apartado se limita a demostrar el correcto funcionamiento del modelo propuesto ilustrando alguno de sus resultados.

Un sencillo análisis que permite sacar ciertas conclusiones directas consiste en definir y ejecutar un segundo experimento (Experiment_2) en el que utilizar posiciones diferentes para los pines de contacto de los tres utillajes, acercándolos al pin centrador para visualizar el efecto de esta decisión sobre la precisión en la orientación de las piezas en el ensamble emulado. Los puntos de finidos en el segundo experimento se ilustran en la Figura 8.25.

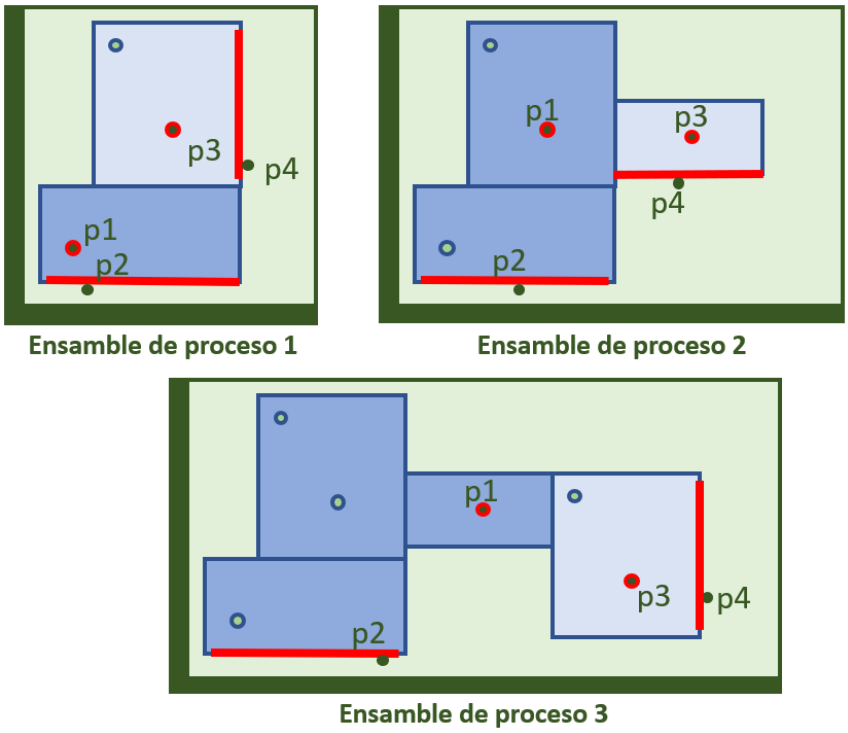


Figura 8.25. Posición de los pines en los utillajes del caso 2.

A partir de los resultados obtenidos de la ejecución de dos experimentos, la Figura 8.26 muestra en forma de gráfica los valores de la orientación para la pieza C en el ensamble final, a la salida de la tercera etapa. A nivel nominal, la orientación de la pieza tiene un valor igual a 0, pero como se puede observar, en ambos casos se han obtenido valores desviados que difieren considerablemente entre los dos casos. Así, en el caso de alejar el punto de contacto del eje de giro (pines centradores), el efecto del error en el posicionado del eje tiene mayor efecto sobre el error en la orientación de la pieza (Caso 1), mientras que si ambos pines están más cercanos (caso 2) se reduce este tipo de error. Sin embargo, en el Caso 2 se ha detectado un ligero pero apreciable incremento de la variabilidad entre productos simulados (desviación típica), con lo que esta solución es más sensible a los errores que el Caso 1, que tiene un comportamiento más estable, aunque con una media superior, como se ha indicado.

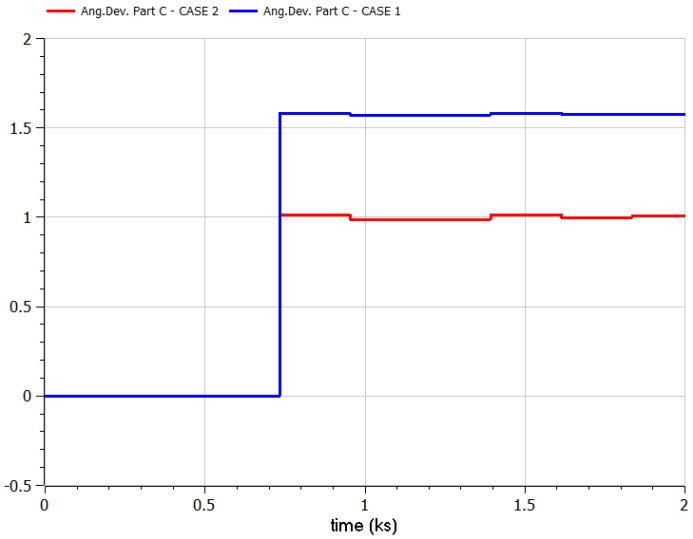


Figura 8.26. Error en la orientación de la pieza D. Comparativa entre casos.

Sería interesante abordar otras comparativas, por ejemplo, para analizar la selección de unas u otras caras para el contacto con los pines de apoyo, determinando el efecto de esta decisión en las desviaciones de orientación de cada pieza, pero estas propuestas exceden del propósito de esta sección.

Finalmente, cabe indicar que la librería desarrollada en Modelica permite obtener una representación gráfica de los ensambles de proceso de cada etapa, permitiendo visualizar su evolución temporal mediante una animación en la que se aprecian los cambios en los artefactos emulados. La Figura 8.27 presenta una captura de pantalla de la animación obtenida en OpenModelica.

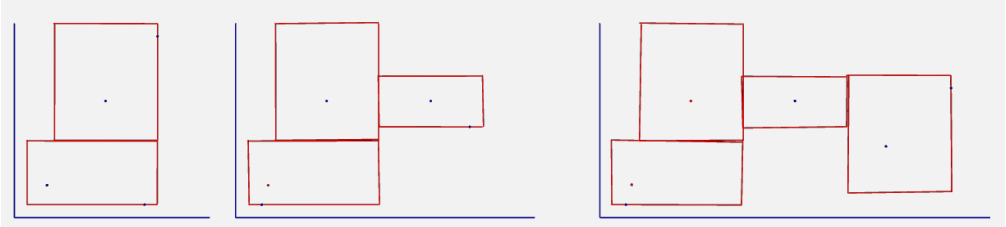


Figura 8.27. Captura de la animación obtenida con OpenModelica.

9. Conclusiones y trabajos futuros

A modo de cierre de la investigación realizada hasta el momento, en este último capítulo se presenta una síntesis del mismo, un compendio de conclusiones extraídas, identificando tanto las aportaciones como las limitaciones del trabajo, y un listado de los trabajos futuros que le darán continuidad.

En primer lugar, es importante recordar la fuerte relación de la investigación con la ingeniería de sistemas basada en modelos (MBSE), un enfoque que puede aportar mucho valor a la industria manufacturera actual y que va a sustentar muchas iniciativas de futuro. Para favorecer la implantación de este enfoque en el diseño de sistemas de fabricación discreta, se ha propuesto una metodología orientada al aseguramiento de la consistencia de los modelos a utilizar en las etapas tempranas de su ciclo de vida, una cuestión fundamental en este enfoque. Adoptando el enfoque MBSE centrado en la simulación, se aborda la creación de modelos de simulación que permitan validar el diseño y configuración del sistema de fabricación, analizando su impacto en la fabricación de los productos, especialmente en la productividad y la calidad geométrica. Otro aspecto a destacar es que los fundamentos de esta propuesta metodológica son trasladables a otros tipos de sistemas, modelos y etapas diferentes a los ahora considerados.

Los modelos de simulación que se han propuesto son una importante herramienta para los procesos de mejora de la productividad y calidad, un objetivo básico en paradigmas industriales operativos como el Production Quality. Además, estas simulaciones también pueden resultar de gran utilidad en el diseño de sistemas de fabricación reconfigurables, dando soporte al testeo virtual de los principios de solución desarrollados. El uso de la simulación permite evaluar los diseños y configuraciones del producto y del sistema de fabricación, antes de que sus ciclos de vida intersecten (Ecosistema de Fabricación del NIST) en la etapa de ejecución de la fabricación.

Para llevar a cabo los análisis planteados basados en simulación se ha optado por modelar un supersistema que integra, como partes, los productos, procesos y recursos, denominado en este documento sistema PPR. Este enfoque de modelado y el hecho de que los análisis contemplen la productividad y la calidad geométrica, obliga al desarrollo de simulaciones multi-dominio, circunstancia que requiere del establecimiento de múltiples relaciones de dependencia entre las diferentes partes (modelos) que constituyen el modelo del sistema PPR.

Estas cuestiones, recogidas en el planteamiento y el objetivo principal de la investigación, han servido de guía en el desarrollo de este trabajo, cuyas conclusiones

y resultados se resumen en el siguiente apartado, que da paso al apartado dedicado a trabajos futuros.

9.1. Resultados y conclusiones

En esta sección, que se estructura en tres partes, se pretende dar respuesta a las preguntas de investigación planteadas en el inicio de la investigación (sección 1.3) en base a los resultados obtenidos y a las principales conclusiones que se pueden extraer de la misma.

En primer lugar, en relación a las metodologías para el modelado de sistema de simulación y particularmente para el análisis de la calidad y la productividad (RQ1), cabe indicar que se han explorado propuestas alineadas con el enfoque MBSE y que proponen el uso complementario de modelos descriptivos (SysML) y de análisis (Modelica), pero en la búsqueda realizada no se han encontrado trabajos que aborden específicamente la simulación conjunta de la productividad y la calidad. Este hecho ha motivado el diseño de una metodología específica para la construcción de este tipo de simulaciones, que se ha concretado en las siguientes aportaciones:

- Marco de la propuesta, en el que se establecen los diversos tipos de modelos involucrados, así como relaciones entre ellos y las tareas de transformación de modelos expresados en diferentes lenguajes.

- Procedimiento para la definición de modelos de simulación orientado a asegurar su consistencia en base a la semántica específica del dominio bajo análisis. Este procedimiento empieza con el modelado del sistema referente, a partir del cual se establecen los requisitos para el sistema de simulación. Seguidamente se propone diseñar el sistema de simulación y los experimentos con SysML, asegurando su consistencia. Finalmente se procede a la transformación de estos modelos a Modelica, un lenguaje que soporta la ejecución de las simulaciones.

Entre las principales conclusiones vinculadas con primera pregunta de investigación destacan las siguientes:

- El marco propuesto se ha mostrado adecuado para el enfoque MBSE centrado en la simulación, porque los modelos que lo integran soportan en su totalidad las actividades que conforman el procedimiento. Si nos centramos en las actividades de modelado involucradas en el procedimiento y en los entornos de edición utilizados para las mismas, cabe indicar que ambas favorecen un diseño colaborativo.
- La selección del lenguaje EGL para la implementación de los códigos de transformación de modelos se ha mostrado adecuada. El hecho de que Papyrus esté construido sobre Eclipse, ha permitido integrar todo el proceso de edición y transformación de modelos, ratificándose que la elección del entorno ha sido adecuada.

- Como ha demostrado el caso de estudio presentado, el procedimiento propuesto establece una transformación de modelos que asegura la correcta definición de las simulaciones en base a los modelos descriptivos desarrollados, evitando la pérdida de información y asegurando la consistencia del modelo resultante.

- Por su parte, el hecho de que se haya dotado al entorno de modelado con toda una serie de recursos (perfiles, librerías y aplicaciones de transformación) desarrollados en este trabajo, facilita enormemente el trabajo que el usuario tiene que realizar para construir modelos válidos.

A continuación, se aborda la respuesta a la RQ3, centrada en la definición de la semántica a considerar en los modelos de simulación propuestos. Tras analizar los requisitos de los sistemas de simulación y valorar los conceptos involucrados en su definición, se han desarrollado diversos metamodelos que se presentan a continuación:

- Modelo conceptual sobre los conceptos propios de la simulación de sistemas PPR, dando soporte al modelado de su comportamiento discreto y al flujo de materiales. Este metamodelo se ha implementado posteriormente como perfil SysML (SysML4MSSim).

- Modelo conceptual sobre los conceptos propios del modelado de artefactos físicos, tanto la especificación basada en features como su representación basada en TTRS. Este metamodelo se ha implementado posteriormente como perfil SysML (SysML4TA).

- Modelo conceptual sobre los conceptos propios de la simulación de sistemas PPR que incorporan el análisis de la calidad geométrica, basada en el modelado TTRS de los ensambles de proceso. Este metamodelo se ha implementado posteriormente como perfil SysML (SysML4GDPSim), especializando estereotipos del perfil SysML4MSSim y haciendo uso de estereotipos del perfil SysML4TA.

- Metamodelo sobre los conceptos básicos de Modelica, identificados para establecer su correlación con construcciones SysML y facilitar así las transformaciones de modelos de un lenguaje a otro. Este metamodelo se ha implementado posteriormente como perfil SysML (MMT2Modelica).

- Definición de reglas OCL en cada estereotipo de los perfiles desarrollados para trasladar la semántica estática de los lenguajes propuestos.

En relación con estas propuestas y en el contexto de la pregunta formulada, se pueden destacar las siguientes conclusiones:

- El desarrollo de modelos conceptuales exige un amplio conocimiento del dominio de interés. En este caso, ha sido necesario indagar sobre las características de los modelos de simulación, el dominio de los sistemas de fabricación, el modelo TTRS y el metamodelo Modelica, entre otras cuestiones.

- La utilización de modelos conceptuales, basado en diagramas de clases UML/SysML para mostrar/establecer la sintaxis abstracta de los lenguajes propuestos ha resultado adecuada para establecer los estereotipos necesarios y mostrar las relaciones y restricciones entre los mismos. Sin embargo, esta información se ha tenido que transformar posteriormente en una definición de los estereotipos y de las reglas OCL necesarias que definen la semántica estática del lenguaje. Aunque este planteamiento permite detectar muchas inconsistencias, presenta limitaciones que derivan del grado de formalización semántica que aplica, inconveniente que se puede salvar mediante el uso de

lenguajes ontológicos, que permiten establecer los axiomas que declaran los conceptos del dominio. Al respecto conviene recordar que una ontología de dominio es una especificación explícita de la conceptualización de ese dominio.

- La aplicación del perfil SysML4MSSim (y su extensión SysML4GDPSim) permite definir un sistema de simulación para el análisis de sistemas de fabricación con una estructura muy cercana a la arquitectura física del sistema referente. El comportamiento está definido en los bloques con comportamiento atómico (a nivel de subfase en esta investigación) y emerge en los bloques complejos a partir de la interacción entre sus partes.

- Aunque se han encontrado propuestas que apuntan al uso de los conceptos TTRS para representar tolerancias, no se han encontrado trabajos que traten las desviaciones (reales o simuladas) de la geometría y su tratamiento, tanto para abordar los problemas de acumulación (basados en el peor caso, en estadísticos o en simulación) como los de propagación entre etapas.

- Como se comprueba en los modelos desarrollados, los conceptos TTRS, combinados con construcciones matemáticas como el DMV, son una solución válida para la definición de modelos matemáticos que representen geometrías nominales y desviadas, así como sus relaciones, dando soporte al análisis de desviaciones y tolerancias geométricas.

- La aplicación del perfil MMT2Modelica facilita enormemente las transformaciones, permitiendo identificar las construcciones de SysML definidas en el modelado de un sistema software basado en objetos (con un comportamiento temporal definido) con los conceptos del metamodelo del lenguaje Modelica. Esta identificación de conceptos permite definir algoritmos de transformación automática más sencillos, que navegan e interpretan el modelo origen en base a los estereotipos de este perfil, que también condicionan la definición del modelo texto resultante. Aunque han servido de base otras propuestas previas, como SysML2Modelica, se ha construido el perfil adaptado a las necesidades específicas del mecanismo de transformación propuesto.

A continuación, se da respuesta a la RQ2, discutiendo todos aquellos aspectos vinculados con el uso combinado de SysML y Modelica para el modelado de simulaciones híbridas. Al respecto, resulta fundamental comentar que, como se ha indicado anteriormente, el uso combinado de modelos descriptivos y de simulación (en concreto de SysML y Modelica) es la base de la metodología desarrollada. Para comprobar su complementariedad y las ventajas que aporta el uso combinado de ambos lenguajes, se han desarrollado diferentes tareas que han dado lugar a los siguientes resultados:

- Recursos de modelado (librerías), definidos tanto en SysML como en Modelica, y que dan soporte específicamente al análisis de la productividad y la calidad en sistemas de fabricación multietapa. Sobre una misma librería de base, que sirve para simular el comportamiento discreto de estos sistemas y el flujo de materiales, se han presentado dos propuestas alternativas para el modelado de las desviaciones geométricas y su propagación:

    - Simulación basada en la técnica Stream of Variation, adoptada para el análisis de un proceso de ensamblado 2D. En este tipo de modelos, se

establece un conjunto de variables de estado que definen características de los productos, concretamente las desviaciones en posición y orientación de cada pieza del ensamble. Estas variables se modifican progresivamente a lo largo de las etapas consideradas mediante expresiones lineales matriciales, simulando la acumulación de desviaciones geométricas.

  - Simulación basada en el modelado de ensambles de procesos con TTRS. En esta solución alternativa, los conceptos TTRS han sido la base para desarrollar modelos matemáticos en los que se representan superficies de los artefactos y las relaciones entre ellas, soportando tanto su definición nominal como la simulación de geometrías desviadas. El uso de este tipo de modelo aplicado a la definición de ensambles de proceso permite calcular las desviaciones que se producen en una única etapa. Unos cálculos que, dependiendo del modelo de acumulación seleccionado, dan resultados diferentes, porque en algunos (p.e. Simulación Montecarlo) se consideran las funciones de distribución de las desviaciones. Unas desviaciones que se transmiten a las siguientes etapas del proceso multietapa, pudiéndose obtener las desviaciones de las características geométricas del producto final para comprobar si estas respetan las tolerancias especificadas.

- Mecanismo de transformación automática de modelos SysML a texto Modelica. Esta transformación automática permite mantener la consistencia entre el diseño del sistema de simulación en SysML y su definición como modelo ejecutable Modelica. La propuesta de transformación se basa en un algoritmo en el que se plantean bucles para la transformación de paquetes, bloques y especificaciones de instancia (experimentos).

- Código EGL, implementado para la transformación de los modelos SysML estereotipados, que han sido construidos utilizando las librerías predefinidas, a modelos Modelica ejecutables. Este código se ha validado en los casos de estudio presentados.

Tras reflexionar sobre el uso combinado de SysML y Modelica, en base a las propuestas elaboradas, se pueden extraer las siguientes conclusiones:

- Modelica es un lenguaje con capacidades para abordar simulaciones híbridas y multidominio y el entorno de modelado de sistemas Papyrus se ha mostrado adecuado para el modelado de tipo de sistemas de simulación.

- El modelado descriptivo con SysML es una opción válida para el diseño de modelos de simulación. Los mecanismos con los que cuenta, como la creación de perfiles o la definición de reglas OCL, han permitido la creación de un lenguaje específico (DSML) que permite definir modelo que posteriormente se transformará de forma automática a un modelo de simulación ejecutable.

- El modelado SysML de un sistema de simulación a nivel de usuario es relativamente intuitivo y sólo requiere de un conocimiento básico del lenguaje. Sin embargo, el modelado a nivel de desarrollador exige un mayor conocimiento y destreza. Este es el motivo por el que se han desarrollado librerías de elementos

de modelado predefinidos, que facilitan la tarea de modelado al usuario y aseguran la consistencia del modelo, limitando la introducción de errores.

- La librería desarrollada para el modelado TTRS de artefactos físicos no sólo tiene valor en el marco de la metodología propuesta o en la simulación de sistemas PPR, sino que tiene suficiente generalidad para representar matemáticamente geometrías y relaciones de cualquier tipo de artefacto físico. Estas construcciones tienen un nivel de detalle suficiente como para definir tolerancias geométricas y simular sus efectos sobre las desviaciones, como demuestran los casos desarrollados, presentados en este y en otros trabajos, por lo que esta librería es un recurso interesante también de cara al planteamiento de otros análisis.

- Los dos tipos de modelos de simulación propuestos (SoV y TTRS) permiten analizar y comparar estrategias con diferentes lazos de control, evaluando su impacto tanto en la productividad como en la calidad geométrica. Sin embargo, sólo en el caso de simulación con SoV se ha contemplado el análisis de estrategias. La simulación con TTRS requeriría de una nueva formulación que realizara el cálculo de las correcciones a implementar, que no se ha abordado porque excede alcance establecido para esta investigación.

Por último, es necesario mencionar que el caso propuesto en el Capítulo 8 permite poner a prueba la metodología propuesta, tanto en cuanto al procedimiento como al uso de los lenguajes seleccionados y los recursos desarrollados. A pesar de que el experimento propuesto aborda un caso muy simple, limitado al análisis 2D y a procesos de ensamble, se ha querido limitar el alcance o la complejidad tratada para poner el foco de atención en la propuesta metodológica y la aplicación de los recursos propuestos (perfiles, librerías, transformaciones), evitando perderse en explicaciones más detalladas propias de casos más complejos o específicos. Si atendemos a los resultados y conclusiones del Capítulo 3, esto no debe ser entendido como una debilidad de la propuesta, que es igualmente capaz de abordar este tipo de análisis, como se pretende explorar en trabajos futuros.

9.2. Trabajos futuros

A continuación, se plantean algunas líneas de investigación que se prevé explorar en trabajos futuros. En primer lugar, es deseable ampliar los supuestos abordados en esta investigación y desarrollar la simulación de otros procesos de fabricación, como es el mecanizado, particularizando los lenguajes propuestos y ampliando las librerías desarrolladas. Esta línea de trabajo incluye también el análisis de estrategias de control, en las que hacer uso de las desviaciones modeladas en TTRS para el cálculo de las correcciones y ajustes que mejoren la calidad geométrica del producto simulado.

Una segunda línea de trabajo pretende dar continuidad a la propuesta, profundizando en la definición de modelos de especificación del sistema referente y estableciendo relaciones de dependencia con los modelos de análisis. Esta integración de modelos no solo está orientada al aseguramiento de la consistencia, sino que también pretende aprovechar gran parte de la información de especificación como base para la construcción del modelo de análisis. Aunque estos mecanismos se han explorado, por ejemplo, en la relación entre Feature y representación TTRS, se pretende ampliar y extender a la definición completa del sistema referente (sistema PPR).

Por otra parte, puede ser de gran interés en investigar cómo transformar modelos SysML orientados a la simulación a modelos texto Modelica sin que se requiera de la construcción de elementos de librería predefinidos no incluidos en el estándar Modelica. La automatización de los mecanismos de transformación generales permitiría obtener las librerías Modelica a partir de su definición en SysML, asegurando su consistencia y disminuyendo el tiempo de desarrollo. Adicionalmente, y visto el gran potencial de la automatización de transformaciones, se propone profundizar en el desarrollo de otro tipo de transformaciones para facilitar el uso combinado del modelado SysML y otras herramientas CAx, particularmente las CAT (Computer Aided Tolerancing).

Por último, una línea de investigación más ambiciosa consistiría en definir una ontología de dominio que contemple los conceptos manejados en los metamodelos propuestos, mediante el uso de un lenguaje ontológico, por ejemplo, OWL (Ontology Web Language). Esto permitiría validar la consistencia a nivel de instancias (individuos en un modelo ontológico) mediante el uso de razonadores. Adicionalmente, estos razonadores también permitirían comprobar la validez de los conceptos descritos e inferir nuevos conocimientos sobre el dominio, estableciendo nuevas relaciones y restricciones lógicas.

REFERENCIAS

[1] E. Hozdić, «Smart factory for industry 4.0: A review,» Journal of Modern Manufacturing Systems and Technology. 7, pp. 28-35, 2015.

[2] A. Pereira y F. Romero, «A review of the meanings and the implications of the Industry 4.0 concept,» Procedia Manufacturing. 13., pp. 1206-1214, 2017.

[3] O. Sauer, «Information Technology for the Factory of the Future – State of the Art and Need for Action,» Procedia CIRP, vol. 25, pp. 293-296, 2014.

[4] A. Barnard Feeney, S. Frechette y V. Srinivasan, « portrait of an ISO STEP tolerancing standard as an enabler of smart manufacturing systems,» Journal of Computing and Information Science in Engineering, vol. 15, 2014.

[5] O. Cardin, D. Trentesaux y A. e. a. Thomas, «Coupling predictive scheduling and reactive control in manufacturing hybrid control architectures: state of the art and future challenges,» J Intell Manuf, vol. 28, p. 1503–1517 , 2017.

[6] S. M. Saad, R. Bahadori, C. Bhovar y H. Zhang, «Industry 4.0 and Lean Manufacturing – a systematic review of the state-of-the-art literature and key recommendations for future research,» International Journal of Lean Six Sigma, 2023.

[7] F. Psarommatis, G. May, P.-A. Dreyfus y D. Kiritsis, «Zero defect manufacturing: state-of-the-art review, shortcomings and future directions in research,» International Journal of Production Research, vol. 58, nº 1, pp. 1-18, 2020.

[8] M. Colledani, T. Tolio, A. Fischer, B. Iung, G. Lanza, R. Schmitt y J. Váncza, «Design and management of manufacturing systems for production quality,» CIRP Annals, vol. 63, nº 2, pp. 773-796, 2014.

[9] K. Henderson y A. Salado, «Value and benefits of model-based systems engineering (MBSE): Evidence from the literature,» Systems Engineering, vol. 24, 2021.

[10] M. Bortolini, F. G. Galizia y C. Mora, «Reconfigurable manufacturing systems: Literature review and research trend,» Journal of Manufacturing Systems, vol. 49, pp. 93-106, 2018.

[11] F. Tao, Q. Qi, L. Wang y A. Nee, «Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0 correlation and comparison,» Engineering 5.4, vol. 5, pp. 653-661, 2019.

[12] Y. Qin, W. Lu, Q. Qi, T. Li, M. Huang, P. Scott y X. Jiang, «Explicitly representing the semantics of composite positional tolerance for patterns of holes,» International Journal of Advanced Manufacturing Technology, vol. 90, pp. 2121-2137, 2017.

[13] N. Anwer, B. Schleich, L. Mathieu y S. Wartzack, «From solid modelling to skin model shapes: Shifting paradigms in computer-aided tolerancing,» CIRP Annals - Manufacturing Technology , vol. 63, nº 1, pp. 137-140, 2014.

[14] D. D. M. Ferreira, S. Patalano, S. Gerbino, F. Mhenni y J.-Y. Choley, «A hierarchical set of SysML Model-based objects for tolerance specification,» de IEEE International Symposium, EDINBURGH, United Kingdom., 2016.

[15] R. Barbedienne, O. Penas, J.-Y. Choley, A. Rivière y F. Monica, «Introduction of geometrical contraints modeling in SysML for mechatronic design,» de 8th Europe-Asia Congress on Mecatronics, Tokyo, 2014.

[16] G. Bruscas-Bellido, Modelo basado en Elementos Característicos para la Planificación Supervisora de la Inspección. Tesis doctoral, 2015.

[17] A. Abdul Kadir, X. Xu y E. Hämmerle, «Virtual machine tools and virtual machining—A technological review,» Robotics and Computer-Integrated Manufacturing, vol. 27, pp. 494-508, 2011.
[18] L. Solano, F. Romero y P. Rosado, «An ontology for integrated machining and inspection process planning focusing on resource capabilities,» International Journal of Computer Integrated Manufacturing, vol. 29, pp. 1-15, 2015.
[19] J. V. Abellan-Nebot, J. Liu y F. Romero, «Design of multi-station manufacturing processes by integrating the stream-of-variation model and shop-floor data,» Fournal of Manufacturing Systems, vol. 30, nº 2, pp. 70-82, 2011.
[20] S. Zhou, Q. Huang y J. Shi, «State space modelling of dimensional variation propagation in multistage machining process using differential motion vectors,» IEEE Transactions on robotics and automation, vol. 19, nº 2, pp. 296-309, 2003.
[21] J. Liu, J. Jin y J. Shi, «State space modelling for 3-D variation propagation in rigid-body multistage assembly processes,» IEEE Transactions on Automation Science and Engineering, vol. 7, nº 2, pp. 274-290, 2009.
[22] Sanfilippo, E.; Benavent, S. et al., «Modeling Manufacturing Resources: An Ontological Approach», de 15th IFIP WG 5.1 International Conference, 2018.
[23] S. Benavent-Nácher, P. Rosado, L. Solano, N. Guarino y E. Sanfilippo, «How to Restructure PPDRC and MIRC According to DOLCE,» Procedia Manufacturing, vol. 28, pp. 195-200, 2019.
[24] F. Romero, E. M. Sanfilippo, P. Rosado, S. Borgo y S. Benavent, «Feature in product engineering with single and variant design approaches. A comparative review,» Procedia Manufacturing, vol. 41, pp. 328-335, 2019.
[25] S. Benavent-Nácher, F. Romero y P. Rosado, «Metodología basada en SysML y Modelica para la simulación de sistemas de fabricación,» de II Congreso Internacional Online sobre Tecnología e Ingeniería, 2020.
[26] S. Benavent-Nácher, P. Rosado y F. Romero, «Control strategies comparison for a multi-stage assembly system using simulation,» de 9th Manufacturing Engineering Society International Conference, 2021.
[27] D. Aguilella-Antolí, P. Rosado-Castellano y S. Benavent-Nácher, «A Modelica Library to Simulate Geometrical and Dimensional Deviations in Process Assemblies,» de 10th Manufacturing Engineering Society International Conference, 2023.
[28] F. Romero, P. Rosado, G. Bruscas-Bellido y S. Benavent, «Feature-Based Framework for Inspection Process Planning,» Materials, vol. 11, nº 9, p. 1504, 2018.
[29] S. Benavent-Nácher, P. Rosado y F. Romero, «Revisión del estado actual y perspectivas futuras sobre el modelado de máquinas herramienta,» 3C Tecnología, vol. 7, nº 3, 2018.
[30] S. Benavent-Nácher, P. Rosado y F. Romero, «Multidomain Simulation Model for Analysis of Geometric Variation and Productivity in Multi-Stage Assembly Systems,» Applied Sciences, vol. 10, nº 18, p. 6606, 2020.
[31] S. Benavent-Nácher, P. Rosado, F. Romero y J. Abellán-Nebot, «SYSML4TA: A SysML Profile for Consistent Tolerance Analysis in a Manufacturing System Case Application,» Applied Sciences, vol. 13, nº 6, p. 3794, 2023.
[32] T. Johnson, A. Kerzhner, C. J. J. Paredis y R. Burkhart, «ntegrating Models and Simulations of Continuous Dynamics Into SysML,» ASME Journal of Computing Information Science in Engineering, vol. 12, nº 1, 2011.
[33] A. Akundi y V. Lopez, «A Review on Application of Model Based Systems Engineering to Manufacturing and Production,» Procedia Computer Science, vol. 185, pp. 101-108, 2021.
[34] J. Schmidt y R. Taylor, Simulation and Analysis of Industrial Systems, R. D. Irwin, 1970.

[35] M. E. Kramer, Specification languages for preserving consistency between models of different languages, KIT Scientific Publishing, 2017.
[36] J. Miller y J. Mukerji, «Model Driven Architecture (MDA), ormsc/2001-07-01,» Object Management Group (OMG), 2001.
[37] S. Lajili, S. Hammoudi, O. Camp y M. Gammoudi, «Model Driven Development of Context-Aware Mobile Applications: An architecture and a Metamodel,» de Proceedings of the 4th International Conference on Software and Data Technologies, Sofia (Bulgaria), 2009.
[38] J. Bézivin, «On the unification power of models,» Software & Systems Modeling, vol. 4, pp. 171-188, 2005.
[39] S. Friedenthal, A. Moore y R. Steiner, Practical guide to SysML, Morgan Kaufmann, 2012.
[40] J. Estefan, Survey of Model-Based Systems Engineering (MBSE) Methodologies, INCOSE MBSE Focus Group, 2008.
[41] P. Fritzson, Principles of Object Oriented Modeling and Simulation with Modelica 2.1, Wiley IEEE Press, 2004.
[42] M. Eigner, T. Gilz y R. Zafirov, Interdisciplinary Product Development - Model Based Systems Engineering., 2012.
[43] A. M. Law, Simulation Modeling and Analysis, Fifth Edition, Mc Graw Hill, 2015.
[44] P. Fritzson, «The Modelica Object-Oriented Equation-Based Language and Its OpenModelica Environment with MetaModeling, Interoperability, and Parallel Execution,» Simulation, Modeling, and Programming for Autonomous Robots. , vol. 6472, 2010.
[45] C. Sinnwell, N. Krenkel y Jan C. Aurich, «Conceptual manufacturing system design based on early product information,» CIRP Annals, vol. 68, nº 1, pp. 121-124, 2019.
[46] A. Reichwein y C. Paredis, «Overview of Architecture Frameworks and Modeling Languages for Model-Based Systems Engineering,» Proceedings of the ASME Design Engineering Technical Conference, 2011.
[47] W. Torres, M. van den Brand y A. Serebrenik, «A systematic literature review of cross-domain model consistency checking by model management tools.,» Softw Syst Model, vol. 20, p. 897–916, 2021.
[48] S. Feldmann, K. Kernschmidt, M. Wimmer y B. Vogel-Heuser, «Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering,» Journal of Systems and Software, vol. 153, nº 105-134, 2019.
[49] A. Qamar, C. Paredis, J. Wikander y C. During, «Dependency Modeling and Model Management in Mechatronic Design,» Journal of Computing and Information Science in Engineering, vol. 12, nº 4, 2012.
[50] G. e. a. Höpfner, «Höpfner, Gregor & Jacobs, Georg & Zerwas, Thilo & Nachmann, Imke & Berroth, Joerg & Guist, Christian & Rumpe, Bernhard & Kohl, Jens. (2021). Model-Based Design Workflows for Cyber-Physical Systems Applied to an Electric-Mechanical Coolant Pump. IOP Confer,» de 19th Drive Train Technology Conference, Aachen (Alemania), 2021.
[51] H. Wagner y C. Zuccaro, «Collaboration between System Architect and Simulation Expert,» de IEEE International Symposium on Systems Engineering, Viena (Austria), 2022.
[52] K. Kernschmidt, G. Barbieri, C. Fantuzzi y B. Vogel-Heuser, «Possibilities and challenges of an integrated development using a combined,» IFAC Proceedings Volumes, vol. 46, nº 9, pp. 1465-1470, 2013.
[53] V. Srinivasan, «Computational Metrology for the Design and Manufacture,» ournal of Computing, vol. 7, nº 1, pp. 3-9, 2007.

[54] ASME, Dimensioning and Tolerancing. Y14.5, 2019.
[55] ISO-17450-1, Geometrical product specifications (GPS) -- General concepts -- Part 1: Model, 2011.
[56] M. Baysal, S. Rachuri, R. Sriram y K. Lyons, «The Open Assembly Model for the Exchange of Assembly and Tolerance Information: Overview and Example,» Mechanical and Aerospace Engineering., 2004.
[57] S. F. C. B. R. B. R. D. S. Steven J. Fenves, CPM 2: A Revised Core Product Model for Representing Design Information. NIST Interagency/Internal Report (NISTIR), National Institute of Standards and Technology, 2005.
[58] M. M. Baysal, Functional and Behavioral Product Information Representation and Consistency. Tesis doctoral, 2012.
[59] P. e. a. Grauberger, «The contact and channel approach – 20 years of application experience in product engineering,» Journal of Engineering Design, vol. 31, pp. 241-265, 2020.
[60] A. &. R. A. &. S. P. &. V. C. (. .. 1.-1. 1.-1.-4.-5.-5. Clément, «The TTRS: 13 Constraints for Dimensioning and Tolerancing,» Geometric Design Tolerancing: Theories, Standards and Applications, p. 122–131, 1998.
[61] Y. Cao, Y. Liu, J. & Mao y J. Yang, «3DTS: A 3D tolerancing system based on mathematical definition.,» Journal of Zhejiang University: Science , vol. 7, pp. 1810-1818, 2006.
[62] ISO-10303, Industrial automation systems and integration product data representation and exchange., International Organization for Standardization, 2014.
[63] N. Anwer, B. Schleich, L. Mathieu y S. Wartzack, «From solid modelling to skin model shapes: Shifting paradigms in computer-aided tolerancing,» CIRP Annals - Manufacturing Technology , vol. 63, nº 1, pp. 137-140, 2014.
[64] Y. Qin, Q. Qi, W. Lu, X. Liu, P. Scott y X. Jiang, «A review of representation models of tolerance information,» International Journal of Advanced Manufacturing Technology., vol. 95, p. 2193–2206, 2018.
[65] K. Ogata y Y. Yang, Modern control engineering, Pearson Education, Inc, 2012.
[66] J. Shi, «Stream of variation modelling and analysis for multistage manufacturing processes,» CRC press,, 2006.
[67] J. Jin y J. Shi, «State space modeling of sheet metal assembly for dimensional control,» ASME. J. Manuf. Sci. Eng., vol. 121, nº 4, p. 756–762, 1999.
[68] J. Liu, J. Shi y S. J. Hu., «Quality-assured setup planning based on the Stream-of-Variation model for multi-stage machining processes,» International Manufacturing Science and Engineering Conference, vol. 47624, 2006.
[69] C. Q. Liu, Y. Ding y Y. Chen, « Optimal coordinate sensor placements for estimating mean and variance components of variation sources,» IIE Transactions, vol. 9, nº 877-889, p. 37, 2005.
[70] J. e. a. Barhak, «Integration of reconfigurable inspection with stream of variations methodology.,» International Journal of Machine Tools and Manufacture, vol. 45, nº 4-5, pp. 407-419, 2005.
[71] J. e. a. Abellán-Nebot, «Optimal inspection/actuator placement for robust dimensional compensation in multistage manufacturing processes,» Computational Methods and Production Engineering., pp. 31-50, 2017.
[72] J. V. Abellan-Nebot, J. Liu y F. Romero, «Quality prediction and compensation in multi-station machining processes using sensor-based fixtures,» Robotics and Computer-Integrated Manufacturing, vol. 28, nº 2, pp. 208-219, 2012.
[73] D. Djurdjanovic y J. Ni, «Online stochastic control of dimensional quality in multistation manufacturing systems,» Proceedings of the Institution of Mechanical

Engineers, Part B: Journal of Engineering Manufacture, vol. 221, nº 5, pp. 865-880, 2007.
[74] ISO 10303-2, Industrial automation systems and integration. Product data representation and exchange. Part 2: Vocabulary, International Organization for Standardization.
[75] ISO_19439:2006, Enterprise integration. Framework for enterprise modelling, International Organization for Standardization, 2006.
[76] ISO-15531-1:2004, Industrial automation systems and integration. Industrial manufacturing management data. Part 1: General overview, International Organization for Standardization, 2004.
[77] A. &. Y. R. &. M. J.-J. &. G. R. &. C. J. &. B. j.-p. Cutting-Decelle, «ISO 15531 MANDATE: A Product-process-resource based Approach for Managing Modularity in Production Management.,» Current Engineering: R&A, vol. 15, pp. 217-235, 2007.
[78] N. P. Suh, D. S. Cochran y P. C. Lima, «Manufacturing System Design,» Annals of the ClRP, vol. 47, nº 2, 1998.
[79] B. Ramis Ferrer, W. Mohammed y M. e. a. Ahmad, «Comparing ontologies and databases: a critical review of lifecycle engineering models in manufacturing,» Knowl Inf Syst, vol. 63, p. 1271–1304, 2021.
[80] Y. Lu, K. Morris y S. Frechette, Current Standards Landscape for Smart Manufacturing Systems. Nat. Inst. Stand. Technol.., National Institute of Standards and Technology, 2016.
[81] H. Rampersad, Integrated and Simultaneous Design for Robotic Assembly: Product Development, Planning., John Wiley & Sons Ltd, 1994.
[82] Mussawar Ahmad et al., «Knowledge-based PPR modelling for assembly automation,» CIRP Journal of Manufacturing Science and Technology, vol. 21, nº 33-46, 2018.
[83] T. Tolio et al., «SPECIES—Co-evolution of products, processes and production systems,» CIRP Annals, vol. 59, nº 2, pp. 672-693, 2010.
[84] A. Jayal, F. Badurdeen, O. Dillon y I. Jawahir, «Sustainable manufacturing: Modeling and optimization challenges at the product, process and system levels,» CIRP Journal of Manufacturing Science and Technology, vol. 2, nº 3, pp. 144-152, 2010.
[85] K. Agyapong-Kodua, C. Haraszkó y I. Németh, «Recipe-based Integrated Semantic Product, Process, Resource (PPR) Digital Modelling Methodology,» Procedia CIRP, vol. 17, pp. 112-117, 2014.
[86] L. Kathrein, K. Meixner, D. Winkler, A. Luder y S. Biffl, «A Meta-Model for Representing Consistency as Extension to the Formal Process Description,» de 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2019.
[87] M. U. A. K. Y. e. a. Mehrabi, «Trends and perspectives in flexible and reconfigurable manufacturing systems,» Journal of Intelligent Manufacturing , vol. 13, p. 135–146, 2002.
[88] Y. G. X. &. G. W. Koren, «Reconfigurable manufacturing systems: Principles, design, and future trends,» Front. Mech. Eng., vol. 13, p. 121–136, 2018.
[89] J. Serey, M. Alfaro, G. Fuertes, M. Vargas, R. Ternero, C. Duran, J. Sabattin y S. Gutierrez, «Framework for the Strategic Adoption of Industry 4.0: A Focus on Intelligent Systems,» Processes, vol. 11, p. 2973, 2023.
[90] OMG, OMG Systems Modeling Language (OMG SysML), Version 1.6. formal/19-11-01, Object Management Group, 2019.
[91] L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modeling Language, Addison-Wesley Professional, 2013.
[92] OMG, XML Metadata Interchange, Version 2.5.1, Object Management Group (OMG), 2015.

[93] OMG, Unified Modeling Language (UML), Version 2.5.1, Object Management Group (OMG), 2017.
[94] OMG, OMG Meta Object Facility (MOF) Core Specification, Version 2.5.1, Object Management Group (OMG), 2019.
[95] OMG, Object Constraint Language (OCL), Version 2.4, Object Management Group (OMG), 2014.
[96] OMG, OMG System Modeling Language, Version 2.0 beta, Object Management Group, 2023.
[97] Open Source Modelica Consortium (OSMC), «Modelica Association,» [En línea]. Available: https://modelica.org/association/. [Último acceso: 2023].
[98] M. Association, «Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.7,» 2023.
[99] A. Elsheikh y P. Palensky, «Modelica by Application Power Systems V1.1,» 2021.
[100] P. Fritzson, «Introduction to Object-Oriented Modeling and Simulation with OpenModelica,» 2012.
[101] C. Paredis, Y. Bernard, R. Burkhart, H. P. de Koning, S. Friedenthal, P. Fritzson, N. Rouquette y W. Schamai, «An Overview of the SysML-Modelica Transformation Specification,» de INCOSE International Simposium, 2010.
[102] D. Systemes, «Dymla,» [En línea]. Available: https://www.3ds.com/products-services/catia/products/dymola/.
[103] ANSYS, «Introduction to Ansys Simplorer,» [En línea]. Available: https://www.ansys.com/training-center/course-catalog/electronics/introduction-to-ansys-simplorer.
[104] Altair, «Control software solidThinking Activate,» [En línea]. Available: https://www.directindustry.com/prod/altair/product-5874-1760808.html.
[105] Wolfram, «Wolfram SystemModeler,» [En línea]. Available: https://www.wolfram.com/system-modeler/.
[106] V. E. Software, «Simcenter STAR-CCM+,» [En línea]. Available: https://volupe.se/products/simcenter-star-ccm/?gclid=CjwKCAiAu9yqBhBmEiwAHTx5pyfLdQohwxStF7f9zuSYOWrjdcv1xlD9jtRFDgt5orJs77MwnbQ8PhoCSZoQAvD_BwE.
[107] Open Source Modelica Consortium (OSMC), «OpenModelica,» [En línea]. Available: https://openmodelica.org/. [Último acceso: 2023].
[108] Open Source Modelica Consortium (OSMC), «ModelicaML - A UML Profile for Modelica,» [En línea]. Available: https://openmodelica.org/free-and-open-source-software/modelica-modeling-language-modelicaml/.
[109] Kern, W. et al., «Alternatives to assembly line production in the automotive industry.,» de 23rd International 798 Conference on Production Research, 2015.
[110] F. Cellier y E. Kofman, Continuous system simulation, Springer Science & Business Media, 2006.
[111] P. T. Beery, A model-based systems engineering methodology for employing architecture in system analysis (Tesis doctoral), 2016.
[112] M. Meißnera, G. Jacobs, P. Jagla y J. Sprehe, «Model based systems engineering as enabler for rapid engineering change management,» de 31st CIRP Design Conference, 2021.
[113] L. Bassi, C. Secchi, C. Fantuzzi y M. Bonfe, «An object-oriented approach to manufacturing systems modeling,» IEEE International Conference on Automation Science and Engineering, pp. 442-447, 2006.
[114] D. Systemes, «Cameo Systems Modeler,» [En línea]. Available: https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/.

[115] D. Systemes, «MagicDraw,» [En línea]. Available: https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/.
[116] Microsoft, «Visio help and learning,» [En línea]. Available: https://support.microsoft.com/en-us/visio.
[117] V. Paradigm, «Visual Paradigm Community Edition - A Free UML Tool,» [En línea]. Available: https://www.visual-paradigm.com/solution/freeumltool/.
[118] OMG, MOF Query/View/Transformation (QVT), Version 1.3, Object Management Group, 2016.
[119] H. Peng, Z. Peng y Z. Zhou, «Manufacturing variation modeling and process evaluation based on small displacement torsors and functional tolerance requirements,» Journal of Advanced Mechanical Design, Systems, and Manufacturing, vol. 15, nº 3, p. 819, 2021.
[120] P. Fonseca, «Introducción a la simulación,» [En línea]. Available: https://openaccess.uoc.edu/bitstream/10609/142846/24/PLA6_Introducci%C3%B3n%20a%20la%20simulaci%C3%B3n.pdf. [Último acceso: 2023].
[121] D. A. Bodner y L. F. McGinnis, «A Structured Approach to Simulation Modeling of Manufacturing Systems,» de Proceedings of the 2002 Industrial Engineering Research Conference , 2002.
[122] S. Bifll, A. Lüder y D. Gerhard, Multidisciplinary Enginering for Ciber-Physical Production System, S. Biffl, A. Lüder y D. Gerhard, Edits., Springer Cham, 2017.
[123] J. Miller, G. Baramidze, A. Sheth y P. Fishwick, Ontologies for Modeling and Simulation: Initial Framework., 2004.
[124] Jagla, P.; et al., «Using SysML to Support Impact Analysis on Structural Dynamics Simulation Models,» Procedia CIRP, vol. 100, 2021.
[125] Brahmi, R. et al, «nteroperability of CAD models and SysML specifications for the automated checking of design requirements,» Procedia CIRP, vol. 100, pp. 259-264, 2021.
[126] A. Berriche, F. Mhenni, A. Mlika y J.-Y. Choley, «Towards Model Synchronization for Consistency Management of Mechatronic Systems.,» Applied Sciences, vol. 10, p. 3577, 2020.
[127] A. Albers y C. Zingel, «Interdisciplinary Systems Modeling using the Contact & Channel-Model for SysML,» de In Proceedings of the 18th International Conference on Engineering Design, 2011.
[128] A. Albert y C. Zingel, «Extending SysML for Engineering Designers by Integration of the Contact & Channel – Approach (C&C2-A) for Function-Based Modeling of Technical Systems,» Procedia Computer Science, vol. 16, pp. 353-362, 2013.
[129] A. &. W. E. Albers, «The Contact and Channel Approach (C&C2-A) - relating a system's physical structure to its functionality.,» de An Anthology of Theories and Models of Design, 2014, p. 151–171.
[130] K. Kernschmidt, «Interdisciplinary structural modeling of mechatronic production systems using SysML4Mechatronics (Tesis doctoral),» 2019.
[131] Nachmann, I. et al., «Modeling mechanical functional architectures in SysML.,» de 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, 2020.
[132] F. D. Monica, S. Patalano, J. Y. Choley, F. Mhenni y S. Gerbino, «A hierarchical set of SysML Model-based objects for tolerance specification,» de IEEE International Symposium on Systems Engineering (ISSE), 2016.
[133] Nikolaidou, M. et al. , «Simulating SysML models: Overview and challenges,» de 10th System of Systems Engineering Conference, 2015.

[134] Kawahara, R. et al, «erification of embedded system's specification using collaborative simulation of SysML and Simulink models.,» de Model-Based Systems Engineering International Conference , 2009.
[135] Sakairi, T. et al., «Designing a Control System using SysML and Simulink.,» de Proceedings of SICE Annual Conference., 2012.
[136] W. Schamai, P. Fritzson, C. Paredis y A. Pop, «Towards Unified System Modeling and Simulation with ModelicaML Modeling of Executable Behavior Using Graphical Notations,» de Proceedings of the 7th International Modelica Conference, 2009.
[137] A. &. A. D. &. F. P. Pop, «Towards Unified System Modeling with the ModelicaML UML Profile,» de Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, 2007.
[138] OMG, SysML-Modelica Transformation, Version 1.0, Object Management Group, 2012.
[139] Paredis, C. et al., «An Overview of the SysML-Modelica Transformation Specification. INCOSE International S,» de INCOSE International Symposium, 2012.
[140] A. Votintsev, P. Witschel, N. Regnat y P. E. Stelzig, «Comparative Study of Model-Based and Multi-Domain System Engineering Approaches for Industrial Settings,» de Modelling Foundations and Applications, 2012.
[141] P. Vasaiely, Interactive Simulation of SysML Models using Modelica (Tesis doctoral), 2009.
[142] J.-M. Gauthier, F. Bouquet, A. Hammad y F. Peureux, «Tooled Process for Early Validation of SysML Models Using Modelica Simulation,» de Fundamentals of Software Engineering, 2015.
[143] Eclipse, «ATL - a model transformation technology,» [En línea]. Available: https://eclipse.dev/atl/.
[144] «Kermeta,» [En línea]. Available: https://diverse-project.github.io/k3/index.html.
[145] Eclipse, «WTrend Introduction,» [En línea]. Available: https://eclipse.dev/Xtext/xtend/documentation/index.html.
[146] «Eclipse Epsilon,» [En línea]. Available: https://eclipse.dev/epsilon/.
[147] Eclipse, «The Epsilon Object Language (EOL),» [En línea]. Available: https://eclipse.dev/epsilon/doc/eol/.
[148] Eclipse, «The Epsilon Comparison Language (ECL),» [En línea]. Available: https://eclipse.dev/epsilon/doc/ecl/.
[149] Eclipse, «The Epsilon Merging Language (EML),» [En línea]. Available: https://eclipse.dev/epsilon/doc/eml/.
[150] Eclipse, «The Epsilon Validation Language (EVL),» [En línea]. Available: https://eclipse.dev/epsilon/doc/evl/.
[151] Eclipse, «The Epsilon Generation Language (EGL),» [En línea]. Available: https://eclipse.dev/epsilon/doc/egl/.

ANEXOS

Anexo A

A continuación, se presenta la especificación de los estereotipos definidos en el perfil SysML for Manufacturing System Simulation (SysML4MSSim).

«BehavioralElement_sim»
Estereotipo abstracto que permite identificar los bloques que forman parte del sistema de simulación (o el sistema de simulación en su conjunto) y que tienen un comportamiento.

Extiende/especializa a: «Block»
Atributo:
- isAtomic : Boolean

Reglas:

C1. Si se trata de un elemento atómico (isAtomic=true), debe ser activo, tener definido un comportamiento propio de tipo «STM_MainBehavior» StateMachine, y no puede poseer ninguna propiedad tipeada por un bloque estereotipado por una especialización de *«BehavioralElement_sim»*.

```
isAtomic=true implies (self.isActive and self.classifierBehavior()->
select(a|a.oclIsKindOf( UML:: StateMachine)).oclAsType(StateMachine).
getAppliedStereotypes().name->includes( 'STM_MainBehavior') and not
self.base_Class.allAttributes()-> select(b|b.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().allPa
rents().name->includes( 'BehavioralElement_sim')
```

C2. Si se trata de un elemento complejo (isAtomic=false), no puede tener un comportamiento propio de tipo «STM_MainBehavior» StateMachine.

```
isAtomic=false implies (self.isActive=false and not self.
classifierBehavior()-> select(a|a.oclIsKindOf( UML:: StateMachine)).
oclAsType(StateMachine). getAppliedStereotypes().name->includes(
'STM_MainBehavior') and self.base_Class.allAttributes()->
select(c|c.type.oclIsKindOf(UML:: Class)).type.oclAsType(UML::Class).
getAppliedStereotypes().allParents().name->includes(
'BehavioralElement_sim')
```

«STM_MainBehavior»
Especialización de la metaclase <StateMachine> que se utiliza para definir el comportamiento propio de un «BehavioralElement_sim» Block de tipo atómico.

Extiende/especializa a: <StateMachine>

«SimulationSystem4MS»
Especialización de «BehavioralElement_sim» que sirve para estereotipar el bloque que representa el sistema de simulación en su conjunto, definido para analizar un sistema de fabricación (MS).

Extiende/especializa a: *«BehavioralElement_sim»*
Reglas:

C3. Un «SimulationSystem4MS» Block está compuesto por al menos un uso de cada uno de los siguientes bloques: «SimConfigurator» Block, «InputGenerator» Block y «SimulatedSystem» Block.

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType( UML::Class).getAppliedStereotypes().
name->includes('SimConfigurator') and self.base_Class.allAttributes()
-> select(a|a.type.oclIsKindOf( UML::Class)).type.oclAsType(
UML::Class).getAppliedStereotypes().name->includes('SimConfigurator')
and self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType( UML::Class).getAppliedStereotypes().
name->includes('SimulatedSystem')
```

«Configurator»
Bloque que contiene el conjunto de parámetros necesarios para definir la configuración de la simulación. Algunos de estos parámetros son constantes, pero otros pueden variar a lo largo del tiempo simulado, con lo que este bloque podría incluir ConstraintBlocks o comportamientos (máquinas de estado, por ejemplo) para gobernar dicha evolución de los parámetros. Además, puede incluir referencias (shared relationships) a los modelos de especificación, de los cuales extraer algunos parámetros.

Extiende/especializa a: «Block»

«InputGenerator»
Especialización del «BehavioralElement_sim» para definir bloques que generan las entradas o estímulos para excitar el sistema simulado. En el caso de la simulación de MS, estos elementos generan los datos necesarios para simular las unidades materiales (piezas, ensambles, lotes de producto, etc.) que fluyen y se transforman en un bloque «MS_sim».

Extiende/especializa a: *«BehavioralElement_sim»*
Reglas:

C4. Un «InputGenerator» Block posee al menos un puerto «FU_Port» y un puerto «C_Port» Ports (aunque también pueden estar aplicados a un mismo puerto), para establecer conexiones con usos de «MS_sim» Blocks y soportar la trasmisión de unidades de flujo y la sincronización de comportamientos.

```
self.base_Class.allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
name->includes('FU_Port') and self.base_Class.allAttributes() ->
select(b|b.type.oclIsKindOf( UML::Class)).type.oclAsType(
UML::Class).getAppliedStereotypes(). name->includes('C_Port')
```

«OutputCollector»
Especialización del «BehavioralElement_sim» para definir bloques encargados de recopilar y almacenar las salidas generadas por el sistema simulado, en este caso de un bloque «MS_sim».

Extiende/especializa a: *«BehavioralElement_sim»*

Reglas:

C5. Un «OutputCollector» Block posee al menos un puerto «FU_Port» y un puerto «C_Port» Ports (aunque también pueden estar aplicados a un mismo puerto), para establecer conexiones con usos de «MS_sim» Blocks y soportar la trasmisión de unidades de flujo y la sincronización de comportamientos.

```
self.base_Class.allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
name->includes('FU_Port') and self.base_Class.allAttributes() ->
select(b|b.type.oclIsKindOf( UML::Class)).type.oclAsType(
UML::Class).getAppliedStereotypes(). name->includes('C_Port')
```

«ResultsManager»

Bloque encargado de gestionar y tratar matemáticamente los resultados de la simulación, no sólo a partir de las salidas del sistema simulado (obtenidas de usos del «SimulatedSystem» o «OutputCollector» Blocks), sino de variable de cualquier otro elemento del sistema de simulación (como usos del «InputGenerator» Blocks, por ejemplo). Dicho tratamiento matemático queda definido mediante ConstraintBlocks.

Extiende/especializa a: «Block»

Reglas:

C6. Un «ResultsManager» Block posee al menos un «DataPort» Ports para establecer conexiones con otros elementos de un «SimulationSystem4MS» Block y soportar la transmisión de datos comportamientos.

```
self.base_Class.allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
name->includes('DataPort')
```

«DataPort»

Puerto que se utiliza para transmitir datos entre componentes del sistema de simulación.

Extiende/especializa a: <Port>

«FU_Port»

Puerto que se utiliza para dar soporte a la transmisión de datos entre componentes del sistema de simulación, emulando el flujo logístico.

Extiende/especializa a: «DataPort»

Atributo:

- direction : DirectionTypes

Reglas:

C7. Su tipo debe estar definido en un «FlowUnit» Block o por bloques estereotipados con especializaciones de «FlowUnit».

```
self.base_Port.type.oclAsType(UML::Class).getAppliedStereotypes().all
Parents().name->includes('FlowUnit') or self.base_Port.type.
oclAsType(UML::Class).getAppliedStereotypes()name->includes(
'FlowUnit')
```

«C_Port»

Puerto que se utiliza para dar soporte al flujo de control, la comunicación entre componentes del sistema de simulación y la sincronización de sus comportamientos.

Extiende/especializa a : «DataPort»

«FlowUnit»
Bloque que representa la principal unidad de flujo considerada en el sistema de simulación, y que se emplea principalmente para definir «FU_Port» Ports.

Extiende/especializa a: «Block»
Reglas:

C8. Debe estar compuesto por usos de un «Product_sim» Block

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
name->includes('Product_sim')
```

«ManufResource_sim»
Especialización de «BehavioralElement_sim» que identifica los recursos del sistema de fabricación y simula su comportamiento. Un recurso puede estar compuesto por otros recursos. Sin embargo, conviene identificar el nivel más atómico considerado, que en esta propuesta será la subfase. Para ello se define el atributo booleano "isSubphase".

Extiende/especializa a: *«BehavioralElement_sim»*
Atributos:

- isSubfase:Booleano

«ManufResourceSpecif_data»
Bloque se emplea para definir la información de especificación relativa a un determinado tipo de recursos de fabricación considerados en la simulación. Cuenta con una propiedad booleana (isTransformer) para identificar las especificaciones de recursos transformadores.

Extiende/especializa a: «Block»
Atributos:

- isTransformer : Boolean

Reglas:

C9. Si es un recurso transformador, debe tener al menos una propiedad «ConfigSpecif».

```
isTransformer=true implies (self.base_Class.allAtributes()->select(
a|a.type.oclIsKindOf(UML::Class)).type.oclAsType(UML::Class).
getAppliedStereotypes().name->includes('ConfigSpecif')
```

«ProcessingResource_sim»
Estereotipo abstracto que es una especialización del estereotipo «ManufResource_sim» para identificar los bloques en los que se define un recurso procesador, es decir, aquellas partes del sistema de simulación que representan recursos del sistema de fabricación y por las que fluyen unidades materiales o de producto.

Extiende/especializa a: «ManufResource_sim»
Reglas:

C10. Deben tener al menos dos puertos de tipo «FU_Port» Port

```
self.base_Class.allAtributes()->select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
select(b|b.name='FU_Port').size()>1
```

«TansformResource_sim»
Especialización del estereotipo «ProcessingResource_sim» que representa un recurso transformador, es decir, aquellos que modifican físicamente el producto entrante, de manera

que la definición del producto de entrada y de salida es forzosamente diferente. En esta propuesta se contemplan tanto recursos de mecanizado como de ensamble. Además, puede haber recursos que admitan diversas configuraciones, circunstancia en la que es necesario contar con una propiedad que indique el estado activo en cada instante.

Extiende/especializa a: «ProcessingResource_sim»
Reglas:

C11. Los «FU_Port» Ports definidos en el «ManufResourceSim» Block deben estar tipeados al menos por dos «FlowUnit» Blocks compuestos por usos de «Product_sim» Blocks diferentes, ya que los artefactos de la entrada y la salida deben ser diferente.

```
self.base_Class.allAtributes()->select(a|a.oclIsKindOf( UML::Port))
.type.first<> self.base_Class.allAtributes()->select(a|a.oclIsKindOf(
UML::Port)).type.second
```

C12. Debe poseer una «ActiveConfiguration» property.

```
self.base_Class.allAtributes().getAppliedStereotypes()name->
includes(ActiveConfiguration')
```

«ActiveConfiguration»
Propiedad de un «TransformResource_sim» Block que identifica la configuración actual del recurso de fabricación.

Extiende/especializa a: <Property>
Reglas:

C13. La propiedad es de tipo enumeración.

```
self.base_Property.type.oclIsTypeOf(UML::Enumeration)
```

«ConfigSpecif»
Propiedad de un «ManufResourceSpecif_data» Block que define las posibles configuraciones del recurso de fabricación.

Extiende/especializa a: <Enumeration>

«LogisticResource_sim»
Especialización del estereotipo *«ManufResourceSim»* que representa un recurso logístico del sistema de fabricación y simula su comportamiento. Un recurso logístico es aquel que no transforma físicamente el producto entrante, de manera que el artefacto de entrada y salida son forzosamente iguales.

Extiende/especializa a: *«ProcessingfResourceSim»*
Reglas:

C14. Los «FU_Port» Ports definidos en el «LogisticResourceSim» Block deben estar tipeados por el mismo «ProductBatchSim» Block, o por «ProductBatchSim» Blocks diferentes pero compuestos por usos de un mismo «ProductSim» Blocks y en un mismo estado.

```
self.base_Class.allAtributes()->select(a|a.oclIsKindOf( UML::Port))
.type.first = self.base_Class.allAtributes()->select(a|a.oclIsKindOf(
UML::Port)).type.second
```

«MS_sim»

Especialización del «TransformResource_sim» que se emplea para identificar el bloque que emula el sistema de fabricación en su conjunto en el sistema de simulación.

Extiende/especializa a: «TransformResource_sim»

«ControlResource_sim»
Estereotipo que especializa el «ManufResource_sim» para representar una unidad de control, incluyendo la monitorización, control y toma de decisiones. Para ello, debe estar conectado con otros elementos, mediante conexiones entre puertos específicos («DataPort» Ports.

Extiende/especializa a: «ManufResource_sim»
Reglas:

C15. Tiene al menos un «C_Port» Port.

```
self.base_Class.allAtributes()->select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().name
->includes('FU_Port')
```

«Product_sim»
Bloque que representa una unidad de producto que fluye por el «SimulatedMMS», simulando su fabricación.

Extiende/especializa a: «Block»
Reglas:

C16. Un «ProductSim» Block debe referenciar (shared relationship) a un «ProductSpecif_data» Block, que define las especificaciones del producto.

```
self.base_Class.allAtributes()->select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().name
->includes('ProductSpecif_data')
```

«ActiveState»
Propiedad de un «Product_sim» Block que identifica el estado de fabricación en el que se encuentra el producto simulado.

Extiende/especializa a: <Property>
Reglas:

C17. La propiedad es de tipo enumeración.

```
self.base_Property.type.oclIsTypeOf(UML::Enumeration)
```

«ProductSpecif_data»
Bloque que define las especificaciones de un determinado tipo de producto que se fabrica en el MMS simulado.

Extiende/especializa a: «Block»
Reglas:

C18. Un «ProductSpecif» Block debe poseer una actividad «ProcessPlan» que define el plan de procesos diseñado para la fabricación del producto

```
self.base_Class.ownedElement()->select(a|a.oclIsKindOf(
UML::Activity)).oclAsType(UML::Activity).getAppliedStereotypes().name
->includes('ProcessPlan')
```

«StateSpecif»
Propiedad de un «Product_sim» Block que define qué estado está activo en ese instante.

Extiende/especializa a: <Property>
Reglas:

C19. La propiedad es de tipo enumeración.

```
self.base_Property.type.oclIsTypeOf(UML::Enumeration)
```

«NativeProcessPlan»
Actividad mediante la cual se definen los diversos procesos por los que debe pasar un determinado tipo de producto durante su fabricación.

Extiende/especializa a: <Activity>
Reglas:

C20. Todas las acciones definidas deben ser de tipo «ManufProcess» Action.

```
self.base_Activity.ownedElement->select(a|a.oclIsKindOf(UML::
CallBehaviorAction)).size()= self.base_Activity.ownedElement->
select(b|b.oclIsKindOf(UML::CallBehaviorAction)).oclAsType(
CallBehaviorAction).getAppliedStereotypes()-> select(c|c.name=
'ManufProcess').size()
```

«ManufProcess»
Acción de un «NativeProcessPlan» Activity que representa un proceso de fabricación en el plan de producción nativo. Cuenta con un atributo booleano (isAtomic) para identificar los procesos atómicos, que en esta propuesta se referirá a las subfases, es decir, todas las operaciones de fabricación que se realizan en una misma máquina y un mismo amarre.

Extiende/especializa a: <CallBehaviorAction>
Atributos:

- isAtomic : Booleano

«ResourceAllocation»
Especialización del estereotipo «AllocateActivityPartition» de SysML empleado para asignar recursos a uno o más procesos del plan de procesos nativo.

Extiende/especializa a: «AllocateActivityPartition»
Reglas:

C21. Cada partición representa a un bloque estereotipado con una especialización de «ManufResource_sim».

```
self.base_ActivityPartition.oclAsType(UML::ActivityPartition).
represents.oclAsType(UML::Class).getAppliedStereotypes().alParents().
name->includes('ManufResource_sim'
```

Anexo B

A continuación, se presenta la especificación de los estereotipos definidos en el perfil SysML for Tolerance Analysis (SysML4TA).

«Artifact»
Artefacto, especificación de una entidad distintiva (componente) en un Sistema físico. Se trata de un estereotipo abstracto cuyas especializaciones son «Part» y «Assembly».

Extiende/especializa a: «Block»

«Part»
Pieza, especificación de un artefacto atómico que no puede ser descompuesto en otros artefactos.

Extiende/especializa a: «Artifact» («Block»)
Reglas:

C1. Un «Part» Block no puede poseer propiedades de tipo «Artifact» («Part» o «Assembly») Block.

```
not self.base_Class.ownedAttribute-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
allParents().name->includes('Artifact')
```

C2. Cualquier «Part» Block debe contener al menos una propiedad de tipo «FormFeature» o «F_TTRS_Repr» Block.

```
self.base_Class.ownedAttribute-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()
.name->includes('FormFeature') or self. base_Class.ownedAttribute->
select(e|e.type.oclIsKindOf(UML::Class )).type.oclAsType(UML::Class).
getAppliedStereotypes().name-> includes('F_TTRS_Repr')
```

«Assembly»
Ensamble, especificación de un artefacto complejo que está compuesto por al menos dos artefactos.

Extiende/especializa a: «Artifact» («Block»)
Reglas:

C3. Un «Assembly» Block está compuesto por al menos una propiedad de tipo «Artifact» («Part» or «Assembly») Blocks.

```
self.base_Class.ownedAttribute-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
allParents().name->includes('Artifact')
```

C4. Todas sus propiedades de tipo «Artifact» Block deben poseer al menos un «AssemblyFeature» or «AssemblyMGRE» Port para soportar sus relaciones de ensamble con otros artefactos.

```
self.base_Class.ownedAttribute-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes().
name->includes('AssemblyFeature') or self.base_Class.ownedAttribute->
select(e|e.type.oclIsKindOf(UML::Class)).type.oclAsType(UML::Class).
getAppliedStereotypes().name->includes('AssemblyMGRE')
```

«Feature»
Especificación de una porción de la forma de un artefacto.

Extiende/especializa a: «Block»

«FC_Specification»
Especificación de una condición funcional, definiendo los límites de la variabilidad aceptable de un feature objetivo (target) con respecto a algunos features de referencia (datums).

Extiende/especializa a: «Block»
Reglas:

C5. Un «FC_Specification» Block debe poseer una «ToleranceSize» Property, y entre 2 y 4 «FeatureUsage» Ports.

```
self.base_Class.allAttributes().getAppliedStereotypes()-> select
(b|b.name= 'ToleranceSize').size()=1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).type.
oclAsType(UML::Class).getAppliedStereotypes()-> select( b|b.name=
'FeatureUsage')).size()>1 and self.base_Class.allAttributes() ->
select(a|a.type.oclIsKindOf( UML::Class)).type.oclAsType(UML::Class).
getAppliedStereotypes()-> select (b|b.name= 'FeatureUsage')).size()<5
```

«GT_Specification»
Especificación de una tolerancia geométrica, definiendo el tamaño de la zona de tolerancia que representa los límites de la variabilidad aceptable de acuerdo con un tipo de relación de posición/orientación impuesta desde unos feautes datum a un feature target.

Extiende/especializa a: «Block»
Reglas:

C6. Un «GT_Specification» Block debe poseer una «ToleranceSize» Property, y entre 2 y 4 «FeatureUsage» Ports.

```
self.base_Class.allAttributes().getAppliedStereotypes()-> select
(b|b.name= 'ToleranceSize').size()=1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).
type.oclAsType(UML::Class).getAppliedStereotypes()-> select
(b|b.name= 'FeatureUsage')).size()>1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).type
.oclAsType(UML::Class). getAppliedStereotypes()-> select (b|b.name=
'FeatureUsage')).size()<5
```

«ToleranceSize»
Propiedad que especifica el tamaño de una zona de tolerancia mediante un valor real.

Extiende/especializa a: <Property>

«FeatureUsage»
Puerto que representa un feature participando en una especificación de tolerancia geométrica, ya sea como referencia (datum) o como objetivo (target). Para definir el rol, el estereotipo «FeatureUsage» tiene un atributo ("role") tipeado por la enumeración RoleTypes, que incluye los EnumerationLiteral (datum, target). Además, el estereotipo «FeatureUsage» tiene otro atributo ("order") tipeado por la enumeración OrderOptions, que incluye los EnumerationLiteral (none, FirstRef, SecondRef, ThirdRef) con el fin de definir el orden de referencia, en caso de actuar como datum.

Extiende/especializa a: <Port>
Atributos:
- role : RoleTypes [1]
- order : OrderOptions [1]

Reglas:

C7. Si el «FeatureUsage» Port es un feature objetivo (role=target), no se puede definir ningún orden (order=none), mientras que si se trata de un «FeatureUsage» Port que actúa como referencia (role=datum), su orden debe ser definido (order≠none).
```
role='target' implies order='none' and role='datum' implies not
order='none'
```

«Assembly Feature»
Puerto que representa un feature ofrecido por un artefacto para participar en una relación de ensamble.

Extiende/especializa a: <Port>
Reglas:

C8. Un «Assembly Feature» Port debe estar tipeado por un «Feature» Block.
```
self.base_Class.allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class). getAppliedStereotypes()->
select (b|b.name= 'Feature'))->notEmpty()
```

«F_TTRS_Repr»
Representación de un feature basada en el modelo TTRS, incluyendo sus MGREs definidos respecto a los sistemas de referencia local y global, relacionados a través de una transformación.

Extiende/especializa a: «Block»
Reglas:

C9. Debe poseer: a) un «MGRE_repr» Block, representando la definición local; b) un behavior Port tipeado por un «MGRE_repr» Block, representando la definición global; c) una ConstraintBlock que de soporte a la transformación entre los sistemas de referencia local y global; d) un «TM_Port» behavior Port, que posibilita el uso de la matriz de transformación definida en el artefacto poseedor («Part» Block).
```
self.base_Class.allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class). getAppliedStereotypes()->
select (b|b.name= 'MGRE_repr')).size()=2 and self.base_Class.
ownedAttributes ->select(c:Port|c.type.oclIsKindOf(UML::Class)).
type.oclAsType(UML::Class).getAppliedStereotypes()-> select
(d|d.name= MGRE_repr')).size()=1 and self.base_Class.allAttributes()
->select(e|e.type. oclIsKindOf( UML::Class)).type.oclAsType(
```

```
UML::Class).getAppliedStereotypes()-> select (b|b.name=
'ConstraintBlock')).size=1 and self.base_Class.allAttributes()->
select(e|e.type. oclIsKindOf( UML::Class)).type.oclAsType(
UML::Class).getAppliedStereotypes()-> select (b|b.name=
'TM_Port')).size()=1
```

«MGRE»
Elemento geométrico mínimo de referencia para una superficie de acuerdo a su clase de invarianza. Existen siete tipos de MGRE, que serán definidos a nivel de librería, y cada uno de ellos está caracterizado por un conjunto de RGEs.

Extiende/especializa a: «Block»
Reglas:

C10. Un «MGRE» Block debe poseer entre uno y tres usos de «RGE» Blocks.

```
self.base_Class.allAttributes()-> select(e|e.type. oclIsKindOf(
UML::Class)).type.oclAsType( UML::Class).getAppliedStereotypes()->
select (b|b.name= 'RGE')).size()=1 and self.base_Class.
allAttributes()-> select(e|e.type. oclIsKindOf( UML::Class)).
type.oclAsType( UML::Class).getAppliedStereotypes()-> select
(b|b.name= 'RGE')).size()=<4
```

C11. Un «MGRE» Block debe poseer tantos usos de «RGE Constraint» ConstraintBlocks como el número de RGEs menos 1.

```
base_Class.ownedAttributes->select(a|a.type.getAppliedStereotypes()->
select (b|b.name= 'RGE_Constraint')).size()=base_Class.
ownedAttributes->select(a|a.type.getAppliedStereotypes()-> select
(b|b.name= 'RGE')).size-1
```

«MGRE_repr»
Representación de un feature basada en el modelo TTRS, incluyendo tanto la definición nominal como desviada.

Extiende/especializa a: «Block»
Reglas:

C12. Un «MGRE_repr» Block debe contener dos «MGRE_usage» behavior Ports para representar la definición nominal y desviada.

```
self.base_Class.allAttributes()-> select(e|e.type. oclIsKindOf(
UML::Class)).type.oclAsType( UML::Class).getAppliedStereotypes()->
select (b|b.name= 'MGRE_usage')).size()=1
```

«MGRE_usage»
Puerto de comportamiento (behavior Port) que define un MGRE, diferenciando su uso para soportar la definición nominal o desviada de una geometría. El estereotipo cuenta con un atributo (isDeviated) que distingue entre su uso para definir geometrías nominales (false) o desviadas (true)

Extiende/especializa a: <Port>
Atributos:

- isDeviated : Boolean [1]

«RGE»
Bloque que representa un elemento geométrico reducido (concepto del modelo TTRS) que es utilizado para construir un MGRE. Pueden ser puntos, líneas o planos, pero estas especializaciones se definirán a nivel de librería.

Extiende/especializa a: «Block»

«TTRS»
Entidad que representa un par de MGREs relacionados, definiendo restricciones de posición/orientación.

Extiende/especializa a: «Block»
Reglas:

C13. Un «TTRS» Block debe poseer: a) dos puertos tipeados por «MGRE» Blocks; b) dos usos (parts) tipeados por «MGRE» Blocks; c) al menos una «RGE_Constraint» ConstraintBlock.

```
self.base_Class.allAttributes()-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType( UML::Class).getAppliedStereotypes()->
select (b|b.name= 'MGRE')).size()=4 and self.base_Class.
allAttributes()-> select(e:Port|e.type. oclIsKindOf( UML::Class)).
type.oclAsType( UML::Class).getAppliedStereotypes()-> select
(b|b.name= 'RGE')).size()=2 and self.base_Class.allAttributes()->
select(e|e.type. oclIsKindOf( UML::Class)).type.oclAsType(
UML::Class).getAppliedStereotypes()-> select (b|b.name=
'RGE_Constraint')).size()<0
```

«RGE_Constraint»
Restricción geométrica entre dos RGEs, correspondiente a uno de los 13 casos simples propuestos por el modelo TTRS. Estos 13 casos está definidos a nivel de librería.

Extiende/especializa a: «ConstraintBlock»
Reglas:

C14. Un «RGE_Constraint» ConstraintBlock debe poseer al menos dos parámetros tipeados por «RGE» Blocks.

```
self.base_Class.allAttributes()-> select(e|e.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
select (b|b.name= 'RGE')).size()>1
```

«Representation»
Propiedad que representa otro elemento del modelo desde un punto de vista diferente. El estereotipo «Representation» cuenta con un atributo ("representsTo") para definir el elemento al que está representando.

Extiende/especializa a: <Property>
Atributos:

- representsTo : Property [1]

«AssemblyMGRE»
Puerto que representa una geometría ofrecida por un artefacto para participar en una relación de ensamble, siendo definida en base al modelo TTRS.

Extiende/especializa a: <Port>

Reglas:

C15. Un «AssemblyMGRE» Port debe estar tipeado por un «F_TTRS_Repr» Block.

```
self.base_Port.type.oclAsType(UML::Class).getAppliedStereotypes().nam
e-> includes('F_TTRS_Repr')
```

«TransformationMatrix»

Matriz homogénea que soporta la traslación y rotación de definiciones de geometría para cambiar el sistema de referencia considerado. Se trata de una propiedad característica de una pieza, de manera que cualquier geometría perteneciente a dicha pieza debe ser transformada aplicando la misma matriz.

Extiende/especializa a: <Property>

Reglas:

C16. El «Part» Block poseedor debe contener únicamente una «TransformationMatrix» Property.

```
self.base_Property.owner.oclAsType( UML::Class).allAtributes()-
>select(a|a.getAppliedStereotypes()name-> include(
'TransformationMatrix')).size()=1
```

«TM_Port»

Puerto de comportamiento (behavior Port) que permite conectar cada uso de un «F_TTRE_repr» Block con la matriz de transformación característica de la pieza a la que pertenece.

Extiende/especializa a: <Port>

«GT_TTRS_definition»

Definición matemática de una tolerancia geométrica entre representaciones de features basadas en TTRS. Debe poseer los elementos suficientes como para definir las geometrías de referencia (datum), la definición de la zona de tolerancia a través de relaciones matemáticas, y la definición de una geometría objetivo (target) dentro de dicha zona de tolerancia. Además, se incluyen mecanismos para validar la especificación y asegurar la consistencia de las tolerancias geométricas definidas.

Extiende/especializa a: «Block»

Reglas:

C17. Un «GT_TTRS_definition» Block debe poseer un uso de una a «TZ_TTRS_definition» Block, y entre 2 y 4 «TTRS_Repr_usage» Ports.

```
self.base_Class. allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
select (b|b.name='TZ_TTRS_definition').size()=1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).
type.oclAsType(UML::Class).getAppliedStereotypes()-> select
(b|b.name= ' TTRS_Repr_usage')).size()>1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).type
.oclAsType(UML::Class). getAppliedStereotypes()-> select (b|b.name= '
TTRS_Repr_usage')).size()<5
```

C18. Un «GT_TTRS_definition» Block debe poseer el mismo número de propiedades tipeadas por «F_TTRS_Repr» Blocks que «TTRS_repr_usage» Ports.

```
self.base_Class. allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
```

```
select (b|b.name='TZ_TTRS_definition').size()= self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).type.
oclAsType(UML::Class).getAppliedStereotypes()-> select (b|b.name=
'TTRS_repr_usage').size()
```

C19. Un «GT_TTRS_definition» Block debe poseer la misma cantidad de propiedades tipeadas por «TTRS» Blocks o más que el doble del número de «TTRS_repr_usage» Ports que representen referencias (role=datum), con el fin de definir tanto las relaciones entre geometrías nominales como desviadas.

```
self.base_Class. allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
select (b|b.name='TTRS').size()>= 2*self.base_Class. allAttributes()
-> select(a|a.type.oclIsKindOf( UML::Class)).type. oclAsType(
UML::Class).getAppliedStereotypes()-> select (b|b.name=
'TTRS_repr_usage')-> select (c|c.role= 'datum').size()
```

C20. Un uso de un «GT_TTRS_definition» Block debe poseer dos propiedades tipeadas por «SpecifValidation» y «TargetConstruction» ConstraintBlocks respectivamente, que relacionan la zona de tolerancia con la geometría objetivo (target).

```
self.base_Class. allAttributes() -> select(a|a.type.oclIsKindOf(
UML::Class)).type.oclAsType(UML::Class).getAppliedStereotypes()->
select (b|b.name=' SpecifValidation').size()=1 and self.base_Class.
allAttributes() -> select(a|a.type.oclIsKindOf( UML::Class)).type.
oclAsType(UML::Class).getAppliedStereotypes()-> select (b|b.name='
TargetConstruction').size()=1
```

C21. Un «GT_TTRS_definition» Block debe poseer un «TargetDeviation» Property en el que se definen las desviaciones para la geometría construida.

```
self.base_Class. allAttributes().getAppliedStereotypes()-> select
(b|b.name=' TargetDeviation').size()=1
```

«SpecifValidation»
Validación de la consistencia de una especificación, comprobando que el MGRE nominal de la geometría objetivo (target) coincida o esté incluida en el MGRE nominal de la zona de tolerancia (construido a partir de los MGRE nominales de las referencias). El resultado de esta validación se define en un parámetro booleano.

Extiende/especializa a: «ConstraintBlock»
Reglas:

C22. Una «SpecifValidation» ConstraintBlock incluye un parámetro Booleano para transferir el resultado de la validación.

```
self.base_Class. allAttributes()-> select(a|a.type=Boolean)->
notEmpty()
```

«TargetConstruction»
Construcción del MGRE desviado de una geometría objetivo (target), aplicando un vector en el que se almacenan las desviaciones en los diversos grados variantes, dentro de los límites impuestos por la zona de tolerancia (su MGRE y su tamaño),

Extiende/especializa a: «ConstraintBlock»
Reglas:

C23. Un «TargetConstruction» ConstraintBlock tiene un parámetro tipeado por un «DeviationVector» DataType.

```
self.base_Class.allAttributes()->select(a|a.type.oclIsKindOf(UML::
DataType).type.oclAsTypeOf(UML::DataType).getAppliedStereotypes().
name->includes('DeviationVector')
```

«TZ_TTRS_definition»
Espacio geométrico utilizado para controlar/definir la posición/orientación de una geometría objetivo (target). Este espacio geométrico está caracterizado por un tamaño, definido en la especificación de una tolerancia geométrica a la que se accede gracias a una AdjuntProperty.

Extiende/especializa a: «Block»
Reglas:

C24. Un «TZ_TTRS_definition» Block posee dos usos de «MGRE» Blocks, uno de ellos para la definición nominal (isNominal=true) y otra para la definición desviada (isNominal=false).

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes().
name->includes('MGRE') and self.base_Class.allAttributes()->
select(a|a.type.oclIsKindOf( UML::Class).type.oclAsTypeOf(UML::
Class).getAppliedStereotypes()->select(c|c.name='MGRE')-
>select(d|d.isNominal=true).size()=1
```

«TTRS_Repr_usage»
Representación basada en TTRS empleada como una geometría objetivo (target) o referencia (datum) en la definición de una tolerancia geométrica. Para definir el rol, el estereotipo «TTRS_Repr_usage» tiene un atributo ("role") tipeado por la enumeración RoleTypes, que incluye los EnumerationLiteral (datum, target). Además, el estereotipo «TTRS_Repr_usage» tiene otro atributo ("order") tipeado por la enumeración OrderOptions, que incluye los EnumerationLiteral (none, FirstRef, SecondRef, ThirdRef) con el fin de definir el orden de referencia, en caso de actuar como datum.

Extiende/especializa a: <Port>
Atributos:

- role : RoleTypes [1]
- order : OrderOptions [1]

Reglas:

C25. Si el «TTRS_Repr_usage» Port es un feature objetivo (role=target), no se puede definir ningún orden (order=none), mientras que si se trata de un «FeatureUsage» Port que actúa como referencia (role=datum), su orden debe ser definido (order≠none).

```
role='target' implies order='none' and role='datum' implies not
order='none'
```

«TargetDeviation»
Propiedad que expresa las desviaciones en los grados variantes de un MGRE target.

Extiende/especializa a: <Property>

«DeviationVector»
DataType que expresa los pequeños desplazamientos de un MGRE desviado.
Extiende/especializa a: <DataType>

Anexo C

A continuación, se presenta la especificación de los estereotipos definidos en el perfil SysML for Geometric Deviation Propagation Simulation (SysML4GDPSim).

«DevProduct_sim»
Bloque que especializa el estereotipo «ProductSim» para dar incluir los datos que definen las desviaciones geométricas del producto.

Extiende/especializa a: «ProductSim»
Reglas:

C1. Contiene un vector de desviaciones de tipo «ProductDeviations» DataType

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf( UML::
DataType).type.oclAsTypeOf(UML::DataType).getAppliedStereotypes().
name->includes('ProductDeviations')
```

«DevProduct_sim»
Especialización del estereotipo «DeviationVector» empleado para definir las desviaciones geométricas en un «DevProduct_sim» Block.

Extiende/especializa a: «DeviationVector»

«DevProductSpecif_data»
Bloque que especializa el estereotipo «ProductSpecif_data» para dar incluir dos o más artefactos que describen físicamente el producto en cada uno de los estados por los que pasa.

Extiende/especializa a: «ProductSpecif_data»
Reglas:

C2. Contiene al menos dos artefactos de producto.

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf( UML::
Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes()-
>select(b|b.name='ProductArtifact').size()>1
```

«ProductArtifact»
Bloque que especializa el estereotipo *«Artifact»* para dar soporte a la definición geométrica de un producto, ya sea una pieza o un ensamble.

Extiende/especializa a: «Artifact»
Reglas:

C3. El bloque debe estar estereotipado también por «Part» o «Assembly».

```
self.base_Class.getAppliedStereotypes().allParents().name-
>includes('Artifact')
```

«DevTransformResource_sim»
Bloque que especializa el estereotipo «ManufResourceSim» para añadir información necesaria para llevar a cabo el análisis de las desviaciones geométricas.

Extiende/especializa a: «TransformResource:sim»
Reglas:

C4. Posee una propiedad tipeada por un «ProcessAssembly»

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes().
name->includes('ProcessAssembly')
```

C5. Posee al menos dos «Dev_FU_Port» Ports

```
self.base_Class.allAttributes()-> select(a:Port|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes()-
>select(b|b.name='MGRE').size()=2
```

«ResourceDeviations»
Especialización del estereotipo «DeviationVector» empleado para definir las desviaciones geométricas en un «DevTransformResource_sim» Block.

Extiende/especializa a: «DeviationVector»

«DevManufResourceSpecif_data»
Bloque que especializa el estereotipo «ManufResourceSpedif_data» para añadir información necesaria para llevar a cabo el análisis de las desviaciones geométricas.

Extiende/especializa a: «ManufResourceSpedif_data»
Reglas:

C6. Posee una propiedad tipeada por un «ConfiguratedMachineArtifact»

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes().
name->includes('ProcessAssembly')
```

«ConfiguratedMachineArtifact»
Bloque que especializa el estereotipo «Artifact» para representar específicamente una máquina configurada, es decir, el montaje formado por un equipo de fabricación (máquina) y los utillajes y/o herramientas necesarias para realizar un determinado proceso de fabricación.

Extiende/especializa a: «Artifact»
Reglas:

C7. El bloque debe estar estereotipado también por «Part» o «Assembly».

```
self.base_Class.getAppliedStereotypes().allParents().name->
includes('Artifact')
```

«ProcessAssembly»
Bloque que especializa el estereotipo «Assembly» para representar específicamente un ensamble de proceso, es decir, aquel compuesto por una máquina configurada y un producto.

Extiende/especializa a: «Assembly»
Reglas:

C8. Está compuesto por un uso de un «ConfiguratedMachine» Block y un uso de un «ProductArtifact» Block.

```
self.base_Class.allAttributes()-> select(a|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes().
name->includes('ConfiguratedMachine') and self.base_Class.
allAttributes()-> select(a|a.type.oclIsKindOf( UML::Class).type.
oclAsTypeOf(UML::Class).getAppliedStereotypes(). name-
>includes('ProductArtifact')
```

Anexo D

A continuación, se presenta la especificación de los estereotipos definidos en el perfil Model-to-Model Transformation to Modelica (SysML4MMT2Modelica).

«M_Class»
Estereotipo abstracto que sirve como generalización para representar cualquier clase Modelica. Posee dos atributos booleanos para indicar si se trata de una clase parcial y si está modelada en Modelica como elemento de una librería predefinida.

Atributos:
- isPartial : Boolean
- ModelicaLibraryElement : Boolean

«M_Model»

Especialización de M_Class y de Block que representa un modelo Modelica.

Extiende/especializa a: *«M_Class»* / «Block»

«M_Block»
Especialización de M_Model que representa un bloque Modelica, es decir, un modelo en el que todos los «M_ConnectorUse» definidos deben estar definidos como “input” o “output”.

Extiende/especializa a: «M_Model»
Reglas:

C1. Todos los «M_ConnectorUse» definidos deben tener en este estereotipo el atributo isInput o isOutput verdadero.

```
self.base_Class.allAttributes()-> select(a:Port|a.type.oclIsKindOf(
UML::Class).type.oclAsTypeOf(UML::Class).getAppliedStereotypes()->
select(b|b.name='M_ConnectorUse')->select(b|b.isInput=false and
b.isOutput=false).isEmpty()
```

«M_Experiment»
Extensión de la metaclase InstanceSpecification que instancia un «M_Model», y que tiene un Slot por cada «M_Parameter» del modelo.

Extiende/especializa a: <InstanceSpecification>
Reglas:

C2. Tiene tantos Slots como propiedades estereotipadas con «M_Parameter» tiene el «M_Model» al que instancia.

```
Self.base_InstanceSpecification.solts.size()=self.base_InstanceSpecif
ication.classifier.oclAsTypeOf(UML::Class).allAtributes()->
```

```
select(a|a.getAppliedStereotypes()->select(b|b.name='M_Parameter')).
size()
```

«M_ConnectorDef»
Especialización de M_Class y de Block aplicable a bloques en los que se define un puerto estereotipado como «M_ConnectorUse»

Extiende/especializa a: *«M_Class»* / «Block»

«M_Package»
Especialización de M_Class y extensión de la metaclase Package aplicable a paquetes SysML que representan paquetes Modelica.

Extiende/especializa a: *«M_Class»* / <Package>

«M_Type»
Especialización de M_Class y extensión de la metaclase Enumeration aplicable a enumeraciones SysML en las que se define un "type" deModelica.

Extiende/especializa a: *«M_Class»* / <Enumeration>

«M_Record»
Especialización de M_Class y extensión de la metaclase DataType para definir una clase Modelica de tipo "record".

Extiende/especializa a: *«M_Class»* / <DataType>

«M_Function»
Especialización de M_Class y extensión de la metaclase FunctionBehavior para definir, como expresión opaca, un cálculo algorítmico sobre unas entradas, obteniendo unas salidas.

Extiende/especializa a: *«M_Class»* / <FunctionBehavior>

«M_Property»
Estereotipo abstracto que sirve como generalización para representar cualquier tipo de propiedad Modelica.

Atributos:
- isInner : Boolean
- isOuter : Boolean
- isReplaceable : Boolean

«M_Part»
Estereotipo abstracto que especializa el M_Property que sirve como generalización para representar cualquier tipo de propiedad parte o puerto (usos de modelos o conectores Modelica).

Extiende/especializa a: *«M_Property»*

«M_Component»
Especialización de M_Part para definir una propiedad parte de un bloque «M_Model» o «M_Block».

Extiende/especializa a: *«M_Part»* / <Property>
Reglas:

C3. Su tipo debe ser un bloque «M_Model» o «M_Block».

```
self.base_Property.type.oclAsTypeOf(UML::Class).
getAppliedStereotypes().name->includes('M_Model') or self.
base_Property.type. oclAsTypeOf(UML::Class).
getAppliedStereotypes().name-> includes('M_Block')
```

«M_ConnectorUse»
Especialización de M_Part para definir un puerto, cuyo tipo se define en un bloque «M_ConnectorDef».

Extiende/especializa a: *«M_Part»* / <Port>
Reglas:

C4. Su tipo debe ser un bloque «M_ConnectorDef».

```
self.base_Property.type.oclAsTypeOf(UML::Class).
getAppliedStereotypes().name->includes('M_ConnectorDef')
```

«M_Variable»
Especialización de M_Property y extensión de la metaclase Property que se utiliza para definir una variable en Modelica, identificando si tiene un comportamiento discreto.

Extiende/especializa a: *«M_Property»* / <Property>
Atributos:

- isDiscrete : Boolean

«M_Parameter»
Especialización de M_Property y extensión de la metaclase Property que se utiliza para definir un parámetro en un modelo Modelica.

Extiende/especializa a: *«M_Property»* / <Property>

«M_Function»
Extensión de la metaclase FunctionBehavior que se utiliza para definir una función Modelica como una expresión opaca.

Extiende/especializa a: <FunctionBehavior>

«M_Equation»
Extensión de la metaclase Constraint que se utiliza para definir el fragmento de ecuaciones en un modelo Modelica como una expresión opaca.

Extiende/especializa a: <Constraint>

«M_Connection»

Especialización del estereotipo «BindingConnector» que se utiliza para conectar propiedades estereotipadas con especializaciones de «M_Part».

Extiende/especializa a: «BindingConnector»

Anexo E

A continuación, se presenta el código EGL que define el algoritmo utilizado en la transformación de modelos SysML a texto Modelica bajo la consideración de disponer de librerías equivalentes en ambos lenguajes. Con el fin de facilitar la lectura e interpretación del código, se numeran las líneas de código ya que, por su longitud, no siempre coinciden con líneas del documento.

```
1. [% var paquetes==m!Package.all.select(a|a.getAppliedStereotypes().exists(
   b|b.name=='M_Package'));%]
2. [% var paquetePrincipal=m!Package.all.select(
   e|e.getAppliedStereotypes().exists( s|s.name=='RootModelicaPackage')).first;%]
3. [% var elementos=paquetePrincipal.packagedElement;%]
4. [% var subPaquetes=elementos.select(
   a:Package|a.getAppliedStereotypes().exists( s|s.name=='M_Package'));%]
5. [% var equations=m!Constraint.all.select(a|a.getAppliedStereotypes().exists(
   s|s.name=='M_Equation'));%]
6. [%=equations.name%]
7. [%=equations.first.qualifiedName.type%]
8. [%=writePackage(paquetePrincipal)%]
9. [%  %]
10.[%  %]
11.[% operation writePackage(p:Package): String {%]
12.package [%=p.name%]
13.[% var subPaquetes=p.packagedElement.select(
   a:Package|a.getAppliedStereotypes().exists(s|s.name=='M_Package'));%]
14.[% var modelos=p.packagedElement.select(
   a|a.getAppliedStereotypes().exists(s|s.name=='M_Model'));%]
15.[% var tipos=p.packagedElement.select(
   a|a.getAppliedStereotypes().exists(s|s.name=='M_Type'));%]
16.[% var records=p.packagedElement.select(
   a|a.getAppliedStereotypes().exists(s|s.name=='M_Record'));%]
17.[% var functions=p.packagedElement.select(
   a|a.getAppliedStereotypes().exists(s|s.name=='M_Function'));%]
18.[% var experiments=p.packagedElement.select(
   a|a.getAppliedStereotypes().exists(s|s.name=='M_Experiment'));%]
19.[% for (subP in subPaquetes){%]
20.[%=writePackage(subP)%]
21.[%}%]
22.[% for (mod in modelos){%]
23.[%=writeModel(mod)%]
24.[%}%]
25.[% for (t in tipos){%]
26.[%=writeType(t)%]
27.[%}%]
28.[% for (r in records){%]
29.[%=writeRecord(r)%]
```

```
30. [%}%]
31. [% for (f in functions){%]
32. [%=writeFunction(f)%]
33. [%}%]
34. [% for (e in experiments){%]
35. [%=writeExperiment(e)%]
36. [%}%]
37. end  [%=p.name%];
38. [%return "";%]
39. [% } %]
40. [%  %]
41. [%  %]
42. [% operation writeModel(m:Class): String {%]
43. model [%=m.name%]
44. [% if (m.parents->notEmpty) {var direccionPadre : String =
    m.parents.first.qualifiedName;var nuevaDireccionPadre:String=
    direccionPadre.replaceAll("::", ".");%]
45. extends [%=nuevaDireccionPadre%]; [% }%]
46. [% var parameters=m.features.select(
    a|a.getAppliedStereotypes().exists(s|s.name=='M_Parameter'));%]
47. [% for (param in parameters){%]
48. [%=param.type.name%][%=param.name%][%if (param.lower>1) {%]
    [[%=param.lower%]][%}%];
49. [%}%]
50. [% var variables=m.features.select(
    a|a.getAppliedStereotypes().exists(s|s.name=='M_Variable'));%]
51. [% for (variab in variables){%]
52. [%=variab.type.name%] [%=variab.name%][%if (variab.lower>1) {%]
    [[%=variab.lower%]][%}%];
53. [%}%]
54. [% var componentes=m.features.select(
    a|a.getAppliedStereotypes().exists(s|s.name=='M_Component'));%]
55. [% for (comp in componentes){%] [%var direccionTipo : String=
    comp.type.qualifiedName; var nuevaDireccionTipo:String=
    direccionTipo.replaceAll("::", ".");%]
56. [%=nuevaDireccionTipo%] [%=comp.name%];
57. [%}%]
58. [% var puertos=m.features.select(
    a|a.getAppliedStereotypes().exists(s|s.name=='M_ConnectorUse'));%]
59. [% for (p in puertos){%] [%var direccionTipo : String =p.type.qualifiedName;
    var nuevaDireccionTipo:String=direccionTipo.replaceAll("::", ".");%]
60. [%=nuevaDireccionTipo%] [%=p.name%];
61. [%}%]
62. equations
63. [% var m_equations=equations.select(
    a|a.constrainedElement.first.name=m.name);%]
64. [% for (eq in m_equations){%]
65. [%=eq.specification.body%]
66. [%}%]
67. end  [%=m.name%];
a.  [%return "";%]
68. [% } %]
69. [% %]
70. [% operation writeType(t:Enumeration): String {%]
71. [% var contador : Integer = t.ownedLiteral.size();%]
72. type [%=t.name%] = enumeration([% for (lit in t.ownedLiteral){%][%=lit.name%]
    [%if (contador>1) {%], [%} else{} contador=contador-1;} %]);
73. end  [%=t.name%];
74. [%return "";%]
```

```
75. [% } %]
76. [% %]
77. [% operation writeRecord(r:DataType): String {%]
78. record [%=r.name%]
79. [% var variables=r.features.select(
    a|a.getAppliedStereotypes().exists(s|s.name=='M_Variable'));%]
80. [% for (variab in variables){%]
81. [%=variab.type.name%] [%=variab.name%][%if (variab.lower>1) {%]
    [[%=variab.lower%]][%}%];
82. [%}%]
83. end  [%=r.name%];
84. [%return "";%]
85. [% } %]
86. [% %]
87. [% operation writeFunction(f:FunctionBehavior): String {%]
88. function [%=f.name%]
89. [%=f.body%]
90. end  [%=f.name%];
91. [%return "";%]
92. [% } %]
93. [% operation writeExperiment(e:InstanceSpecification): String {%]
94. model [%=e.name%]
95. [%var direccionModelo : String =e.classifier.first.qualifiedName;var
    nuevaDireccionModelo:String=direccionModelo.replaceAll("::", ".");%]
96. [%=nuevaDireccionModelo%] [%=e.name%]([%=writeValues(e, false, '')%]);
97. end  [%=e.name%];
98. [%return "";%]
99. [% } %]
100.  [% operation writeValues(e:InstanceSpecification,
    pre:Boolean,preText:String): String {%]
101.  [%var slt : Set =e.slot;%][% var cont : Integer = slt.size();%] [%for (s in
    slt){%][%if (pre){%][%=preText%].[%}%][%=s.definingFeature.name%][%if
    (s.value.first.isComputable){%]=[%=s.value.first.value%][%} else
    {%].[%=s.value.first.instance.name%][%=writeValues(s.value.first.instance,
    true,s.definingFeature.name)%], [%}%][%}%]
102.  [%return "";%]
103.  [% } %]
```